Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
MSXML 5.0 SDK

What is MSXML?

Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office allows customers to build high-performance XML-based applications that provide a high degree of interoperability with other applications that adhere to the XML 1.0 standard.

Among the core services MSXML 5.0 provides is developer support for the following:

- The Document Object Model (DOM), a standard library of application programming interfaces (APIs) for accessing XML documents.
- The XML Schema definition language (XSD), a current W3C standard for using XML to create XML Schemas. XML Schemas can be used to validate other XML documents.
- The Schema Object Model (SOM), an additional set of APIs for accessing XML Schema documents programmatically.
- Extensible Stylesheet Language Transformations (XSLT) 1.0, a current W3C XML style sheet language standard. XSLT is recommended for transforming XML documents.
- The XML Path Language (XPath) 1.0, a current W3C XML standard used by XSLT and other XML programming vocabularies to query and filter data stored in XML documents.
- The Simple API for XML (SAX), a programmatic alternative to DOM-based processing.

What's New?

This release of the MSXML SDK provides the following new sections and topics.

- Program with DOM in C/C++
- Program with DOM in C/C++ Using Smart Pointer Class Wrappers
For information about new product features in MSXML 5.0, see What's New.

Important Notices About MSXML and This SDK

- Roadmap to the MSXML SDK
- Installing and Registering the MSXML SDK
- GUID and ProgID Information
- Dependencies in MSXML 5.0

Tips and FAQ

- Tips for Converting Samples to VBScript
- FAQ About Schemas
- FAQ About the SOM
- FAQ About XSLT
- FAQ About SAX2

Feedback and Support

Send error reports, feature requests, and comments about the MSXML documentation or samples directly to the MSXML documentation team.

For free help with MSXML issues, try posting to the MSXML public newsgroup. This newsgroup is monitored by Microsoft Product Support Services (PSS) engineers who cover MSXML, and by other experienced MSXML developers.

Further information about support options can be found on the Microsoft Help and Support Web site.

Other Resources

Microsoft Data Access Components (MDAC) | Web Services, part of the
Platform SDK documentation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Roadmap to the MSXML SDK

SDK Objectives

This Software Development Kit (SDK) provides documentation for developers writing applications with Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office. Before using the SDK, you should be aware of certain goals, non-goals, and user prerequisites.

Goals of the SDK

The primary goal of this SDK is to provide experienced developers with the conceptual and reference information they need to develop applications with MSXML.

Non-Goals of the SDK

This SDK is not intended to be an in-depth tutorial on XML or on any of its related standards, and does not replace the World Wide Web Consortium (W3C) specifications for these standards. (The Other Resources section below provides links to these standards on the Internet.)

This SDK is not intended to demonstrate every possible use of MSXML, and does not document every known problem with MSXML. See Help and Support, below, for links to support resources.

User Prerequisites

To use this SDK, you should be familiar with accessing and using COM components from languages such as Visual Basic, C/C++, and JScript. You should also be familiar with XML and its related standards.
Components of the SDK

The MSXML SDK contains two main types of sections: Developer's Guides and Reference sections. Developer's Guides introduce the core technologies of MSXML, and provide conceptual and procedural information, including standalone samples. Reference sections provide detailed information about MSXML objects, interfaces, properties, and methods, including syntax in Visual Basic, C/C++, and JScript. Snippets and other short samples are also included.

Note Before you run code samples from this SDK, ensure that you have MSXML version 5.0 installed on your machine by searching for msxml5.dll in your system directory. If you have the wrong version of MSXML installed, you might receive an error.

The following is a brief description of each Developer's Guide and Reference section in the MSXML SDK.

XML Developer's Guide

This guide provides introductory information about XML, a basic XML tutorial, and conceptual topics including a document map, namespace information, and security information.

DOM Developer's Guide

This section introduces the key concepts and terminologies of the Microsoft implementation of the XML Document Object Model (DOM) and the MSXML DOM Helper APIs. This guide describes how to work with document parts, and provides tutorials for performing basic DOM programming tasks in Visual Basic, C++, and JScript.

DOM Reference

This section provides reference information for experienced developers. This includes syntax and usage information about DOM objects, interfaces, properties, and methods, and about the MSXML Helper APIs for working with the DOM.
XML Digital Signature Reference

This section provides reference information, including samples, about the MSXML APIs for XML digital signatures. These APIs were first implemented in MSXML 5.0, in accordance with the W3C Recommendation 12 February 2002 specification.

XSD Schema Developer's Guide

This section provides conceptual information about XML Schema definition language (XSD) schemas. XSD is the current World Wide Web Consortium (W3C) specification for XML schemas. Major topics in this section include validation, data types, and structures. The XSD Schema Developer's Guide also contains the XDR Schema Developer's Guide. XML-Data Reduced (XDR) schema is an interim schema language offered to developers while the W3C worked on a draft implementation. MSXML versions 4.0 and later support both XSD and XDR.

XML Schema Reference (XSD)

This section provides reference information about XSD schema elements, data types, and regular expressions.

XDR Schema Reference

This section provides reference information about XDR schema elements and data types.

SOM Developer's Guide

This section introduces the Schema Object Model (SOM), a set of classes for extracting information from XSD schemas. The section includes suggestions for practical uses of the SOM and three sample applications.

SOM Reference

This section provides reference information about the interfaces, properties, and methods of the SOM.

XSLT Developer's Guide
This section introduces XSL Transformations (XSLT) features and the XSLT process, describes various types of transformations, and demonstrates basic XSLT tasks.

**XSLT Reference**

This section provides reference information about XSLT elements, functions, and syntax.

**XPath Developer's Guide**

This section introduces the XML Path Language (XPath) for addressing parts of an XML document. The section includes an XPath tutorial application, and demonstrates basic XPath tasks.

**XPath Reference**

This section provides reference information about XPath syntax and functions.

**SAX2 Developer's Guide**

This section introduces the Microsoft implementation of the latest version of the Simple API for XML (SAX2). The section includes a JumpStart SAX application, as well as several more substantial sample applications that demonstrate common SAX tasks.

**SAX2 Reference**

This section provides reference information about the SAX interfaces, properties, and methods. Also included are SAX common notices for C/C++ and Visual Basic.

**Help and Support**

For free help with MSXML issues, try posting to the MSXML public newsgroup. This newsgroup is monitored by Microsoft Product Support Services (PSS) engineers who cover MSXML, and by other experienced MSXML developers.

Further information about support options can be found on the Microsoft Help
Other Resources

XML Recommendation
Namespaces in XML Recommendation
DOM Level 1 Specification
XML-Signature Syntax and Processing
XML Schema Part 0: Primer
XML Schema Part 1: Structures
XML Schema Part 2: Datatypes
XSLT Recommendation
XPath Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
What's New

Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office provides several new features not available in previous production releases of MSXML.

- **XML Digital signature**

  MSXML 5.0 implements support for XML digital signature, allowing you to sign and verify XML data in DOM documents, DOM nodes, or SAX streams. For more information, see [XML Digital Signature](#).

- **Validation of DOM nodes**

  In earlier versions of MSXML, only full DOM documents could be validated. MSXML 5.0 now provides the ability to validate a single DOM node as an XML document fragment against the currently loaded DTD, schema, or schema collection. For more information, see the `validateNode` method.

- **Import DOM nodes**

  MSXML 5.0 provides the means to clone a node from one DOM document, so it can be added later into another DOM document using the `appendChild` method. For more information, see the `importNode` method.

- **Additional second-level DOM properties**

  With MSXML 5.0, more second-level DOM properties have been introduced that enable you to use the `setProperty` and `getProperty` methods to more closely monitor or configure settings that affect XML parsing behavior. For more information, see [Second-level properties](#).

- **Standalone SAX XSD validator**

  A new standalone XSD validator, the `mxValidator` CoClass, has been added to MSXML 5.0 for use in SAX applications. For more information, see the [MXValidator CoClass](#).
• Modified support for handling inline schemas

MSXML 5.0 now includes new options in the DOM and SAX APIs that allow for support of inline XSD schemas. For more information, see Using Inline Schemas (XSD).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Installing and Registering the MSXML 5.0 SDK

Development Environment

The following system components are required for developing with Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office.

- Windows-compatible computer
- A supported 32-bit version of a Microsoft Windows® operating system product, such as Microsoft Windows® XP, Microsoft Windows® 2000, Microsoft Windows NT® 4.0, Microsoft Windows® Millennium Edition, or Microsoft Windows 98
- Microsoft Internet Explorer 5.0 or later, if your solutions create output for display on the web browser
- Microsoft Windows Script Host, if you want to view output without the web browser
- Microsoft® Visual Studio®, if you create solutions with Visual Basic® or C++
Installing and Registering MSXML 5.0

The Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office Software Development Kit is installed on your computer in side-by-side mode. This mode protects the quality of applications that currently use the msxml.dll, msxml2.dll, msxml3.dll, or msxml4.dll files. This feature is important because different versions of the Microsoft Windows® operating system and Microsoft Internet Explorer are released with different versions of MSXML.

Side-by-side mode also allows you to decide which version of the parser to use in your code. When you install the MSXML 5.0 SDK from the Microsoft XML Download Center, the setup program installs two files onto your computer: msxml5.dll and either msxml5a.dll (if the computer runs under Windows 9x/ME) or msxml5r.dll (if the computer runs under Windows NT/2000/XP). The setup program registers the msxml5.dll file in the computer's registry.

After you install the MSXML 5.0 SDK, calling applications must use version-dependent CLSIDs and ProgIDs to reference and use msxml5.dll. This ensures stability in your applications. For example, you might have legacy coding that uses version-independent CLSIDs and ProgIDs such as the following JScript code:

```javascript
var xml = new ActiveXObject("Msxml2.DOMDocument");
var xsl = new ActiveXObject("Msxml2.DOMDocument");
```

Even if MSXML 5.0 is deployed to client computers, the previous code would continue to run, and to call an earlier version of MSXML, such as version 2.0, 2.6, or 3.0. The exact version depends on other details, such as whether applications have been installed that are dependent on a specific version of MSXML, and that reconfigure the system to use that version by default. For more information and a list of CLSIDs and ProgIDs, see GUID and ProgID Information.
Earlier Versions of MSXML and the Xmlinst.exe Utility

**Note** The MSXML 5.0 SDK does not provide an `xmlinst.exe` utility.

Earlier versions of MSXML, notably MSXML 2.5 and 3.0, can be installed in replace mode with the help of the `xmlinst.exe` utility. In this mode, the newly installed version of MSXML replaces an older version and becomes the default XML parser. When you call an application using version-independent CLSIDs or ProgIDs, the default XML parser is invoked. This causes compatibility problems in older applications.

For this reason, MSXML 5.0 can only be installed side-by-side. The `xmlinst.exe` utility does not work for it. To access MSXML 5.0 from an application, you must use the version-dependent CLSID or ProgID to instantiate the parser explicitly.
Using Version-Dependent ProgIDs

The following Microsoft JScript® code shows how to create an instance of the DOMDocument object using the version-dependent ProgID, Msxml2.DOMDocument.5.0.

```javascript
var xml = new ActiveXObject("Msxml2.DOMDocument.5.0");
var xsl = new ActiveXObject("Msxml2.DOMDocument.5.0");
```

This ProgID is for MSXML 5.0. For a full example of how to call and use MSXML 5.0 to load and transform XML documents from script contained in a Web page, see Implementing MSXML 5.0 in Web Scripting.

In a similar way, to code to MSXML 2.6, use the following JScript syntax:

```javascript
var xml = new ActiveXObject("Msxml2.DOMDocument.2.6");
var xsl = new ActiveXObject("Msxml2.DOMDocument.2.6");
```

For more information about ProgIDs, see GUID and ProgID Information. For more information about creating the DOMDocument object, see Using DOMDocument.

**Important** Ensure the stability of your applications by updating older version-independent CLSIDs and ProgIDs to MSXML 5.0 version-dependent CLSIDs and ProgIDs. Otherwise, mixing the version-dependent and version-independent CLSIDs or ProgIDs in your code can cause problems with your application.
Limitation of Side-by-Side Mode

Currently, with MSXML 5.0 installed in side-by-side mode, Microsoft® Internet Explorer does not provide full MSXML 5.0 support for data islands. When you create a data island on a Web page and reference it as an object, the browser creates that object based on the version of MSXML used by the browser. For Internet Explorer 5.0 or later, this can be MSXML 2.0 or MSXML 2.5, both of which support Extensible Stylesheet Language (XSL), but not XSLT.

See Also

Dependencies in MSXML 5.0

Other Resources

XML Development Center
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
GUID and ProgID Information

Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office, formerly known as the Microsoft XML Parser, is installed on your computer in side-by-side mode. This means that installing MSXML 5.0 does not replace any previously installed version of the MSXML parser on your computer. This is done to protect the quality of applications that are currently using msxml.dll, msxml2.dll, msxml3.dll or msxml4.dll. Side-by-side mode also allows you to decide which version of the parser to use in your code.

Note  Version-independent GUIDs and ProgIDs are no longer supported in MSXML.

After you install MSXML 5.0, applications that use version-independent ProgIDs continue to run using the latest version of MSXML prior to version 5.0. To use version 5.0, applications must be written to use the appropriate version-dependent class IDs (CLSIDs) and ProgIDs that reference Msxml5.dll.

This ensures stability in your applications. It also enables customers to implement new and improved versioning solutions by using the side-by-side installation technology provided with Windows XP.

- Why Version-Independent GUIDs and ProgIDs Were Removed
- Workarounds to Version Independence
- Assessing the Impact to Your Environment
- MSXML 5.0 GUIDs and ProgIDs
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Why Version-Independent GUIDs and ProgIDs Were Removed

Although version-independent GUIDs and ProgIDs appear simpler for coding, they are not supported in MSXML 5.0 for the following reasons.

- The introduction of Windows XP Side-by-Side installation. This technology allows you to simulate the effects of version-independent ProgIDs and replace mode in a cleaner and more manageable way.
- Customer experience reveals that in previous implementations of MSXML, version independence and replace mode are complicated to manage and can introduce instability in the production environment.

For example, the following types of application errors and support issues can arise. These problems easily outweigh the benefits of version independence and replace mode that were provided by previous versions of MSXML.

- For Active Server Pages (ASP) programmers, using version-independent ProgIDs can result in unintended upgrades of legacy ASP-based sites to the latest version of MSXML. You can avoid this problem by modifying your Web pages to reference version-dependent programming IDs. For more information, see Workarounds to Version Independence.
- In some cases, updating MSXML to use replace mode is intended. More often than not, however, it is too complex to predict the overall impact of a replace mode upgrade on a system. Although some applications might benefit from such an upgrade, other applications might break as a result of it. In many cases, you will not be able to know or fully predict which applications on your system depend on MSXML and will be affected by a replace mode upgrade.
- If a customer relies on the built-in XSLT support of Microsoft® Internet Explorer for wide client-side deployment of a Web-based application, replace mode might hide the fact that some client desktops were not upgraded to the appropriate version of MSXML. Therefore, users running previous versions of MSXML replace mode might experience incomplete or incorrect functionality. This means that end users might experience instability in the application, instead of receiving a notification message to
"Upgrade to current version MSXML."

**Note** Replace mode upgrades are no longer supported in MSXML 4.0 and later versions. The Xmlinst.exe tool (a separate download) is for use with MSXML 3.0. It does not work with later versions of MSXML.

**See Also**

[GUID and ProgID Information](#) | [Assessing the Impact to Your Environment](#) | [MSXML 5.0 GUIDs and ProgIDs](#) | [Dependencies in MSXML 5.0](#) | [MSXML 5.0 and Windows XP](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Workarounds to Version Independence

At least two solutions exist for working around the removal of version independence in MSXML 5.0.

- If you are developing for the Windows XP platform, you must use the new side-by-side installation technology to provide manifests for MSXML 5.0 and your application. These manifests can simulate the effects of version independence or replace mode.
- If you are developing to another Windows platform, you can typically change to version-dependent programming IDs in your Web script or other applications.

The following sections discuss these two situations and some workarounds for each.
**Windows XP Side-by-Side Installation**

If you are developing applications for Windows XP and want to continue to use MSXML version independence, the side-by-side installation technology provided with Windows XP allows individual applications to reference objects in whatever way you want them to.

Windows XP Side-by-Side installation works by using manifests. A manifest is a small XML file that describes a component or components of an application. This allows you to avoid lookups in the Windows registry, and provides efficient side-by-side functionality without affecting the entire Windows environment.

If you are interested in using side-by-side installation to simulate MSXML version independence or replace mode functionality for MSXML 5.0, you should complete the following steps:

1. Manually update the MSXML 5.0 manifest to make the new object imitate older versions.
2. Generate a manifest for your application and edit it to use MSXML 5.0 instead of MSXML 4.0, 3.0, or 2.0.

For more information, see MSXML 5.0 and Windows XP, and the Windows XP Side-by-Side installation documentation.

**Note** If you are running Windows 2000, you might be able use Windows Side-by-Side installation 1.0. However, this option has not been tested with MSXML.
Implementing MSXML 5.0 in Web Scripting

You should be aware that any linking to a style sheet file from an XML file that uses the `<?xsl-stylesheet?>` processing instruction will default to using either MSXML 2.0 or MSXML 3.0. However, you can write either server-side or client-side script to use MSXML 4.0 and later versions instead.

For more information, see [Initiate XSLT in a Script](#).

See Also

[GUID and ProgID Information](#) | [Why Version-Independent GUIDs and ProgIDs Were Removed](#) | [Assessing the Impact to Your Environment](#) | [MSXML 5.0 GUIDs and ProgIDs](#) | [Dependencies in MSXML 5.0](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
### Assessing the Impact to Your Environment

The following table lists possible scenarios for dealing with the removal of version independence in MSXML 5.0. Locate the scenario that best describes your current situation, then read the notes for installing and using MSXML 5.0 in that environment.

<table>
<thead>
<tr>
<th>If you…</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never had MSXML upgraded:</td>
<td>No action is required. MSXML 5.0 (msxml5.dll) will install side-by-side with earlier versions of MSXML without affecting any existing functionality. Earlier versions of MSXML are typically provided with Internet Explorer version 4.01 or more recent versions of Microsoft Windows. Microsoft 2.6 was shipped with SQL Server 2000 and MDAC 2.6.</td>
</tr>
<tr>
<td>Installed MSXML 2.6, 3.0, or 4.0 in either side-by-side or replace mode:</td>
<td>MSXML 3.0 is shipping with a number of products, including Visual Studio.Net, Office XP, and Windows XP. It can also be downloaded and installed separately from the Microsoft Web site. MSXML 4.0 is a separate download from the Microsoft Web site that installs separately. In either case, MSXML 5.0 installs side-by-side without affecting any existing functionality for these versions. All version-independent ProgIDs and GUIDs now point to the MSXML 4.0 Beta 1 release. The Beta 2 release of MSXML 4.0, and the later versions, will be shipped as part of Internet Explorer 5.0.</td>
</tr>
</tbody>
</table>
Installed MSXML 4.0 Beta 1 release in side-by-side mode:

releases that will upgrade it, will not have version-independent IDs. Therefore, uninstalling or replacing MSXML 4.0 Beta 1 might make version-independent IDs inaccessible.

In this case, you need to reregister or reinstall MSXML 2.0, 2.6, or 3.0 to make them work again. To do this:

- Open a command prompt window and type the following commands:
  
  cd %windir%\system32
  regsvr32 msxml3.dll (or msxml2.dll if you used or are restoring a version prior to MSXML 3.0)
  xmlinst.exe (to use MSXML in replace mode)
  regsvr32 msxml4.dll

  **Note** Regsvr32.exe is only available on computers running Windows 2000 or Windows XP, or Windows NT with the current service pack. For Windows 9x/Me computers, you can copy the Regsvr32.exe utility to your system and register it with the commands listed above. You can also avoid repeating the steps to register it by reinstalling a previous version of MSXML instead.

This means that you installed the previous Beta 1 release of MSXML 4.0 and ran the Xmlinst.exe utility. In this case, you have replaced earlier versions of MSXML (such as 2.0, 2.6, and 3.0) for various applications, including Internet Explorer, which are now
using the Beta 1 release.

With MSXML 4.0 Beta 2 and later, this functionality will be lost. The msxml4.dll file will no longer contain objects that are expected by these applications.

To revert to a previous version of MSXML for these applications, you have two options.

**Option 1:** Restore MSXML 2.0 as the default XML parser for applications such as Internet Explorer 5. In this case, you should:

1. Reregister msxml.dll.
2. Reregister msxml3.dll (if you need it).

**Option 2:** Restore MSXML 3.0 in replace mode. This restores default handling for XML files and XSLT style sheets in applications such as Internet Explorer. In this case you should:

1. Remove msxml4*.dll.
2. Reinstall (or reregister) msxml3.dll.
3. Run the Xmlinst.exe utility.
4. Install MSXML 4.0 Beta 2 or later.

**Note** You can override the default XML file handling of Internet Explorer by using script in your Web pages. For more information, see [Workarounds to Version Independence](#).

**See Also**

[GUID and ProgID Information](#) | Why Version-Independent GUIDs and ProgIDs Were Removed | Dependencies in MSXML 5.0 | MSXML 5.0 and Windows XP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
# MSXML 5.0 GUIDs and ProgIDs

The following table lists the version-dependent GUIDs and ProgIDs that can be used to invoke MSXML 5.0 in your applications.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>GUID</th>
<th>Prog</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLSID_DOMDocument50</td>
<td><code>{88d969e5-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.DOMDocument.5.0</td>
</tr>
<tr>
<td>CLSID_DSOControl50</td>
<td><code>{88d969e9-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.DSOControl.5.0</td>
</tr>
<tr>
<td>CLSID_FreeThreadedDOMDocument50</td>
<td><code>{88d969e6-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.FreeThreadedDOMDocument.5.0</td>
</tr>
<tr>
<td>CLSID_MXCanonicalWriter50</td>
<td><code>{88d969f4-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.MXCanonicalWriter.5.0</td>
</tr>
<tr>
<td>CLSID_MXDigitalSignature50</td>
<td><code>{88d969f5-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.MXDigitalSignature.5.0</td>
</tr>
<tr>
<td>CLSID_MXHTMLWriter50</td>
<td><code>{88d969f0-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.MXHTMLWriter.5.0</td>
</tr>
<tr>
<td>CLSID_MXNamespaceManager50</td>
<td><code>{88d969f1-f192-11d4-a65f-0040963251e5}</code></td>
<td>Msxml2.MXNamespaceManager.5.0</td>
</tr>
<tr>
<td>CLSID_MXParser50</td>
<td><code>{bc90bc5e-947b-4bc9-8e7a-7b69d886c6f8}</code></td>
<td>Msxml2.MXParser.5.0</td>
</tr>
<tr>
<td>CLSID</td>
<td>GUID</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>CLSID_MXSXWriter50</td>
<td>e8a4-4b5b-8943-7547bc9ed607</td>
<td>Msxml2.MXSXWriter.</td>
</tr>
<tr>
<td></td>
<td>{88d969f3-f192-11d4-a65f-0040963251e5}</td>
<td></td>
</tr>
<tr>
<td>CLSID_MXValidator50</td>
<td>88d969ef-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.MXValidator.</td>
</tr>
<tr>
<td>CLSID_MXXMLWriter50</td>
<td>88d969ee-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.MXXMLWriter.</td>
</tr>
<tr>
<td>CLSID_SAXAttributes50</td>
<td>ae92950d-56b6-4618-a0b4-72c0701cad2a</td>
<td>Msxml2.SAXAttributes.</td>
</tr>
<tr>
<td>CLSID_SAXSXReader50</td>
<td>88d969ec-8b8b-4c3d-859e-af6cd158be0f</td>
<td>Msxml2.SAXSXReader.</td>
</tr>
<tr>
<td>CLSID_SAXXMLReader50</td>
<td>88d969eb-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.SAXXMLReader.</td>
</tr>
<tr>
<td>CLSID_ServerXMLHTTP50</td>
<td>88d969ea-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.ServerXMLHTTP.</td>
</tr>
<tr>
<td>CLSID_XMLHTTP50</td>
<td>88d969e7-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.XMLHTTP.9.</td>
</tr>
<tr>
<td>CLSID_XMLSchemaCache50</td>
<td>88d969e8-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.XMLSchemaCache.</td>
</tr>
<tr>
<td>CLSID_XSLTemplate50</td>
<td>88d969e9-f192-11d4-a65f-0040963251e5</td>
<td>Msxml2.XSLTemplate.</td>
</tr>
</tbody>
</table>
See Also

GUID and ProgID Information | Why Version-Independent GUIDs and ProgIDs Were Removed | Workarounds to Version Independence | Assessing the Impact to Your Environment | Dependencies in MSXML 5.0 | MSXML 5.0 and Windows XP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Dependencies in MSXML 5.0

The following are potential dependency issues for Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office, formerly known as the Microsoft XML Parser. Developers should be aware of these dependencies when planning their applications.

- MSXML 5.0 and Later with Internet Explorer
- MSXML 5.0 with WinHTTP5
- Referencing MSXML 5.0 within VBA Projects
- MSXML 5.0 and Windows XP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
The default XML parser for current versions of Internet Explorer is either MSXML 2.0 or MSXML 3.0. Internet Explorer currently does not ship with MSXML 4.0 or later as its default XML parser or XSL processor for applying style sheet templates. To use features of these versions of MSXML from within Internet Explorer you must use scripting, when an XML DOM object is instantiated using the appropriate version-specific ProgID.

For more information, see [Initiate XSLT in a Script](#).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
MSXML 5.0 with WinHTTP5

MSXML 5.0 installs and depends upon Microsoft® Windows® HTTP Services (WinHTTP) 5.0. WinHTTP 5.0 is a new server-side component that provides an updated Hypertext Transfer Protocol (HTTP) stack for developing server-based applications for the Web.

MSXML 5.0 uses WinHTTP 5.0 for the ServerXMLHTTP object, a server-side interface for handling HTTP requests. This object differs from the XMLHTTP object, which performs in a similar role as a client-side interface on most computers running Windows. The primary difference between these two interfaces is that XMLHTTP cannot be used in a server environment. This is because it depends upon the WinInet component, which was not designed to provide adequate performance in server-side use.

Because of the dependency between WinHTTP 5.0 and MSXML 5.0, you will probably need to deploy WinHTTP 5.0 with your applications if you are working with the following related technologies:

- Active Server Pages (ASP)
- Internet Server Application Programming Interface (ISAPI) extensions for Internet Information Services (IIS)

If you are developing server-side applications in these contexts with MSXML 5.0, be sure to distribute WinHTTP 5.0 or later with your application.

You should also be aware that if you are planning wide client deployment, you cannot use a component designed to use WinHTTP and MSXML for server-side applications in any client-side applications. This is because the WinHTTP component is not supported in Windows 9x/Me. Therefore, any code reference to the ServerXMLHTTP object will fail when executed on computers running these versions of Windows. For Windows 9x/Me, your application must use the older client-side XMLHTTP object.

**Note**  WinHTTP 5.1 will ship with the forthcoming release of Windows XP Server. This version will work side-by-side with WinHTTP 5.0. You will no longer have to monitor package and deployment issues for WinHTTP when
you develop applications that use this platform.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Referencing MSXML 5.0 within VBA Projects

MSXML 5.0 can be used when working within applications that support Visual Basic for Applications (VBA), such as Microsoft Office 2000 or Microsoft Office XP.

For example, to set a reference to MSXML 5.0 in a VBA project within Word 2000 or Word XP, do the following:

1. Open Word 2000 or Word XP.
2. From the Tools menu, point to Macros, and then select Visual Basic Editor.
3. From the Tools menu in Visual Basic Editor, select References.
4. In the list of available references, select Microsoft XML, v 5.0.
5. Click OK.

Once this reference is made, you can call and use MSXML 5.0 application programming interfaces directly from within your VBA project code modules.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
MSXML 5.0 and Windows XP

To enable developers to better deal with DLL conflicts, Microsoft Windows XP and Windows .NET Server provide new features for installing shared components (also known as assemblies).

For example, a new service pack updated version of MSXML 5.0 when installed in side-by-side mode, will not replace any earlier version of MSXML 5.0 used by another application. Also this type of side-by-side installation allows both versions to be loaded in memory and used simultaneously by separate applications that depend upon the different versions. This represents an improvement over the way globally shared installation of component DLLs worked in earlier versions of Microsoft Windows in which the component was replaced and all applications were required to use the most recent version of the component.
Background on MSXML 5.0 and Side-by-Side Assembly

When MSXML 5.0 is installed on Microsoft® Windows® XP, it is installed as a shared side-by-side assembly. Side-by-side assemblies are:

- One or more .dll files packaged together to install and function as a single shared component.
- Installed by the Windows Installer into the managed assembly store located off of the Windows system root folder.
- Authored so that multiple versions of the component can run at the same time without impacting each other.
- Not registered globally on the system, but are globally available to applications that depend on them.

For Windows XP, you specify all dependencies for your compiled applications or components with manifests. Assembly manifests are used with shared dynamic linked libraries. Application manifests are used with executables.

The table below summarizes some of the key differences between globally shared and side-by-side installations of the MSXML component. The same differences will apply for other Windows shared assemblies as well.

<table>
<thead>
<tr>
<th>Globally shared installation</th>
<th>Side-by-side installation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The only way to install globally shared components under Windows 2000 and earlier versions of Microsoft Windows.</td>
<td>The recommended way to author and install shared components under Windows XP and other future versions of Microsoft Windows.</td>
</tr>
<tr>
<td>Requires higher support costs for maintaining an application over its life span.</td>
<td>Can substantially reduce support costs over the life span of your application.</td>
</tr>
<tr>
<td>Shared components are more impacted by updates.</td>
<td>Shared components are less impacted by updates.</td>
</tr>
<tr>
<td>Uses Windows Registry to store COM activation data.</td>
<td>Uses manifest files to store COM activation data.</td>
</tr>
<tr>
<td></td>
<td>Multiple versions of a component or</td>
</tr>
</tbody>
</table>
Only a single version of a shared component or application can exist in the Windows system folder (i.e., the \%systemroot\%\system32 folder). An application can be shared and in use at the same time. Each version installs in its own distinctly named subfolder under the Windows side-by-side store (i.e., the \%systemroot\%\WinSxs folder).

The following topics provide more information about working with manifests and MSXML 5.0 in Windows XP and Windows .NET Server.

- **Creating a Manifest for Your Application**
- **Creating a Manifest for Your Component DLL**
- **Troubleshooting Side-by-Side Installations**
- **Managing MSXML Version Dependencies**

**Notes**

- Side-by-Side technology described in this section applies only when you deploy applications that use MSXML 5.0 or later to computers running Microsoft® Windows® XP and Windows® .NET Server. If your deployment involves earlier versions of Microsoft® Windows, such as Windows 2000, Windows NT, Windows Me, or Windows 98, disregard the information in this section.
- Windows XP Side-by-Side installation should not be confused with the side-by-side installation provided in previous versions of MSXML. The previous side-by-side installation was implemented by modifying GUIDs and ProgIDs. This topic covers Windows XP Side-by-Side installation exclusively.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Creating a Manifest for Your Application

To deploy applications that use MSXML 5.0 to computers running Windows XP, you must create an application manifest and deploy it with your application.

The application manifest is an XML file that:

- Uniquely identifies the application
- Describes dependencies on specific versions of side-by-side assemblies

For example, if your application depends upon the RTM version of MSXML 5.0, write your application manifest so it uses the RTM version of MSXML 5.0 installed in the Windows global store. The following procedure describes how to create a manifest for this purpose.

To create a manifest for your application

1. Copy this sample manifest and paste it into any text editor.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="X86"
    name="Microsoft.Windows.mysampleApp"
    type="win32"
/>
<description>
    Your app description here
</description>
<dependency>
    <dependentAssembly>
        <assemblyIdentity
            type="win32"
            name="Microsoft.MSXML2"
            version="5.0.0.0"
            processorArchitecture="X86"
            publicKeyToken="6bd6b9abf345378f"
            language="*"
        />
    </dependentAssembly>
</dependency>
</assembly>
```
2. After you have pasted the sample code above, you must update it as appropriate for your application. To do this, change the featured values (those in bold in the sample code) of the first <assemblyIdentity> element as follows:
   1. For the version attribute, set the value to the current version number for your component or application. This number is specified in the properties for your project when it is built using Microsoft® Visual Studio.
   2. For the processorArchitecture attribute, only change the value of the processor architecture if you are deploying to a processor architecture that is not Intel-x86 compatible. Otherwise, use the suggested default.
   3. For the name attribute, set the value to the registered name for your application. For most Windows applications, you only need to replace "mysampleapp" with the base name of the executable (.exe) file containing your application.
   4. For the type attribute, only change the value if you are deploying to a version of Windows that does not run under the Win32 API. Otherwise, use the suggested default.

3. Update the description for your application.

   The <description> element is optional, so you can either update its contents to include a brief description of your application or remove it from your manifest code. If you choose to remove it, be sure to remove both its opening and closing tags and any character data contained within them.

4. For the dependent assembly section, update the featured attribute values for the <assemblyIdentity> element so that its values are appropriate to your version of MSXML.

   For the version and tokenKey attributes, verify that the values match those used in the assembly manifest that is provided for MSXML 5.0 when it installs under Windows XP. Assembly manifests are located in the special WinSxS/Manifest directory. The assembly manifest file for MSXML 5.0 is called **msxml5.manifest**.

5. Save the application manifest to file in the same directory as the executable file for your application.
Name your manifest file so that it has the same name as your application, and add the "manifest" extension to the end of it. For example, if your application executable is called `test.exe`, the manifest file should be named `test.exe.manifest`.

6. Decide how to package and deploy your application manifest with your application.

As an alternative for new applications, you can include the application manifest directly in the header file for your application. In this case, also add the following line to the header file:

```plaintext
CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST "YourApp.exe.manifest"
```

Be sure you update "YourApp.exe.manifest" with the name for your manifest file. After you add the line above to your application header file, the manifest will be directly included with your application binary file the next time it is compiled.

If you do not choose to include the manifest in this way, include the manifest file separately in the Windows Installer package (.msi) you create to install your application.

**Note** If you provide your manifest both separately and internally in the resource section of your compiled executable, the file system version of your manifest always takes precedence over the one included with the executable file.

7. Test your application to ensure that you get the results you want.

Testing your application before deployment is always important, but you should be sure that your test computer does not already have the side-by-side assembly for MSXML 5.0 registered on it. Otherwise, your testing will be compromised and the results will be incorrect.

8. Deploy or update your application as a Windows Installer package.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Creating a Manifest for Your Component DLL

The following procedure shows how to configure an ActiveX/COM DLL that uses MSXML 5.0 and which is in turn referenced by a Visual Basic EXE application. Here is a diagram of how this application design looks:

Project1.exe (a Visual Basic Standard EXE)  
calls DllTest.dll (ActiveX/COM DLL)  
which depends upon and uses MSXML 5.0 assembly

In this scenario, to achieve side-by-side installation under Windows XP you need to:

- Create Project1.exe, a simple client application for testing Dlltest.dll
- Author an assembly manifest for DllTest.dll.

To create a manifest for your component DLL

1. Create Dlltest.dll and Project1.exe to use with procedure.

Note   If you want, you can create these files yourself from scratch, using either Visual Basic or Visual C++. For more information about how to create these files using Visual Basic, see Using Visual Basic to Create Sample DLL and EXE files.

1. Before you begin, you need to know the following information about DllTest.dll:
   1. Description
   2. CLSID
   3. ProgID
   4. Threading model
   5. Type library ID

If you are using Visual C++, a good place to look is the .rgs file, which looks like this:
HKCR
{
    DllTest.FusionTest.1 = s 'FusionTest Class'
    {
        CLSID = s '{CDC6C7A2-C16B-4D84-8E46-EBEDBCC50BF}'
    }
    DllTest.FusionTest = s 'FusionTest Class'
    {
        CLSID = s '{CDC6C7A2-C16B-4D84-8E46-EBEDBCC50BF}'
        CurVer = s 'DllTest.FusionTest.1'
    }
    NoRemove CLSID
    {
        ForceRemove {CDC6C7A2-C16B-4D84-8E46-EBEDBCC50BF} = s 'FusionTest Class'
        {
            ProgID = s 'DllTest.FusionTest.1'
            VersionIndependentProgID = s 'DllTest.FusionTest'
            ForceRemove 'Programmable'
            InprocServer32 = s '%MODULE%'
            {
                val ThreadingModel = s 'Apartment'
            }
        }
        'TypeLib' = s '{4D2F7845-6A75-40EF-AD97-179C5088E507}'
    }
}

If you are using Visual Basic, you might want to use the OLE Viewer tool to browse these properties for the object and its type library.

2. Open Notepad, and copy and paste following code. Save it as DllTest.dll.manifest in the same folder as the Project1.exe and DllTest.dll files.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
<assemblyIdentity
    version="1.0.0.1"
    processorArchitecture="x86"
    name="DllTest.dll"
    type="win32" />
<file name="DllTest.dll">
<typelib
    tlbid="{4D2F7845-6A75-40EF-AD97-179C5088E507}"
    version="1.0"
    helpdir=""/>
<dependentAssembly>
    <assemblyIdentity
```
3. After you have pasted the sample code above, you must update it as appropriate for the type library and object used by your application. To do this, change the featured values (those in bold in the sample code) as follows for the DLL file used by your application:
   1. For the name attribute of the first <assemblyIdentity> element, set the value to "DllTest.dll".
   2. For the name attribute of the <file> element, set the value to "DllTest.dll".
   3. For the tlbid attribute of the <typelib> element, set the value to the type library ID for the DLL.
   4. For the description attribute of the <comClass> element, set the value to the description for the DLL.
   5. For the clsid attribute of the <comClass> element, set the value to the CLSID for the DLL.
   6. For the threadingModel attribute of the <comClass> element, set the value to the threading model implemented for the DLL.
   7. For the tlbid attribute of the <comClass> element, set the value to the type library ID for the DLL.

   **Note** If you have additional DLLs packaged in this assembly, you can add additional instances of the <file> element, for example:

   ```xml
   <file name="Another.dll">
     ...
   </file>
   ```

   You will then need to repeat this step (and each of its substeps) for each <file> element you add.
4. Run and test the application.
   1. Double-click Project1.exe.

      You should see a form with a Test DLL button centered on it.

2. Click Test DLL.

      You should see a message box that displays "<test/>". If you do, the application has been successfully tested.

      If the application does not run correctly in this way, you will need to do further troubleshooting to determine the problem. For example, you might see this error:

      "This application has failed to start because the application configuration is incorrect. Reinstalling the application may fix this problem."

      If this occurs, check the manifest file for errors and validate it. You do not need to reinstall your application to perform this checking. For more information about tools for checking manifests, see the following section.

**Note**  Modifying your component so that it becomes public (i.e., shared globally to other applications on the system) requires a number of additional steps and considerations. For more information, download and review the whitepaper referenced at the end of this article, [How to Build and Service Isolated Applications and Side-by-Side Assemblies for Windows XP](#).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Using Visual Basic to Create Sample DLL and EXE files

In [Creating a Manifest for Your Component DLL](#), you are given instructions on how to author a manifest and test side-by-side installation of your component DLL that uses MSXML 5.0. The process requires you to have or make a simple EXE that uses another small DLL, which in turn, depends upon MSXML 5.0. Since you need these files to test side-by-side installation under Windows XP, the procedures in this topic provide you instructions for how to generate those files.

To create DllTest.dll using Visual Basic

1. From Visual Basic, create a **New Project** (select **ActiveX DLL** as the project type).
2. Rename the project as "DllTest" and the class module as "TestSxS".
3. From the **Project** menu, select **References**.
4. From the list of available references, select **Microsoft XML, v5.0**.
5. Paste the following code in the TestSxS class module:

```vbscript
Option Explicit
Private xmlDoc As MSXML2.DOMDocument50
Private mvarIt As String

Public Property Let It(ByVal vData As String)
    mvarIt = vData
End Property

Public Property Get It() As String
    Set xmlDoc = New DOMDocument50
    xmlDoc.async = False
    xmlDoc.loadXML "<" & mvarIt & ">
    It = xmlDoc.xml
End Property

Private Sub Class_Initialize()
    mvarIt = "test"
End Sub

Private Sub Class_Terminate()
```

[72x674]
Set xmlDoc = Nothing
End Sub

6. From the File menu, select Make DllTest.dll.

Select a folder on your computer (such as c:\temp) where you want the compiled DllTest.dll file to be placed, and then click OK.

7. If you will be using and testing DllTest.dll on this computer, you need to unregister it. To unregister DllTest.dll, do the following:
   1. Open a command prompt window and change the directory to the folder that you used in Step 6.
   2. Type "regsvr32 /u DllTest.dll", and then press Enter.

   Note  Visual Basic automatically creates the type library for a project, and registers the DLL globally on your development computer when you create it. If you copy DllTest.dll directly from your computer to another computer to test it with your application, you can skip this step.

To create MyTestApp.exe using Visual Basic

1. From Visual Basic, create a New Project. Select Standard EXE as the project type.
2. Rename the project as "MyTestApp".
3. From the Project menu, select References.
4. Click Browse. Browse to the location where you created DllTest.dll in the previous procedure, and select it.

   You should see DllTest selected in the list of available references. After you do, click OK.

5. Create a command button (Command1) on Form1 and change its Caption property from "Command1" to "Test DLL".
6. Double click on the Command1 button and enter code in the Form1 module to match the following:

   Private Sub Command1_Click()
   Dim MyObject As New DllTest.TestSxS
   MyObject.It = "test"
   MsgBox MyObject.It
   End Sub
7. From the **File** menu, select **Make MyTestApp.exe**.

   Select the same folder on your computer where you created DllTest.dll (such as c:\temp). Click **OK**.

**See Also**

[Creating a Manifest for Your Component DLL](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Troubleshooting Side-by-Side Installations

If you are having problems testing or developing your side-by-side installed applications under Windows XP, the following tips and best practices might help.

- Verify the version and path of any dependent DLLs loaded by your application. A variety of tools are available that can provide this information. For information about using ListDLLs (a non-Microsoft freeware utility) to do this, see Obtaining Information about Dependent DLLs.
- Manifest files are written in XML, and XML markup is case-sensitive. Verify that you have preserved exact case in any characters used for element or attribute names.
- In general, if you are using Notepad or another text editing program, using a tool to validate your manifests before you attempt to use them should help catch authoring errors and make sure that your manifest files conform to the manifest file schema. One such tool for validating manifests is Manifestchk.vbs, which is available from MSDN.
- Make sure that your manifest files are named identically to the names of your application or assemblies. For example, the manifest file for MyApplication.exe should be named "MyApplication.exe.manifest" and the name for the manifest file for MyComponent.dll should be "MyComponent.dll.manifest", and so on.
- When preparing your side-by-side installations for deployment, do not use .cab files to package your application for installing on Windows XP. If you do need to use a .cab file, be aware that side-by-side installation is no longer available.
- Public key tokens and signatures are used to verify the trustworthiness of your DLL. If you are concerned about this issue, consider using additional tools like Mt.exe or Pktextract.exe to help prepare your side-by-side installation.
- Preferably, use one of the current versions of a third-party installation tool such as InstallShield that supports the Windows Installer 2.0. These tools
simplify the process of packaging and installing side-by-side assemblies by automating the design of various aspects of your Windows installer package (.msi) file.
Obtaining Information about Dependent DLLs

When testing side-by-side installation of your component, you might want to determine additional information about the version and location of MSXML 5.0 loading with it. This task becomes more important if more than one version of MSXML 5.0 has been side-by-side installed under Windows XP and you are trying to troubleshoot problems where potentially the version you need is not the one loading with your component or application.

This section provides instructions for using a third-party utility, ListDLLs, to check the version of Msxml4.dll loaded. ListDLLs also works well for checking version information for other dependent DLLs you might use in your applications.

To use ListDLLs to obtain information about dependent DLLs

1. Download the ListDLLs utility from the SysInternals.com Web site.

   Note  ListDLLs is currently a freeware utility made available by a third-party software vendor, SysInternals.com. As such, it is not a Microsoft product and is not supported by Microsoft. For more information about ListDLLs, contact SysInternals.com directly.

2. Extract and install your copy of Listdlls.exe to the folder for testing where you previously created your other sample files (Dlltest.dll and Project1.exe).
3. Run Project1.exe from the same test folder location.
4. Open a command prompt window and change directories to the test folder location. Then enter the following command:

   C:\temp>listdlls project1.exe

At this point, you should see output similar to the following. This indicates that Msxml4.dll has not been loaded by the sample application.

ListDLLs V2.23 - DLL lister for Win9x/NT
Copyright (C) 1997-2000 Mark Russinovich
http://www.sysinternals.com
---------------------------------------------------------------------
Project1.exe pid: 2808
Command line: "C:\data\SxS-article\Project1.exe"

<table>
<thead>
<tr>
<th>Base</th>
<th>Size</th>
<th>Version</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00400000</td>
<td>0x4000</td>
<td>1.00.0000.0000</td>
<td>C:\temp\Project1.exe</td>
</tr>
<tr>
<td>0x77f50000</td>
<td>0xa9000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\System32\ntdll</td>
</tr>
<tr>
<td>0x77e60000</td>
<td>0xe5000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\kern32</td>
</tr>
<tr>
<td>0x73420000</td>
<td>0x153000</td>
<td>6.00.0092.0037</td>
<td>C:\WINDOWS\System32\MSVE</td>
</tr>
<tr>
<td>0x77d40000</td>
<td>0x8d000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\USER32</td>
</tr>
<tr>
<td>0x77c70000</td>
<td>0x40000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\GDI3</td>
</tr>
<tr>
<td>0x77dd0000</td>
<td>0x8b000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\KERNEL32</td>
</tr>
<tr>
<td>0x77cc0000</td>
<td>0x75000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\RPCRT4</td>
</tr>
<tr>
<td>0x771b0000</td>
<td>0x11a000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\OLE3</td>
</tr>
<tr>
<td>0x77120000</td>
<td>0x8b000</td>
<td>3.50.5014.0000</td>
<td>C:\WINDOWS\system32\OLEXML</td>
</tr>
<tr>
<td>0x77cc0000</td>
<td>0x75000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\system32\MSVC</td>
</tr>
<tr>
<td>0x74720000</td>
<td>0x4b000</td>
<td>5.01.2600.0000</td>
<td>C:\WINDOWS\System32\MSCT</td>
</tr>
</tbody>
</table>

5. Click **Test DLL** in the sample application.
6. Repeat the command you entered in Step 4.

You should now see lines similar to the following. This indicates that both Dlltest.dll and Msxml4.dll are loaded:

```
0x10000000 0x43000 1.00.0000.0001 C:\temp\DllTest.dll
0x69b10000 0x139000 4.00.9004.0000 C:\WINDOWS\System32\msxml
```

**Other Resources**

- [How to Build and Service Isolated Applications and Side-by-Side Assemblies for Windows XP](#)
- [Isolated Applications and Side-by-Side Assemblies](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Managing MSXML Version Dependencies

Creating an application manifest will work when you first deploy your application with MSXML 5.0. But what happens if you want to control which version of MSXML your application uses as new versions of it are released?

For example, you might want to:

- Have your application use the version of MSXML that you originally tested and deployed with your application when it was released.
- Have your application updated to use a more recent version of MSXML as new versions are released in the future.

Windows XP enables assembly publishers, application publishers, and administrators to change side-by-side assembly dependencies after deployment with publisher configuration and per-application configuration.
Publisher configuration

Publisher configuration redirects applications and assemblies that have a default dependence on one version of a side-by-side assembly, so that they use another specified version instead. Assembly publishers can implement a publisher configuration by authoring and installing publisher configuration files. Publisher configuration files override the default configuration specified by the application manifest. Note that a per-application configuration can override the publisher configuration with application configuration files.

Typically, a publisher configuration would be installed when you install a service pack that contains an assembly update. This enables the applications to use the updated assembly without having to rebuild all the affected applications. For example, an administrator might deploy an assembly update by redirecting all applications and assemblies on the system that use assembly version 1.0.0.0 by default to use version 1.1.0.0 instead. In this case, the new assembly version must be backward compatible.
**Per-application configuration**

Per-application configuration overrides the default configuration and publisher configuration on a per-application basis. This redirects the dependence of a specific application from one version of a side-by-side assembly to another specified version of the assembly.

A per-application configuration might be necessary if a particular application requires an assembly version that is different from the version that is normally specified as a default or publisher configuration.

**Example: Updating publisher and per-application configurations**

The following table shows an example of a hypothetical configuration in chronological order. Consider two applications, A and B, which both use component (assembly) X.

<table>
<thead>
<tr>
<th>Action</th>
<th>Application A uses component X version</th>
<th>Application B uses component X version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state. Component X 1.0 is installed. The application manifest of Application A (Manifest A) points to it. Application B does not have a manifest.</td>
<td>1.0</td>
<td>None. Cannot use X.</td>
</tr>
<tr>
<td>Install X version 1.1.</td>
<td>1.0</td>
<td>Cannot use X.</td>
</tr>
<tr>
<td>Change Manifest A to point to X 1.1.</td>
<td>1.1</td>
<td>Cannot use X.</td>
</tr>
<tr>
<td>Create an application manifest for Application B (Manifest B) to point to X 1.1.</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Install X 1.2.</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Publisher configuration file installed, redirecting</td>
<td>1.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>
everything from X 1.0 to 1.1.

Publisher configuration file installed, redirecting everything from X 1.1 to 1.2.

Manifest A is updated to point to 1.2.

X 1.3 installed.

Publisher configuration installed, redirecting everything from X 1.1 to 1.3.

Manifest B is updated to point to 1.3.

X 1.4 installed.

Publisher configuration installed, redirecting everything from 1.2 to 1.4.

This example demonstrates some of the possible situations you should pay attention to when using publisher and application configuration.

For more information about publisher and application configuration, see the Windows Platform SDK.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
**Tips for Converting Samples to VBScript**

**Note** This topic is intended to provide some guidance for VBScript developers. For more information about JScript or VBScript, see the documentation available from the [MSDN Scripting Developer's Center](https://msdn.microsoft.com).

Most of the sample code in the MSXML SDK is written in JScript, Visual Basic, or C/C++. However, much of the provided JScript code can be modified to run successfully as VBScript.

The following are some general tips for converting JScript examples to functional VBScript.

- **Use the CreateObject function to replace instances of the ActiveXObject object**
- **Remove line termination characters used in JScript sample code**
- **Declare variables using the correct syntax**
- **Remove parentheses as needed when you set properties or call methods that require additional parameters**
- **Substitute the MsgBox function for alert() method calls**
Use the CreateObject function to replace instances of the ActiveXObject object

When working with MSXML in either JScript or VBScript, you need to create an automation object (for example, a DOMDocument object), then assign its contents to a variable.

To perform this step in JScript, you create a new instance of the ActiveXObject object and assign its contents to a variable, like this:

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
```

To perform this same step in VBScript, you need to use the CreateObject function, and assign the value it returns to the named variable, like this:

```vbnet
set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
```

Note that because the value returned is of the object data type, in VBScript you need to use the set statement to assign it to the variable (xmlDoc).
Remove line termination characters used in JScript sample code

In JScript, a line of code can be spread over multiple lines of text in the source file. Therefore, semi-colons are used as line termination characters to indicate the end of each line of code.

These semi-colons are not necessary in VBScript. By default, VBScript treats each line of text as a single line of code, unless you use the Visual Basic line continuation character (an underscore) to indicate that the current line wraps to the next line of text.
Declare variables using the correct syntax

In JScript, you declare a variable using the `var` statement, like this:

```
var node;
```

If you are declaring variables in VBScript, use the `Dim` statement, like this:

```
Dim node
```
Remove parentheses as needed when you set properties or call methods that require additional parameters

In JScript, you need to enclose parameters in parentheses when you set properties or call a method on an object. For example:

```javascript
xmlDoc.load("books.xml");
xmlDoc.setProperty("SelectionLanguage", "XPath");
currNode = xmlDoc.selectSingleNode("//book/author");
```

In VBScript, you need to remove the parentheses from the first two lines, like this:

```vbnet
xmlDoc.load "books.xml"
xmlDoc.setProperty "SelectionLanguage", "XPath"
```

However, you do use parentheses in VBScript when the method is used as part of the right side of an equal assignment statement. So the third line is the same in both JScript and VBScript:

```vbnet
currNode = xmlDoc.selectSingleNode("//book/author")
```
Substitute the MessageBox function for alert() method calls

When you write JScript to display XML or other text string results in message boxes within Internet Explorer, you use the alert() method of the window object. To do this in VBScript, use the MsgBox function.

Example

The following are two different HTML files that display the contents of a simple XML DOM document within an HTML page. The first file uses JScript; the second uses VBScript. The differences are highlighted in bold text.

PopupUsingJScript.htm

```html
<HTML>
<HEAD>
<TITLE>Displaying a Simple DOM Document in a Browser Popup</TITLE>
<SCRIPT LANGUAGE="JScript">
function displayXml() {
    var xmldoc = new ActiveXObject("MSXML2.DOMDocument.5.0");
xmldoc.loadXML("<root><child></child></root>");
alert(xmldoc.xml);
}
</SCRIPT>
</HEAD>
<BODY>
<BUTTON onClick="displayXml();">Display XML</BUTTON>
</BODY>
</HTML>
```

PopupUsingVBScript.htm

```html
<HTML>
<HEAD>
<TITLE>Displaying a Simple DOM Document in a Browser Popup</TITLE>
<SCRIPT LANGUAGE="VBScript">
Sub DisplayXml
    set xmldoc = CreateObject("MSXML2.DOMDocument.5.0")
xmldoc.loadXML "<root><child></child></root>"
    MsgBox xmldoc.xml
End Sub
</SCRIPT>
```
Try It!

1. Open Notepad.
3. From the File menu, click Save As. Save the file as `PopupUsingJScript.htm` to a folder on your computer.
5. From the File menu, click Save As. Save the file as `PopupUsingVBScript.htm` to the same folder in which you saved `PopupUsingJScript.htm`.
6. Open each file in Internet Explorer, in separate browser windows. When each file appears in its own browser window, click Display XML to see output for each.

Output

The following contents of the XML DOM document will appear as a browser popup:

```
<root><child></child></root>
```

The results of Step 6 are the same, regardless of whether you are executing the JScript or VBScript version of the scripted HTML page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Using Language Filtering

The Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office Software Development Kit allows you to display only the information that is relevant to your programming environment, for example, Microsoft® Visual Basic®, C/C++, or script. To use this language filtering functionality, click the Language Filter button in the upper-left corner of the title bar. From the Language menu, select the language. Information that pertains to all languages will be displayed regardless of the language option you select.

Note  After a language filtering option is selected, that selection is persisted until you change the option.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Sample XML File (books.xml)

The following XML file is used in various samples throughout the Microsoft
XML Core Services (MSXML) 5.0 for Microsoft Office SDK.

<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.</description>
  </book>
  <book id="bk104">
    <author>Corets, Eva</author>
    <title>Oberon's Legacy</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-03-10</publish_date>
    <description>In post-apocalypse England, the mysterious agent known only as Oberon helps to create a new life for the inhabitants of London. Sequel to Maeve
Ascendant.

<book id="bk105">
<author>Corets, Eva</author>
<title>The Sundered Grail</title>
<genre>Fantasy</genre>
<price>5.95</price>
<publish_date>2001-09-10</publish_date>
<description>The two daughters of Maeve, half-sisters, battle one another for control of England. Sequel to Oberon's Legacy.</description>
</book>

<book id="bk106">
<author>Randall, Cynthia</author>
<title>Lover Birds</title>
<genre>Romance</genre>
<price>4.95</price>
<publish_date>2000-09-02</publish_date>
<description>When Carla meets Paul at an ornithology conference, tempers fly as feathers get ruffled.</description>
</book>

<book id="bk107">
<author>Thurman, Paula</author>
<title>Splish Splash</title>
<genre>Romance</genre>
<price>4.95</price>
<publish_date>2000-11-02</publish_date>
<description>A deep sea diver finds true love twenty thousand leagues beneath the sea.</description>
</book>

<book id="bk108">
<author>Knorr, Stefan</author>
<title>Creepy Crawlies</title>
<genre>Horror</genre>
<price>4.95</price>
<publish_date>2000-12-06</publish_date>
<description>An anthology of horror stories about roaches, centipedes, scorpions and other insects.</description>
</book>

<book id="bk109">
<author>Kress, Peter</author>
<title>Paradox Lost</title>
<genre>Science Fiction</genre>
<price>6.95</price>
<publish_date>2000-11-02</publish_date>
<description>After an inadvertent trip through a Heisenberg Uncertainty Device, James Salway discovers the problems of being quantum.</description>
</book>

<book id="bk110"/>
<book id="bk111">
  <author>O'Brien, Tim</author>
  <title>MSXML3: A Comprehensive Guide</title>
  <genre>Computer</genre>
  <price>36.95</price>
  <publish_date>2000-12-01</publish_date>
  <description>The Microsoft MSXML3 parser is covered in detail, with attention to XML DOM interfaces, XSLT processing, SAX and more.</description>
</book>

<book id="bk112">
  <author>Galos, Mike</author>
  <title>Visual Studio 7: A Comprehensive Guide</title>
  <genre>Computer</genre>
  <price>49.95</price>
  <publish_date>2001-04-16</publish_date>
  <description>Microsoft Visual Studio 7 is explored in depth, looking at how Visual Basic, Visual C++, C#, and ASP+ are integrated into a comprehensive development environment.</description>
</book>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office
Copyright and Legal Information

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2002 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BizTalk, FrontPage, JScript, MS-DOS, MSDN, Visual Basic, Visual C++, Visual InterDev, Visual Studio, Win32, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
The following topics provide information about Extensible Markup Language (XML).

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Starter Kit</strong></td>
<td>This section provides an introduction to XML, explains some of the benefits of XML, and demonstrates how to create a simple XML document.</td>
</tr>
<tr>
<td><strong>Concepts</strong></td>
<td>This section provides in-depth information about XML.</td>
</tr>
<tr>
<td><strong>How Do I?</strong></td>
<td>This section provides a series of tutorials that demonstrate common XML tasks.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel-
On the surface, XML looks like HTML. Both are derived from the Standard Generalized Markup Language (SGML). Tools that generate HTML can often be reused to generate XML.

XML is different from HTML in two key areas: syntax and semantics.
XML Syntax for Well-formed Documents

Both HTML and XML use <, >, and & to create element and attribute structures. While HTML browsers accept or ignore mangled markup language, XML parsers and applications built on those parsers are less forgiving. Errors in XML syntax halt document processing, and users or applications receive error messages, not a best-guess interpretation of the document structure.

XML documents must be well-formed. That is, they must follow rules for identifying document parts and creating nested element structures. These rules include:

- **An XML document can only have one document element.** The document element is a single element that contains all the content considered to be part of the document itself. This document root element is the first element to appear after the document prolog. For more information, see Elements.
- **All XML elements must have end tags.** While end tags may be optional with certain HTML elements, all elements in XML must have an end tag. For more information, see Elements.
- **XML elements cannot overlap.** If the start tag for an element appears within another element, it must end within the same containing element. For example, the following HTML code suggests a combination of bold and italic by overlapping the structures.

  `<b>This is bold text. <i>This is bold italic text.</i></b> This is italic text.`

  In some HTML browsers, this text appears as follows.

  **This is bold text. This is bold italic text. This is italic text.**

  In an XML parser, however, all processing halts as soon as `</b>` is encountered because the XML parser is looking for `</i>`, and will not accept `</b>`. To achieve the same formatting in XML, use the following syntax.

  `<b>This is bold text.</b> <i><b>This is bold italic text.</b> This is italic text.</i>`

  This extra work for XML document creators results in a leap forward for interoperability. Because XML processors have far less "guessing" code,
they fit more easily into smaller-scale processing, like embedded systems. Structural ambiguities are eliminated from XML documents—all XML parsers see the same nested element structures.

- **All attribute values must be quoted, whether or not they contain spaces.** You still have the option of single- or double-quotes. For more information, see [Attributes](#).

- **You cannot use the characters <, >, or & within the text of your documents.** Use the built-in entities &lt;, &gt;, and &amp;. For more information, see [Character and Entity References](#).
XML Semantics

Although XML is unforgiving about syntax, it offers developers more options for defining meaning in XML documents. HTML is basically one vocabulary with a few variations; `<b>` always means the same thing to an HTML processor. With XML, you can create your own markup vocabulary or choose from markup vocabularies appropriate to your industry or project type. Schemas and document type definitions (DTDs) let you describe these vocabularies, but you can also create documents using vocabularies without formal definitions. Namespaces help you identify the vocabulary you are using.

This approach requires architectures different from those used by browsers. Developers cannot count on XML applications to understand what their markup means or how it is to be presented, understandings that were built into HTML browsers. Browsers can still present XML, but require a style sheet to format to your specifications. These style sheets are built using cascading style sheets (CSS) or XSL Transformations (XSLT). Some browsers, including Internet Explorer 5.0 and later, include a default style sheet, but it is designed more for diagnostics than for presenting information to end users.

XML applications can also bring their own logic to XML vocabularies, rather than relying on style sheets. This logic may take the form of simple scripts or binding to particular presentation modes, or it may involve writing an entire application from scratch. These applications can take advantage of their built-in knowledge of the labeled structures contained in XML documents to process the information in those documents, present them to users, connect them with other data sources, or redirect them to other appropriate consumers.

In This Section

- [Advantages of the XML Format](#)
- [Creating a Simple XML Document](#)
- [Displaying the File in Internet Explorer](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Advantages of the XML Format

In some ways, XML is just another data format. However, XML has several advantages over other formats when storing information.

- XML allows developers to create their own labeled structures for storing information.
- XML parsing is well-defined and widely implemented, making it possible to retrieve information from XML documents in a variety of environments.
- XML is built on a Unicode foundation, making it easier to create internationalized documents.
- Applications can rely on XML parsers to do some structural validation, as well as data type checking (when schemas are used).
- XML formats are text-based, making them more readable, easier to document, and sometimes easier to debug.
- Tools are available for XML processing on different platforms, making it simpler to use XML instead of binary formats to exchange complex information streams.
- XML documents can use much of the infrastructure already built for HTML, including the HTTP protocol and some browsers.

XML isn't appropriate for every situation, however. XML documents tend to be more verbose than the binary formats they replace. They take up more network bandwidth and storage space, or require more processor time for compression. XML parsing can be slower than parsing highly optimized binary formats and can require more memory. However, careful application design can avoid some of these problems.
Creating a Simple XML Document

To demonstrate how XML structures can be used to break down and describe information, we will create a very basic XML document containing an employee record that includes the following fields.

- Name
- Home Address
- Job Title
- Salary

This is the sample employee record.

Jim Kim
1234 South Street
Anywhere, NY 10001
USA
Vice President of Finance
$175,000
Defining Basic Elements

To start our conversion to XML, we need a document or root element, in this case, `<employeeRecord>`, to hold all of our document content. For more information about the document or root element, see Elements.

Next, we will put the employee name into a `<name>` element; the address into a `<homeAddress>` element; the title into a `<jobTitle>` element; and the salary into a `<salary>` element.

```xml
<employeeRecord>
  <name>Jim Kim</name>
  <homeAddress>1234 South Street
  Anywhere, NY 10001
  USA</homeAddress>
  <jobTitle>Vice President of Finance</jobTitle>
  <salary>$175,000</salary>
</employeeRecord>
```

The beginnings of elements are marked with start tags, like `<employeeRecord>`, while the ends of elements are marked with end tags, like `</employeeRecord>`.

All of the elements nest properly. No element contains another element's start tag without including its end tag as well.
Refining Elements

The <name> and <homeAddress> elements contain information that might be useful for sorting and searching, for example, sorting the list by family name or finding all employees who live in a certain country/region or postal code.

We will add some child elements, <givenName>, <middleName>, and <familyName>, to the name element.

<name><givenName>Jim</givenName> <middleName></middleName> <lastName>Kim</lastName></name>

We will also add more detail to our home address element.

<homeAddress><street>1234 South Street</street><city>Anywhere</city>, <stateProvince>NY</stateProvince> <postalCode>10001</postalCode><country>USA</country></homeAddress>

The document now looks as follows:

<employeeRecord>
<name><givenName>Jim</givenName> <middleName></middleName> <lastName>Kim</lastName></name>
<homeAddress><street>1234 South Street</street><city>Anywhere</city>, <stateProvince>NY</stateProvince> <postalCode>10001</postalCode><country>USA</country></homeAddress>
<jobTitle>Vice President of Finance</jobTitle>
<salary>$175,000</salary>
</employeeRecord>

In this version, element structures identify all of the document content. If we had content that might not belong in the document or that might not ever need to contain further structure, we might use an attribute. For example, if this company were paying salaries in more than one currency, it might make sense to indicate that the salary presented here is in U.S., not Canadian, dollars. A currency attribute on the salary element lets us indicate that as follows.

<salary currency="USD">$175,000</salary>

In a similar way, an employeeID attribute on the <employeeRecord> element might make it easier to manipulate employee record documents, especially if we had to combine multiple records into a single XML document.
The complete document now looks as follows.

<employeeRecord employeeID="2344-12Z">

<name><givenName>Jim</givenName> <middleName></middleName> <lastName>Kim</lastName></name>

<homeAddress><street>1234 South Street</street> <city>Anywhere</city>, <stateProvince>NY</stateProvince> <postalCode>10001</postalCode> <country>USA</country></homeAddress>

<jobTitle>Vice President of Finance</jobTitle>

<salary currency="USD">$175,000</salary>

</employeeRecord>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Displaying the File in Internet Explorer

Microsoft® Internet Explorer provides a default view of XML documents that displays their structure, in addition to support for styled display of XML information using cascading style sheets (CSS) or Extensible Stylesheet Language Transformations (XSLT). If an XML document contains an `xml-stylesheet` processing instruction, Internet Explorer uses the specified style sheet to present the information. If no `xml-stylesheet` processing instruction is present, Internet Explorer applies its default style sheet.

The default style sheet is useful for a number of common development cases. Loading a document into Internet Explorer causes Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office to parse the document and check whether it is a well-formed. If the parse fails, Internet Explorer reports error messages. If the parse succeeds, Internet Explorer presents the document structure as an outline, with small + and – icons for expanding and closing sections of the document. If you have long documents with many large sections, this view can help you quickly navigate the information.

Like other Web content, you can load XML documents into Internet Explorer through the **Open** command on the **File** menu, a hyperlinked URL, or from the file system.

**See Also**

[Displaying XML Files in a Browser](#)
Concepts

The following topics provide information about XML document components.

- Document Map
- Using XML with HTML
- Using Namespaces in Documents
- Using XML on the Server
- XML and Security
- XML Data Islands
- Binding the XML Data Source Object to Data
The following is an XML document. Click on the colored text for more information about each structural construct.
Elements

Elements form the backbone of XML documents, creating structures you can manipulate with programs or style sheets. Elements identify named sections of information and are built using markup tags that identify the name, start, and end of the element.

Elements can also contain attribute names and values, which provide additional information about your content. For more information, see Attributes.
Element Names

All elements must have names. Element names are case-sensitive and must start with a letter or underscore. An element name can contain letters, digits, hyphens, underscores, and periods.

Note  Colons are reserved for use with namespaces. For more information about which Unicode characters are acceptable letters and digits, see Appendix B of the XML specification.
Start Tags, End Tags, and Empty Tags

Tags establish boundaries around the content, if any, of the element.

Start tags indicate the beginning of an element and use the following general syntax.

\[<\text{elementName} \text{att1Name}="\text{att1Value}" \text{att2Name}="\text{att2Value}"...>\]

For elements that do not have attributes, the start tag can be reduced.

\[<\text{elementName}>\]

End tags indicate the end of an element and cannot contain attributes. End tags always take the following form.

\[</\text{elementName}>\]

An element is generally considered to include the start and end tags, and everything in between.

\[<\text{person}><\text{givenName}>\text{Peter}</\text{givenName}> <\text{familyName}>\text{Kress}</\text{familyName}>\]

In this case, the \(<\text{person}\>\) element contains two other elements, \(<\text{givenName}\>\) and \(<\text{familyName}\>\), along with a space that separates them. The \(<\text{givenName}\>\) element contains the text Peter while the \(<\text{familyName}\>\) element contains the text Kress.

Empty tags are used to indicate elements that have no textual content, though they can have attributes. The HTML \(<\text{img}\>\) and \(<\text{br}\>\) elements are examples of empty elements. Empty tags can be used as a shortcut when there is no content between the start and end tags of a document. Empty tags look like start tags, except that they contain a slash (/) before the closing >.

\[<\text{elementName} \text{att1Name}="\text{att1Value}" \text{att2Name}="\text{att2Value}".../>\]

In XML, you can indicate an empty element with start and end tags and no white space or content in between, for example, \(<\text{giggle}()<<\text{giggle}>\), or you can use an empty tag, for example, \(<\text{giggle}/>\). The two forms produce identical results in an XML parser.
Element Relationships

Relationships between elements are described using either family or tree metaphors. XML documents must contain a document element (also known as a root element). Although it can be preceded and followed by other markup, such as declarations, processing instructions, comments, and white space, the root must contain all of the content considered to be part of the document itself. For example, the following code can be an XML document with <person> as its root element.

```
<person><givenName>Stephanie</givenName> <familyName>Bourne</familyName></person>
```

The following fragment cannot be an XML document because it has multiple root elements.

```
<givenName>Stephanie</givenName>
<familyName>Bourne</familyName>
```

**Note**  Document fragments can be useful as parts of an XML document but should not be passed to the parser on their own. The parser will report an error when it encounters the second element or text outside of an element.

In the tree metaphor, the leaves refer to elements that do not contain any other elements, like leaves on the end of a branch. Leaf elements are generally elements containing only text or nothing at all; leaf nodes are generally empty elements or text. In the document map, all of the text describing the books is stored in leaf elements; the text itself is the leaf node.

Family metaphors, such as parent, child, ancestor, descendant, and sibling, are used to describe relationships between elements relative to each other, not necessarily to the entire document. The following abstract sample document illustrates the relationships between elements.

```
<a>
  <b>
    <c>
      <d/><e/><f/>
    </c>
  </b>
</a>
```
The <a> element contains the <b> element, which contains the <c> element, which contains the <d>, <e>, and <f> elements. Using the tree metaphor, <a> is the root element, and <d>, <e>, and <f> are leaf elements. Although <b> and <c> might be considered trunks or branches, these descriptions are rarely used.

The family metaphors provide more levels of description. The only siblings in this document are the <d>, <e>, and <f> elements, all of which are contained by the <c> element. The <c> element is the parent of the <d>, <e>, and <f> elements; the <d>, <e>, and <f> elements are the child elements of the <c> element. In the same way, the <b> element is the parent of the <c> element and the <c> element is the child of the <b> element, while the <a> element is the parent of the <b> element and the <b> element is the child of the <a> element.

Ancestors and descendants are defined in a way similar to parents and children, except that they do not have to contain or be contained directly. The <a> element is the parent of the <b> element, and the ancestor of every element in the document. The <d>, <e>, and <f> elements are descendants of the <a>, <b>, and <c> elements.

See Also

Document Map

Other Resources 📚

Appendix B, Character Classes
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel...
Prolog

The prolog refers to the information that appears before the start tag of the document or root element. It includes information that applies to the document as a whole, such as character encoding, document structure, and style sheets.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<!--catalog last updated 2000-11-01--> 
```

Processing instructions can also appear in the prolog, for example, the `xml-stylesheet` processing instruction, `<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>`. For more information, see Processing Instructions.

Comments can appear in the prolog, for example, `<! -- catalog last updated 2000-11-01--!>`. For more information, see Comments.
XML Declaration

The XML declaration typically appears as the first line in an XML document. The XML declaration is not required, however, if used it must be the first line in the document and no other content or white space can precede it.

The XML declaration in the document map consists of the following:

- The version number, `<?xml version="1.0"?>`.
  
  This is mandatory. Although the number will change for future versions of XML, 1.0 is the current version.

- The encoding declaration, `<?xml version="1.0" encoding="UTF-8"?>`.
  
  This is optional. If used, the encoding declaration must appear immediately after the version information in the XML declaration, and must contain a value representing an existing character encoding.

An XML declaration can also contain a standalone declaration, for example, `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`. Like the encoding declaration, the standalone declaration is optional. If used, the standalone declaration must appear last in the XML declaration.

Encoding Declaration

The encoding declaration identifies which encoding is used to represent the characters in the document. Although XML parsers can determine automatically if a document uses the UTF-8 or UTF-16 Unicode encoding, this declaration should be used in documents that support other encodings.

For example, the following is the encoding declaration for a document that uses the ISO-8859-1 (Latin 1).

`<?xml version="1.0" encoding="ISO-8859-1"?>`

**Note**  Case in the value specified is not considered by the encoding
declaration. "ISO-8859-1" is the equivalent of "iso-8859-1".

The following is the encoding declaration for a document that uses the Japanese encoding method Shift-JIS.

```xml
<?xml version="1.0" encoding="Shift-JIS"?>
```

For more information about how Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office handles character encodings, see Character Encoding, XML, and MSXML.

### Standalone Declaration

The standalone declaration indicates whether a document relies on information from an external source, such as external document type definition (DTD), for its content.

If the standalone declaration has a value of "yes", for example, `<?xml version="1.0" standalone="yes"?>`, the parser will report an error if the document references an external DTD or external entities.

Leaving out the standalone declaration produces the same result as including a standalone declaration of "no". The XML parser will accept external resources, if there are any, without reporting an error.

### See Also

[Document Map](#)
Processing Instructions

Processing instructions can be used to pass information to applications in a way that escapes most XML rules. Processing instructions do not have to follow much internal syntax, can include markup characters without escaping them, and can appear anywhere in the document outside of other markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document. Their appearance is not noted by schema or DTD processors.

Processing instructions must begin with an identifier called a target, which follows rules similar to those for element and attribute names. Processing instruction targets are case-sensitive and must start with a letter or underscore. The rest of the target can contain letters, digits, hyphens, underscores, periods, and colons. Any valid XML textual characters can appear after the target.

Note The World Wide Web Consortium (W3C) Namespaces in XML recommendation suggests you avoid the use of colons in processing instruction names.

The following is the processing instruction syntax in the document map.

`<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>`

Processing of the contents ends immediately after the string `?>` is encountered.
Style Sheet Processing Instructions

In accordance with the W3C recommendation, Microsoft® Internet Explorer implements an xml-stylesheet processing instruction. This processing instruction must appear in the prolog, before the document or root element. Multiple processing instructions can appear, which can be useful with cascading style sheets, but most browsers use the first supported style sheet and ignore the rest.

The syntax for a style sheet processing instruction is as follows:

```xml
<?xml-stylesheet type="type" href="uri" ?>
```

where

- *type* is text/css (to link to a cascading style sheet file) or text/xsl (to link to an XSLT file)
- *uri* is the Uniform Resource Identifier (URI) of the style sheet. This URI is relative to the location of the XML document itself.

The following is an xml-stylesheet processing instruction identifying a style sheet built using a cascading style sheet.

```xml
<?xml-stylesheet href="/style.css" type="text/css" title="default stylesheet" ?>
```

The following is an xml-stylesheet processing instruction identifying a style sheet built using Extensible Stylesheet Language (XSL).

```xml
<?xml-stylesheet href="/style.xsl" type="text/xsl" title="default stylesheet" ?>
```

The following processing instruction is legal, but difficult for applications to process.

```xml
<?style - oh, try /style.css ?>
```

Many processing instructions use pseudo-attributes, content that looks like attributes and is easier for applications to process. The preceding example can be rendered using the W3C xml-stylesheet processing instruction:
You can use processing instructions in the prolog to specify other document-wide processing, but you must write an application to support them.

**Note**  Although the use of processing instructions is sometimes discouraged, their loose structure and the flexibility of their placement make them useful for sending messages to an application without disrupting the flow of XML information.

**See Also**

[Document Map](#)

**Other Resources**

[World Wide Web Consortium Namespaces in XML Recommendation](#)
[Associating Style Sheets with XML Documents, Version 1.0 (W3C Recommendation 29 June 1999)](#)
DOCTYPE Declaration

The DOCTYPE declaration provides a space for a document to identify its root element and document type definition (DTD) by reference to an external file, through direct declarations, or both.

A DOCTYPE declaration can contain:

- The name of the document or root element.
  
  This is required if the DOCTYPE declaration is used.

- System and public identifiers for the DTD that can be used to validate the document structure.

  If a public identifier is used, a system identifier must also be present.

- An internal subset of DTD declarations.

  The internal subset appears between square brackets ([ ]).

A DOCTYPE declaration is mandatory if the document is to be processed in a validating environment. To be valid, the DOCTYPE declaration must identify a DTD that corresponds to the document structure of the document. Nonvalidating parsers will accept documents without DOCTYPE declarations.

The simplest DOCTYPE declaration identifies only the root element of the document.

<!DOCTYPE rootElement>

More often, documents that use the DOCTYPE declaration reference an external document containing the declarations that make up the DTD. The following can be used to identify the external DTD.

<!DOCTYPE rootElement SYSTEM "URIreference">

The URIreference points to a file containing the declarations.
<!DOCTYPE rootElement PUBLIC "PublicIdentifier" "URIreference">

The PublicIdentifier provides a separate identifier that some XML parsers can use to reference the DTD in place of the URIreference. This is useful if the parser is used on a system without a network connection or where that connection would slow down processing significantly.

DOCTYPE declarations can also include declarations directly, in what is referred to as the internal subset. If a DOCTYPE declaration includes the entire DTD directly, without reference to external files, it uses the following syntax.

<!DOCTYPE rootElement [declarations ]>

If the DOCTYPE declaration includes declarations that are to be combined with external files or the external subset, it uses the following syntax.

<!DOCTYPE rootElement SYSTEM "URIreference"[declarations ]>

or

<!DOCTYPE rootElement PUBLIC "PublicIdentifier" "URIreference"[declarations ]>
Comments

Content that is not intended for the XML parser, such as notes about document structure or editing, can be included in a comment. Comments begin with a `<!--` and end with a `-->`, for example, `<!-- catalog last updated 2000-11-01 -->`.

Comments can appear in the document prolog, including the document type definition (DTD); after the document; or in the textual content. Comments cannot appear within attribute values. They cannot appear inside of tags.

The parser considers the comment finished when it encounters a `-->`; it then resumes processing the document as normal XML. For this reason, the string `-->` cannot appear inside of a comment. Apart from that restriction, any legal XML characters can appear in a comment, much like a CDATA section. This makes them very useful for removing XML comment from the stream seen by the parser without removing the content from the document.

The following comment can be used to strip out markup temporarily.

`<!---- <test pattern="SECAM" /><test pattern="NTSC" />--->`

**Note** In HTML, you can use comments to hide scripts and style sheets. To use this technique in XML, you might have to retrieve the comment, extract its contents, check for markup characters, and then reparse it. In this case, CDATA sections are a better option.
Because of its support for the Unicode character set, XML supports a range of characters, including letters, digits, punctuation, and symbols. Most control characters and Unicode compatibility characters are not allowed. For a complete list of character classes, see Appendix B of the XML specification.

**Note** Because XML relies on `<`, `>`, and `&` to delimit markup, you should represent these characters using the character and entity references described in the following topic, or confine them to CDATA sections.

**See Also**

*CDATA Sections* | *Character and Entity References*

**Other Resources**

*Appendix B, Character Classes*
Character and Entity References

Character and entity references provide ways to include information in XML documents by reference rather than by typing characters into the document directly. This can be useful in cases in which:

- Characters cannot be entered directly into a document because they would be interpreted as markup.
- Characters cannot be entered directly into a document because of input device limitations.
- Characters cannot be transported reliably through a processor limited to one-byte characters.
- A character string or document fragment appears repeatedly and can be abbreviated.

For content representation, XML provides a number of syntactical constructs that start with an ampersand (&) and end with a semi-colon (;).

Character references provide a way to insert Unicode characters that are identified by a number pointing to a Unicode code point. Code points can be identified using either decimal or hexadecimal notation.

\&#value;
Syntax used for decimal references.

\&#xvalue;
Syntax used for hexadecimal references.

For example, to insert the Euro symbol, a character still missing from many keyboards, you can insert \&#x20AC; or \&#8364; into a document.

The following table lists the five built-in entities for the characters used for XML markup.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Entity Reference</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lt</td>
<td>&lt;</td>
<td>&lt; (less than)</td>
</tr>
<tr>
<td>gt</td>
<td>&gt;</td>
<td>&gt; (greater than)</td>
</tr>
</tbody>
</table>
amp                &amp;                & (ampersand)
apos               &apos;               ' (apostrophe or single quote)
quot               &quot;               " (double quote)

In cases where the character might cause the XML parser to misinterpret the document structure, use the entity instead of typing the character. The &apos; and &quot; entity references are most commonly used in attribute values.

To write Me&amp;You, for example, use Me&amp;amp;You. For a&lt;b, use a&lt;lt;b. For b&gt;c, use b&gt;gt;c.

You can also define your own entities, much like HTML defines a set of entities for use in HTML. &apos; is not recognized as an HTML file; $#.... must be used when transforming to HTML.

If you're working with a document type definition (DTD) that has defined entities, you can reference them in document content by using the following syntax.

&entityName;
Character Encoding, XML, and MSXML

Character encodings provide a map between a series of numbers and the characters people expect to see when they enter text into computers. The capital letter "A", for example, is represented by the decimal number 65 (41 in hexadecimal) in a variety of character encodings, including the ASCII text familiar to many Western programmers and Windows Code Page-1252, the default encoding used by most Microsoft® Windows® Western systems.

Character encodings are not fonts, which provide graphic representations, glyphs, that map to a particular character encoding. Microsoft Word, for example, includes a version of Arial (Arial Unicode MS) with tens of thousands of characters.

All XML processors are required to understand two transformations of the Unicode character encoding, UTF-8 and UTF-16. Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office supports more encodings, but all text in XML documents is treated internally as the Unicode UCS-2 character encoding.

Even different platforms representing the same set of Western characters can use different bytes to represent the same character, as shown in the following table.

<table>
<thead>
<tr>
<th>Byte</th>
<th>Windows (CP1252)</th>
<th>Macintosh (MacRoman)</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>çE</td>
<td>å</td>
</tr>
<tr>
<td>229</td>
<td>å</td>
<td>Å</td>
</tr>
<tr>
<td>231</td>
<td>ç</td>
<td>Á</td>
</tr>
<tr>
<td>232</td>
<td>è</td>
<td>È</td>
</tr>
<tr>
<td>233</td>
<td>é</td>
<td>È</td>
</tr>
</tbody>
</table>

Parsers can read in documents written ISO-8859-1, Big-5, or Shift-JIS, but the processing rules treat everything as Unicode. MSXML and other XML parsers perform the conversion while loading XML documents.
There are some limitations to auto-detecting character encodings. For example, 8-bit ASCII text is acceptable UTF-8, but UTF-8 is more than 8-bit ASCII text. For reliable processing, XML documents that use character encodings other than UTF-8 or UTF-16 must include an encoding declaration in the XML declaration. This makes it possible for a parser to read the characters correctly or report errors when it cannot process an encoding.

Because the XML declaration is written in basic ASCII text, parsers can read its contents even if the document is in a very different encoding. The encoding declaration significantly increases the likelihood that documents in encodings other than UTF-8 and UTF-16 will be interpreted correctly.

Some transactions, for example, those carried over HTTP and e-mail protocols, also provide information about character encodings. Microsoft Internet Explorer uses that information in document processing, but it isn't available, for example, if you load an XML document from a local hard drive or even a file server.
XML Data Islands and Character Encodings

XML data islands inside of HTML documents receive different encoding handling, depending on whether the content is stored directly within the `<xml>` element or referenced by a `src` attribute.

XML data islands that contain the XML directly in the HTML document use the encoding of the surrounding HTML document. Because the entire document is assumed to use the same encoding, this approach significantly simplifies parsing.

XML data islands that reference the content through a `src` attribute are less constrained. The XML documents they reference can have an XML declaration containing a character encoding, and MSXML will use that character encoding. The contents of the data island will be presented as Unicode, but the parser will handle the conversion automatically.

See Also

XML Declaration

Other Resources

How To Encode XML Data
CDATA Sections

CDATA sections provide a way to tell the parser that there is no markup in the characters contained by the CDATA section. This makes it much easier to create documents containing sections where markup characters might appear, but where no markup is intended. CDATA sections are commonly used for scripting language content and sample XML and HTML content.

The CDATA section in the document map uses the following syntax.

`<![[CDATA[An in-depth look at creating applications with XML, using <]]>`

When an XML parser encounters the initial `<![CDATA[`, it reports the content that follows as characters without attempting to interpret them as element or entity markup. Character references do not work within CDATA sections. When it encounters the concluding `]]>`>, the parser stops reporting and returns to normal parsing.

For example, you can include either of the following CDATA sections in an XML document without causing the parser to report an error.

`<![CDATA[</this is malformed!</malformed</malformed & worse>]]>`

or

`<![CDATA[if (c<10)]]>`

**Note** Content within CDATA sections must be within the range of characters permitted for XML content; control characters and compatibility characters cannot be escaped this way. In addition, the sequence `]]>` cannot appear within a CDATA section because this sequence signals the end of the section. This means that CDATA sections cannot be nested. The sequence also appears in some scripts. Within scripts, it is usually possible to substitute `]]>` for `]]>`.`

**See Also**

Document Map
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Develop
Attributes

Attributes allow you to add information about an element using name-value pairs. Attributes are often used to define properties of elements that are not considered the content of the element, though in some cases (for example, the HTML `img` element) the content of the element is determined by attribute values.

Attributes can appear in start or empty tags, but not in end tags. The syntax looks like:

```xml
<elementName att1Name="att1Value" att2Name="att2Value"...>
```

or:

```xml
<elementName att1Name="att1Value" att2Name="att2Value".../>
```

Attributes must have both a name and a value. No names without values are permitted. An element cannot have two attributes with the same name. Because the order in which attributes appeared inside of an element is not considered important by XML, it might not be preserved by the XML parser.

Like element names, attribute names are case-sensitive and must start with a letter or underscore. The rest of the name can contain letters, digits, hyphens, underscores, and periods.

**Note**  Colons are reserved for use with namespaces. For more information about which Unicode characters are acceptable letters and digits, see Appendix B of the XML specification.

Attribute values must follow the same rules as normal textual content, with a few additional restrictions. Attribute values can contain only text, not element markup. Entity references and character references are allowed within attribute values, but CDATA sections are not.

Finally, quotes might have to be escaped. The XML specification allows you to use either single or double quotes to indicate attributes, though the type of quote used must be the same on both sides of the attribute value. You must use quotes around attribute values, however. XML parsers will simply reject documents that
do not use quotes around attribute values and report an error.

If you use single quotes to indicate an attribute value, you must represent single quotes inside the attribute value using the &apos; entity reference.

```xml
<myElement contraction='isn&apos;t' /> 
```

If you use double quotes to indicate an attribute value, you must represent double quotes inside the attribute value using the &quot; entity reference.

```xml
<myElement question="They asked &quot;Why?&quot;" /> 
```

You can, however, use double quotes inside of single-quoted attribute values and vice-versa, as the following examples demonstrate.

```xml
<myElement contraction="isn't" /> 
<myElement question='They asked "Why?"' /> 
```

You can also use different types of quotes on different attribute values within the same element.

```xml
<myElement contraction="isn't" question='They asked "Why?"' /> 
```

XML parsers do not preserve white space within attribute values the same way that they preserve white space within element textual content. In a validating parser with access to a DTD, all white space within an attribute value of any type other than CDATA is replaced with a single space; starting and ending white space will be stripped.

**Other Resources**

[Appendix B, Character Classes](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
White Space

The World Wide Web Consortium (W3C) XML specification normalizes different line-ending conventions to a single convention but preserves all other white space, except in attribute values. XML also provides a set of tools that documents can use to signal to applications if white space must be preserved.
White Space and the XML Declaration

According to the current XML 1.0 standard, white space is not allowed before the XML declaration.

```xml
<?xml version="1.0"?>
<BOOK>
  <BOOKNAME>XML</BOOKNAME>
</BOOK>
```

If white space appears before the XML declaration, it will be treated as a processing instruction. The information, particularly the encoding, may not be used by the parser.

For more information about the XML declaration, see [XML Declaration](#).
White Space in Element Content

XML parsers are required to report all white space that appears in element content within a document. For this reason, the following three documents are different to an XML parser.

```xml
<document>
<data>1</data>
<data>2</data>
<data>3</data>
</document>

and:

```xml
<document><data>1</data><data>2</data><data>3</data></document>
```

and:

```xml
<document><data>1</data> <data>2</data> <data>3</data></document>
```

For some applications, the values of the three data points matter more than the pretty-printing. For document-oriented XML applications, white space preservation can be critical.

Document authors can use the `xml:space` attribute to identify portions of documents where white space is considered important. Style sheets can also use the `xml:space` attribute as a hook to preserve white space in presentation. However, because many XML applications do not understand the `xml:space` attribute, its use is considered advisory.

The `xml:space` attribute accepts two values.

- `default`
  - This value allows the application to handle white space as necessary. Not including an `xml:space` attribute produces the same result as using the `default` value.
- `preserve`
  - This value instructs the application to maintain white space as is, suggesting that it might have meaning.
The values of xml:space attributes apply to all descendants of the element containing the attribute unless overridden by one of the child elements.

For example, the following documents specify the same white space behavior.

```xml
<poem xml:space="default">
  <author>
    <givenName>Alix</givenName>
    <familyName>Krakowski</familyName>
  </author>
  <verse xml:space="preserve">
    <line>Roses are red,</line>
    <line>Violets are blue.</line>
    <signature xml:space="default">-Alix</signature>
  </verse>
</poem>

and:

```xml
<poem xml:space="default">
  <author xml:space="default">
    <givenName xml:space="default">Alix</givenName>
    <familyName xml:space="default">Krakowski</familyName>
  </author>
  <verse xml:space="preserve">
    <line xml:space="preserve">Roses are red,</line>
    <line xml:space="preserve">Violets are blue.</line>
    <signature xml:space="default">-Alix</signature>
  </verse>
</poem>
```

In both examples, the application is notified that all of the white space in the lines of the poem must be preserved, but that white space in other parts of the document can be handled as necessary.

Like its language-indicating counterpart, xml:lang, the xml:space attribute must be declared in a document type definition (DTD) if used in a validating environment. The xml namespace does not need to be declared because it is reserved by the XML specification.

By default, Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office does not honor the xml:space attribute. If an application must honor the xml:space attribute, the preserveWhiteSpace property of the DOMDocument object must be set to True prior to parsing.
MSXML also provides settings that let you delegate application white space handling to the parser. For more information, see White Space and the DOM.

**Note** Preserving white space information can significantly increase the size of Document Object Model (DOM) trees because of the overhead involved in preserving white space nodes between elements.
White Space in Attributes

Although XML processors preserve all white space in element content, they frequently normalize it in attribute values. Tabs, carriage returns, and spaces are reported as single spaces. In certain types of attributes, they trim white space that comes before or after the main body of the value and reduce white space within the value to single spaces. (If a DTD is available, this trimming will be performed on all attributes that are not of type CDATA.)

For example, an XML document might contain the following:

```xml
<whiteSpaceLoss note1="this is a note." note2="this is a note."/>
```

An XML parser reports both attribute values as "this is a note.", converting the line breaks to single spaces.

If there is a DTD for the document, attributes that are declared to be of types other than CDATA have spaces removed from the beginning and end of the attribute value; all white space clusters inside the value are replaced with single spaces. If there is no DTD, the parser assumes that all attributes are of type CDATA.
End of Line Handling

XML processors treat the character sequence Carriage Return-Line Feed (CRLF) like single CR or LF characters. All are reported as a single LF character. Applications can save documents using the appropriate line-ending convention.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Using XML with HTML

The following topics discuss ways to extend HTML, and to format and transform XML.

- Transforming and Querying XML
- Formatting XML Documents
- Augmenting HTML
- Parsers and Parser Types
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Transforming and Querying XML

As XML becomes a standard way to exchange data on the Web, the need arises for mechanisms to query XML, sort, filter, and shape extracted data, and transform one XML grammar into another.

The XML Path Language (XPath) provides a simple and concise syntax for identifying nodes in an XML document, based on the node's type, name, content, and context in relation to other nodes in the tree.

XSL Transformations (XSLT) provides a grammar in which the results of XPath queries are associated with templates to describe the materialization of data in the XML source document as a new XML document. While this forms the basis for transforming data to display formats such as HTML, any XML grammar can be output, providing for sorting and filtering within a single XML grammar, or translating data from one schema to another.

The World Wide Web Consortium (W3C) is considering a more powerful query language for XML.

See Also

XSLT Developer's Guide | Using XPath Expressions to Select Nodes
Formatting XML Documents

Although Microsoft® Internet Explorer 5.0 and later provides a default style sheet that is useful for exploring document structures, some applications must present the information stored in XML documents directly to the user, without relying on scripts or additional processing code. Internet Explorer 5.0 and later provides support for cascading style sheets (CSS), and Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office provides enhanced support for the Extensible Stylesheet Language (XSL).
XML and Cascading Style Sheets

Cascading style sheets allow developers to describe formatting that should be applied to document structures. Cascading style sheets let document designers make statements like "all paragraph elements should be formatted as separate blocks, with a 24 pixel indent on the first line" or "all price elements should be presented in bold green 12 point sans-serif type except price elements with a status attribute of sale, which should be presented in red."

CSS is commonly described as an annotative style language, meaning that it adds formatting information to the document tree rather than changing the document itself. Its lists of formatting rules can be combined or overridden to provide multiple layers of formatting information appropriate to different document variations without requiring change to the document structure itself.
XML, XSL, and XSLT

XSL provides a set of tools for transforming documents from their original labeled document structure to a new structure that can be used for presentation. Typically, developers create transformations from an XML vocabulary to HTML, though a formatting object vocabulary is under development at the World Wide Web Consortium (W3C).

XSL is commonly described as a transformative style language. Instead of adding information to the original document structure, it creates a new document structure based on rules applied to the content of the original. The transformation language used by XSL, XSL Transformations (XSLT), allows developers to create sophisticated templates detailing how the original document's information should be presented in the new document, known as the result document.

See Also

Augmenting HTML
Augmenting HTML

Adding semantic information to HTML pages is not easy. Historically, various programs have attempted to deal with this problem by using nonstandard tricks, such as hiding data inside HTML comments. However, these comments are awkward and, unlike XML, are not exposed to the object model.

To solve this, the World Wide Web Consortium (W3C) has defined a format for putting XML-based data (data islands) inside HTML pages. Extending HTML through the use of data islands will allow a wide range of applications to use HTML as the primary document or display format, and also to use XML embedded within these documents to hold data.

An HTML page can therefore include, among other things, specific data about the subject of the page. For example, if the page displayed an advertisement for an author's most recent novel, the page can also contain XML data concerning that book, such as its ISBN number, publisher, or suggested retail price. It is not important that this information be displayed, but it is important that this information be accessible and understandable as data.

See Also

XML Data Islands
Parsers and Parser Types

From a developer's point of view, XML parsers are the fundamental XML component, a bridge between XML documents, seen as a long chain of bytes, and applications that process that XML. Almost all XML applications are built on top of parsers.

The parser is responsible for handling XML syntax and, if desired, checking the contents of the document against constraints established in a document type definition (DTD) or schema; the application must understand how to process or display the information. The application is insulated from the details of the XML document, allowing document creators to take advantage of those details without worrying about the application. Document creators must still present correctly structured information to an application, but they can do so using generic tools instead of creating new interfaces for every different transaction.

There are two basic kinds of XML parsers defined in the XML 1.0 specification. Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office can operate in either validating or nonvalidating mode.

**nonvalidating parsers**
Check document syntax and report all violations of well-formedness constraints. Nonvalidating parsers can also add information to the document based on declarations in the DTD. MSXML does read the DTD, including external resources, and acts on that information.

**validating parsers**
Perform the same functions as nonvalidating parsers, but also compare the structures of documents to rules in the DTD.

The needs of your application will determine whether you use validating or nonvalidating mode. If you're building a generic XML application that doesn't expect to see a particular set of document structures, a nonvalidating parser will allow you to work with documents that weren't built to be valid. By default, Microsoft Internet Explorer uses nonvalidating mode to accommodate the largest possible number of documents.

On the other hand, if you're creating systems for exchanging information with
other organizations, and have had to agree on the format and content of those exchanges, DTDs (or schemas) and validating parsers are a good choice. They provide an extra layer of processing that lets you trim a lot of structure-checking code from your programs, while ensuring that document structures conform to the formats you agreed on prior to the transaction.

In many ways, DTDs and schemas are like contracts, and validating parsers enforce the terms of those contracts.
Testing for well-formedness with Internet Explorer 5.5

You can open any XML document labeled as XML in Internet Explorer. If the document is well-formed, Internet Explorer will display the document according to the style sheet specified within the document. If the document does not specify a style sheet, Internet Explorer will display the document using its default style sheet for XML. If the document is not well-formed, Internet Explorer will report an error. If a document has multiple errors, you might have to fix an error, reopen the document, and fix the next error.

Note To label a document as XML, assign it a file extension of .xml if you are opening it from a local file system or file server, or a Multipurpose Internet Mail Extensions (MIME) type of text/xml or application/xml if you are opening it over a Web connection.
Testing for validity with Internet Explorer 5.5

To use Internet Explorer to validate an XML document, you must set a switch in the parser before loading the document.

Microsoft offers an XML validator that sets these switches and provides an interface. This validator allows you to enter an XML document either as a URL or through a form. The validator parses the document using the MSXML parser in validating mode and reports any errors.

The core of the validator is this snippet of Microsoft JScript® code.

```javascript
xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.validateOnParse = true;
xmlDoc.load(url);
```

The code appears as follows in Microsoft Visual Basic® Scripting Edition (VBScript).

```javascript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmlDoc.validateOnParse = true
xmlDoc.load(url)
```

Other Resources

XML Validator
Using Namespaces in Documents

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office supports the World Wide Web Consortium (W3C) Namespaces in XML Recommendation specification. By using XML namespaces to qualify element names, you can avoid naming conflicts for elements that have the same name. Associating a Uniform Resource Identifier (URI) with a namespace ensures that only elements with the same name remain unambiguous.

Namespace support is integrated into the MSXML parser, including the Document Object Model (DOM), Extensible Stylesheet Language (XSL), XML Data Reduced (XDR) schemas, and XML schemas (XSD).

The following topics provide more information about namespaces, and describe the namespace support features in MSXML.

- Reasons for Namespaces
- Namespace Prefixes
- Declaring Namespaces
- Namespace Scope

Other Resources

W3C Namespaces in XML Recommendation
Reasons for Namespaces

The appeal of XML lies in the ability to invent tags that convey meaningful information that can be processed by applications and read by users. Namespaces allow you to associate a vocabulary (or namespace) with an element or attribute name.

Example

The following XML represents information about a book.

```xml
<BOOK>
  <TITLE>XML Developer's Guide</TITLE>
  <PRICE currency="US Dollar">44.95</PRICE>
</BOOK>
```

The following XML represents information about an author.

```xml
<AUTHOR>
  <TITLE>Ms</TITLE>
  <NAME>Ambercrombie Kim</NAME>
</AUTHOR>
```

Although the user can distinguish between the different interpretations of the `<TITLE>` element, a computer program, such as an XML parser, cannot differentiate between the two meanings. Without additional information, the parser cannot determine that the first `<TITLE>` element is intended to refer to a string representing the title of the book, and that the second refers to an enumeration representing the title of the author ("Mr.", "Ms.", "Mrs.", and so on).

Example

The following examples shows TITLE associated with BookInfo and AuthorInfo namespaces.

```xml
<AuthorInfo:TITLE xmlns:AuthorInfo="authors-namespace-URI">Ms.</AuthorInfo:TITLE>
```
The W3C Namespaces in XML recommendation provides the xmlns attribute to uniquely define a namespace for an XML document to use.

**Example**

The following examples show how to declare a namespace with the xmlns attribute.

```xml
xmlns=http://www.example.microsoft.com/books.dtd
xmlns="urn:schemas-microsoft-com:xml-data"
```
Namespace Prefixes

The name preceding the colon, the prefix, maps to an XML namespace identified by a Universal Resource Identifier (URI). The namespace ensures global uniqueness when merging XML sources, while the associated prefix—a short name that substitutes for the namespace URI—must be unique only in the tightly scoped context of the document. With this scheme, no conflicts exist between tags and attributes, and two tags can be the same only if they are from the same namespace and have the same tag name. This allows a document to contain both book and author information without confusion about whether the `<TITLE>` element refers to the book or the author. If a computer program wanted to display the name of a book in a user interface, it would use the object model to look for the `<TITLE>` element of the BookInfo namespace.

Example

The following examples show the `tiger` element prefixed with `zoo`. This indicates that the animal element belongs to the `zoo` namespace as defined in the root element by the `xmlns` attribute.

```xml
<zoo:tiger>
```

Other Resources

[W3C Namespaces in XML Recommendation](https://www.w3.org/TR/REC-xml-names/)

Declaring Namespaces

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office provides support for associating elements and attributes with namespaces. You can use a default or explicit declaration to declare namespaces. In both cases, the declaration associates a Uniform Resource Identifier (URI) with particular element and attribute names.
Choosing Namespace URIs

Developers creating new XML vocabularies might need to choose URIs for use as namespace identifiers. The following are general guidelines for creating new vocabularies that need namespace identifiers.

- Use namespace URIs that you control.
  
  Although the Namespaces in XML recommendation does not prohibit borrowing, do not create a new namespace using a URI you do not control.

- Use URIs that are persistent.
  
  Although they are usually maintained, domain names do expire. Other URI facilities, such as Uniform Resource Names (URNs) and Permanent URLs (PURLs), guarantee persistence beyond the domain name infrastructure.

- Use URIs that consistently point to the same location.
  
  Although the Namespaces in XML recommendation does not prohibit the use of relative URI references in namespace identifiers, their use is largely undefined.

- Identify and describe the namespace URI in the documentation for your vocabulary.
Default Declaration

The default declaration declares a namespace in effect for an element and all other elements contained within it where the element is not prefixed. The following example declares the `<catalog>` element—and all its element descendents that do not have a prefix—to be members of the namespace declared using the URI http://www.example.microsoft.com/catalog/.

Example

```xml
<catalog xmlns="http://www.example.microsoft.com/catalog/">
  <book id="bk101">
    ...
  </book>
  <book id="bk109">
    ...
  </book>
</catalog>
```

If the URI reference is an empty string, unprefixed elements within the scope of the declaration are not considered to belong to any namespace. Default declarations are commonly used when a document contains only elements from a particular namespace, or when one namespace effectively dominates all others.

**Note**  Default namespaces do not directly apply to attributes.
Explicit Declaration

An explicit declaration defines a shorthand, or prefix, to substitute for the full name of a namespace. Use an explicit declaration to reference a node from a namespace separate from your default namespace.

If the catalog example created in the document map had to represent its currency values using an element from a different namespace, it might include both a declaration for the catalog as a whole and an explicit declaration for the element describing the prices.

Explicit declarations of namespace prefixes use attribute names beginning with the xmlns attribute and followed by the prefix. The value of the attribute is the namespace URI.

Example

The following example declares cat and money to be shorthand for the full names of their respective namespaces. All elements beginning with cat: or money: are considered to be from the namespace
"http://www.example.microsoft.com/catalog/" or "http://www.example.microsoft.com/currency/", respectively.

```xml
<cat:catalog xmlns:cat="http://www.example.microsoft.com/catalog/"
             xmlns:money="http://www.example.microsoft.com/currency/">
  <cat:book id="bk101">
    <cat:author>&#71;ambardella, Matthew</cat:author>
    <cat:genre>Computer</cat:genre>
    <money:price>44.95</money:price>
    <cat:publish_date>2000-10-01</cat:publish_date>
    <cat:description><![CDATA[An in-depth look at creating applications]]>
  </cat:book>
  <cat:book id="bk109">
    <cat:author>Kress, Peter</cat:author>
    <cat:title>Paradox Lost</cat:title>
    <cat:genre>Science Fiction</cat:genre>
    <money:price>6.95</money:price>
    <cat:publish_date>2000-11-02</cat:publish_date>
    <cat:description>After an inadvertent trip through a Heisenberg</cat:book>
```
Explicit declarations are useful when a node contains elements from different namespaces. You can also fix default namespace declarations with explicit namespace declarations.

Example

The following example is almost identical to the preceding example. However, all of the elements except price use the default namespace declaration, avoiding repeated cat: prefixes. All of the elements except price are associated with the namespace URI "http://www.example.microsoft.com/catalog/"; price is associated with "http://www.example.microsoft.com/currency/".

```xml
<catalog xmlns="http://www.example.microsoft.com/catalog/
         xmlns:money="http://www.example.microsoft.com/currency/">
  <book id="bk101">
    <author>ambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <money:price>44.95</money:price>
    <publish_date>2000-10-01</publish_date>
    <description><![CDATA[An in-depth look at creating applications]]></description>
  </book>
  <book id="bk109">
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <money:price>6.95</money:price>
    <publish_date>2000-11-02</publish_date>
    <description>After an inadvertent trip through a Heisenberg Unc</description>
  </book>
</catalog>
```

See Also

[Document Map]
Namespace Scope

Namespace declarations are scoped using an element container. All namespace prefixes are valid only within the element in which they are defined. Unqualified elements (those without an explicit namespace on the element) use the default namespace.

Example

The following example shows unqualified elements, <book id="bk101"> and <book id="bk109"> within the "http://www.example.microsoft.com/catalog/" namespace.

```xml
<catalog xmlns="http://www.example.microsoft.com/catalog/">
  <book id="bk101">
    ...
  </book>
  <book id="bk109">
    ...
  </book>
</catalog>
```

Example

Note, however, that unqualified attributes do not belong to the default namespace. Therefore, the id attribute in the previous example does not belong to any namespace. In order for the id attribute to belong to the same namespace as the book element, prefix it like this:

```xml
<catalog xmlns="http://www.example.com/catalog/
          xmlns:cat="http://www.example.com/catalog/">
  <book cat:id="bk101">
    ...
  </book>
  <book cat:id="bk109">
    ...
  </book>
</catalog>
```
Example

In the following example, all unqualified elements within the `<table>` element are scoped to the default namespace "http://www.w3.org/TR/REC-html40". Within the `<details>` tag, the default namespace is set to "urn:example.microsoft.com:beerinfo". All unqualified tags within the `<details>` element are scoped to this namespace.

```xml
<Beers>
<!-- the default namespace is now that of HTML -->
<table xmlns="http://www.w3.org/TR/REC-html40">
<tr><td>Name</td><td>Origin</td><td>Description</td></tr>
<tr>
<!-- drop the HTML namespace inside table cells -->
<td><brandName xmlns="urn:example.microsoft.com:beerinfo">Coho</brandName></td>
<td><origin xmlns="urn:example.microsoft.com:beerinfo">Bath, UK</origin></td>
<details xmlns="urn:example.microsoft.com:beerinfo">
<class>Bitter</class>
<hop>Fuggles</hop>
<pro>Wonderful hop, light alcohol, good summer beer</pro>
<con>Fragile; excessive variance pub to pub</con>
</details>
</td>
</tr>
</table>
</Beers>
```

The following table shows the elements and attributes of the preceding XML example and their properties as presented by Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office.

<table>
<thead>
<tr>
<th>Node type</th>
<th>nodeName</th>
<th>namespaceURI</th>
<th>prefix</th>
<th>baseName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Beers</td>
<td>n/a</td>
<td>n/a</td>
<td>Beers</td>
</tr>
<tr>
<td>Element</td>
<td>table</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>table</td>
</tr>
<tr>
<td>Attribute</td>
<td>xmlns</td>
<td>n/a</td>
<td>n/a</td>
<td>xmlns</td>
</tr>
<tr>
<td>Element</td>
<td>tr</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>tr</td>
</tr>
<tr>
<td>Element</td>
<td>td</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>td</td>
</tr>
<tr>
<td>Element</td>
<td>td</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>td</td>
</tr>
<tr>
<td>Element</td>
<td>td</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>td</td>
</tr>
<tr>
<td>Element</td>
<td>tag</td>
<td>namespace</td>
<td>attributes</td>
<td>value</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>tr</td>
<td>tr</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>tr</td>
</tr>
<tr>
<td>td</td>
<td>td</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>td</td>
</tr>
<tr>
<td>td</td>
<td>td</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>td</td>
</tr>
<tr>
<td>td</td>
<td>td</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td>n/a</td>
<td>td</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>name</th>
<th>namespace</th>
<th>attributes</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlns</td>
<td>xmlns</td>
<td>n/a</td>
<td>n/a</td>
<td>xmlns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>tag</th>
<th>namespace</th>
<th>attributes</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>origin</td>
<td>urn:example.microsoft.com:beerinfo</td>
<td>n/a</td>
<td>origin</td>
</tr>
<tr>
<td>details</td>
<td>details</td>
<td>urn:example.microsoft.com:beerinfo</td>
<td>n/a</td>
<td>details</td>
</tr>
<tr>
<td>class</td>
<td>class</td>
<td>urn:example.microsoft.com:beerinfo</td>
<td>n/a</td>
<td>class</td>
</tr>
<tr>
<td>hop</td>
<td>hop</td>
<td>urn:example.microsoft.com:beerinfo</td>
<td>n/a</td>
<td>hop</td>
</tr>
<tr>
<td>pro</td>
<td>pro</td>
<td>urn:example.microsoft.com:beerinfo</td>
<td>n/a</td>
<td>pro</td>
</tr>
<tr>
<td>con</td>
<td>con</td>
<td>urn:example.microsoft.com:beerinfo</td>
<td>n/a</td>
<td>con</td>
</tr>
</tbody>
</table>
Using XML on the Server

The XML engine included in Microsoft® Internet Explorer 5.0 and later can also be used on the server. This allows data-rich Web applications in which XML data can be created, manipulated, and transmitted between client and server.

The following topics provide an introduction to the use of the Internet Explorer 5.0 XML engine on the server.

- Instantiating the XML Control
- Choosing a Threading Model
- Getting the XML Data
- Sending XML Between the Client and Server
- Saving XML
- Converting XML to HTML
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Instantiating the XML Control

Installing Microsoft® Internet Explorer 5.0 and later on the server installs the msxml.dll, which contains all the components of the XML engine, including Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office, the Extensible Stylesheet Language (XSL) processor, schema support, and so on.

Instantiate the XML control the way you would any other control on the server. For example, from an Active Server Pages (ASP) page you would use code like the following.

```javascript
var xObj = Server.CreateObject("Msxml2.DOMDocument.5.0");
```

Alternatively, a server-side object can be used.

```xml
<OBJECT RUNAT="server" PROGID="msxml2.DOMDocument.5.0" id="xObj"> </OBJECT>
```

The xObj is an empty `IXMLDOMDocument/DOMDocument` object, ready to build in memory or load from an XML file or XML stream.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Choosing a Threading Model

There are two versions of the XML control.

rental model
   The rental model version is designed for single-threaded access. To use the rental model control provided with MSXML 5.0, use the Microsoft.DOMDocument.5.0 progID.

free-threaded model
   The free-threaded model version is designed for multiple thread access. To use the free-threaded control provided with MSXML 5.0, use the Microsoft.FreeThreadedDOMDocument.5.0 progID.

   Note In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

If you plan for several threads to access your XML data from a single control, be sure to use the free-threaded control. If only one thread will access the XML data, use the rental model control for better performance.

The following is a sample global.asa file that creates session-level and application-level free-threaded versions of the XML control.

<SCRIPT LANGUAGE=VBSRcript RUNAT=Server>
Sub Session_OnStart
   ON error RESUME next

   SET Application("AppFXMldoc") = _
       server.CreateObject("Msxml2.FreeThreadedDOMDocument.5.0")
   SET Session("SessFXMldoc") = _
       server.CreateObject("Msxml2.FreeThreadedDOMDocument.5.0")
End Sub
Sub Session_OnEnd
   ON error RESUME next

   SET Session("SessFXMldoc") = nothing
   SET Application("AppFXMldoc") = nothing
   Session("SessFXMldoc") = empty
   Application("AppFXMldoc") = empty
End Sub
</SCRIPT>
Scripts accessing the Session and Application objects will be able to simultaneously access the AppFXMLdoc and SessFXMLdoc objects.
Getting the XML Data

The following are ways to get XML data on the server.

- Use Microsoft® ActiveX® Data Objects (ADO) 2.1 or later to get XML information directly from any database that supports ADO.
- Read data from a database or other source and construct an XML tree using the Document Object Model (DOM).
- Load an XML file using the XML control's load method.
- Construct a string and load it into the XML control using the loadXML method.
- Construct an XML text string directly, without building an XML document object on the server.

Some of these methods are illustrated in the following sections.
Constructing an XML Tree Using the DOM

The XML DOM provides the ability to programatically construct an XML document from data read from a database or other data source.

The following code reads data from a database and constructs an XML document using the DOM. The document is then sent to the client as an XML text stream.

```xml
<% Response.ContentType = "text/xml" %>
<?xml version="1.0" encoding="windows-1252" ?>
<%
' Check to see if a document has data. If it does, don't build it
if (xmldoc.childNodes.length = 0) then
' Build the XML document
    set root = xmldoc.createNode("element","SalesData","")
    xmldoc.appendChild(root)
' Queries the database for customer data
    SQL = "SELECT OrderID, CompanyName, OrderDate, ShippedDate, " & _
          "  ShipCity, ShipState " & _
          " FROM Customers " & _
          "INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID"
    set Conn = Server.CreateObject("ADODB.Connection")
    Conn.Open("pdc")
    set rs = Conn.Execute(SQL)
    ocount = 0
    rs.MoveFirst()
    while (not rs.EOF and ocount < 10)
        set onode = xmldoc.createNode("element","Order","")
        xmldoc.documentElement.appendChild(onode)
        for i = 0 to rs.Fields.Count - 1
            set child = xmldoc.createNode("element",rs.Fields(i).Name,"")
            child.text = rs.Fields(i)
            onode.appendChild(child)
        next
    pcount = 0
    set isnode = xmldoc.createNode("element","Items","")
    onode.appendChild(isnode)
    iSQL = "select ProductName, OrderDetails.UnitPrice as UnitPrice, " & _
          "  Quantity from OrderDetails" & _
          " inner join Products on Products.ProductID = OrderDetails.ProductID"
    iSQL = iSQL & rs("OrderID")
    set irs = Conn.Execute(iSQL)
    irs.MoveFirst()
    while (not irs.EOF and pcount < 3)
        set inode = xmldoc.createNode("element","Item","")
```
isnode.appendChild(inode)
for i = 0 to irs.Fields.Count - 1
    set child = xmldoc.createNode("element",irs.Fields(i).Name,"
    child.text = irs.Fields(i)
    inode.appendChild(child)
next
irs.MoveNext()
pcount = pcount + 1
wend
rs.MoveNext()
ocount = ocount + 1
wend
end if
Response.Write(xmldoc.xml) %>

The DOM methods createNode and appendChild and the text property are used to construct the XML tree. The xml property writes the document out as an XML stream, which is then sent to the client.
Loading XML from a File

The DOM load method can be used to build a DOM document from a URL.

```javascript
xObj = Server.CreateObject("Msxml2.DOMDocument.5.0");
xObj.load(Server.MapPath("simple.xml"));
```

The `MapPath` method resolves the relative path to a full path.
Directly Constructing XML Text

If you do not need to process or cache the XML on the server, you can choose to simply construct the XML text without using the XML control. This does not require the XML control to be installed on the server.

The following example reads data from a database and injects it into the output stream.

```vbscript
<% Response.ContentType = "text/xml" %>
<?xml version="1.0" encoding="windows-1252" ?>
<SalesData>
<%
SQL = "SELECT OrderID, CompanyName, OrderDate, ShippedDate, " & _
"ShipCity, ShipState " & _
" FROM Customers " & _
"INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID "
"ORDER BY ShipState"
set Conn = Server.CreateObject("ADODB.Connection")
set rs = Server.CreateObject("ADODB.Recordset")
Conn.Open("pdc")
set rs = Conn.Execute(SQL)
rs.MoveFirst()
while (not rs.EOF)
<%>
<Order>
<OrderID> <%= rs("OrderID") %> </OrderID>
<Customer> <%= rs("CompanyName") %> </Customer>
<OrderDate> <%= rs("OrderDate") %> </OrderDate>
<ShippedDate> <%= rs("ShippedDate") %> </ShippedDate>
<ShipCity> <%= rs("ShipCity") %> </ShipCity>
<ShipState> <%= rs("ShipState") %> </ShipState>
</Order>
<%>
    rs.MoveNext()    
wend
rs.close()
conn.close()
%
</SalesData>
```

Because the XML tags are generated directly as text, the overhead of building the DOM document object is avoided.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Sending XML Between the Client and Server

When sending XML to the client, you must first create a textual version of the XML.

To create the textual version, you can use the `xml` property of the root node of the XML document, as shown in the following fragment.

```csharp
Response.Write(xObj.xml);
```

You can also use the technique described in [Getting the XML Data](#).

After you have the XML text string, you can use one of the following approaches to send it to the client in response to a client request.

- Maintain the XML as a file on the server. The client can load this file directly using the `load` method, as shown in the following.
  
  ```csharp
  xObj.load("http://www.microsoft.com/sample.xml");
  ```

- Create and return the XML from an Active Server Pages (ASP) page. The client can load the result using the `load` method, as shown in the following.
  
  ```csharp
  xObj.load("http://www.microsoft.com/sample.asp");
  ```

- Create and return XML information in response to a request from the XMLHTTP control.
Getting XML Back from the Client

You can send XML to the server using any standard technique for sending information to the server.

- Post information from a form.
- Post information from a form in a hidden frame.
- Use the XMLHTTP control to send and receive XML information using HTTP requests.

For more information about posting data, see the DHTML documentation.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel...
Saving XML

You can save XML on the server using the save method, as shown in the following sample ASP file.

**SaveXmlString.asp**

```html
<HTML>
<BODY>
<!-- SaveXmlString.asp -->
<% 
    If IsEmpty(Request("strXml")) Then
        Msg = "Please enter a brief XML string, such as '&lt;/root&gt;'."
    Else
        Dim str, xmldoc
        Set xmldoc = CreateObject("Msxml2.DOMDocument.5.0")
        xmldoc.loadXML Request("strXml")
        If xmldoc.parseError.errorCode <> 0 Then
            Msg = "The string you entered was not well-formed XML. " &
            xmldoc.parseError.reason
        Else
            xmldoc.save(Server.MapPath("saved.xml"))
            str = Request("strXml")
            str = Replace(str, Chr(38),"&\#38;"," 1, 1, 1)
            str = Replace(str, Chr(34),""," 1, 1, 1)
            str = Replace(str, Chr(46),"."," 1, 1, 1)
            str = Replace(str, Chr(60),"<"," 1, 1, 1)
            str = Replace(str, Chr(62),">"," 1, 1, 1)
            Msg = "&lt;P&gt;Your XML string:&lt;BR&gt;" & str & 
            "&lt;BR&gt;has been saved to the server as &lt;B&gt;saved.xml&lt;/B
        End If
    End If
%
<FORM METHOD="POST" ACTION="SaveXmlString.asp">
<pre>
<%= Msg %><BR>
<INPUT TYPE="TEXT" NAME="strXml" SIZE=75
VALUE=""% Request("strXml")%"><BR>
<INPUT TYPE="Submit" VALUE="Submit">
</pre>
</FORM>
</BODY>
</HTML>
```
Try It!

1. On your server computer where you are running Internet Information Services (IIS), open Notepad.
2. Copy `SaveXmlString.asp` from above. Paste it into the Notepad window.
3. From the File menu, click Save As. Save the file as `SaveXmlString.asp` to a folder (such as C:\test).
4. From the Start menu, click Run. Type "inetmgr" to open the Internet Information Services snap-in.
5. From IIS, navigate to Internet Information Services\ <your_Web_server_computer>\Default Web Site.
6. From the Action menu, point to New, then click Virtual directory.
7. Complete the New Virtual Directory wizard using the following information:
   - For Alias, enter the name "test" for your new virtual directory.
   - For Directory, browse and select the path (such as "C:\test") that you used in step 3.
   - For Access Permissions, be sure to select the Write check box as well as accepting the default permissions (Read, Run scripts).
8. Using Internet Explorer, open the Web URL (such as "http://MyServer/test/SaveXmlString.asp") to load the page.
9. Enter a brief well-formed XML string (such as "<root/>") in the form. Click Submit.

   **Note**  The `MapPath` method resolves relative paths into the full paths required on the server.

Output

If you enter the suggested XML string ("<root/>"), you should see the following appear in the browser:

Your XML string:
<root/>
has been saved to the server as `saved.xml`.

You can then open the directory you used in steps 3 and 7 and verify that `saved.xml` is present.
Converting XML to HTML

If you want to maintain XML information on the server and format it as HTML to send to the client, you can use any server-side technique, such as from script in an Active Server Pages (ASP) page. In addition, you can use Extensible Stylesheet Language Transformations (XSLT) to format the information.

If the client is Microsoft® Internet Explorer 5.0 or later, you can send an XML file with an embedded XSLT reference, and the browser will automatically format the sent XML using the referenced style sheet. For more information, see Displaying XML Files in a Browser.

See Also

Serving Out XML Files from an HTTP Server | Using XSLT within ASP
XML and Security

Making XML documents secure requires secure infrastructures for transferring and processing those documents. On the client side, security zones in Microsoft® Internet Explorer 5.0 and later can help you manage the flow of XML information. On the server side, you must create applications that manage security, not just for XML documents, but for all the resources, including external entities, document type definitions (DTDs), and style sheets, referenced by those XML documents.

- XML Client Security
- XML Server Security

This topic suggests best practices to make your MSXML applications more robust, and to reduce their vulnerability to malicious intruders.

Set the resolveExternals property to False when you create new DOM documents.

Be careful when handling file input and output.

Remember that XSLT is code.

Be aware of inherited security contexts from Internet Explorer and other host applications.

Check the length of character input and validate against a permitted range of characters.

Implement parse error handling in your code.

Set the resolveExternals property to False when you create new DOM documents.

When you create a new DOMDocument object, the default value for the resolveExternals property is True. This allows files that contain external definitions to be included and resolved as part of the XML document stream at parse time. For example, the following types of external files and resolvable definitions might be resolved and incorporated into your parsed document:

- Text files that contain resolvable modularized DTD instructions, such as entity or namespace references
- Cascading XSD schema or XSLT style sheet files that contain additional rules or templates

Unless you need or expect this behavior, you should set this property explicitly
to False.

**Note** Setting the `resolveExternals` property to `False` does not prevent your document from being validated upon parsing. This is determined by the value of the `validateOnParse` property.

**Be careful when handling file input and output.**

MSXML provides two DOM methods for working with file input and output:

- To read a file from memory or disk as input, you call the `load` method on a `DOMDocument` object.
- To write a file to disk as output, you call the `save` method on a `DOMDocument` object.

Before you write code that deals with file input and output, you should be familiar with the details of how to design file handling code for the APIs you plan to use with these methods. In particular, you should understand the possibilities for loading or working with `IStream` objects if you reference and use them in your design. Because `IStream` objects can be marshaled to other processes, the data you store with them could potentially be cloned or shared to other applications, with unintended consequences.

For more information about working with the `IStream` interface, see "IStream – Compound File Implementation" in the Platform SDK.

**Remember that XSLT is code.**

XSL Transformations (XSLT) might appear to be a style sheet language, but it is actually a programming language. Therefore, many programs that are typically written in script or in languages such as Visual Basic or C/C++ could potentially be designed and written in XSLT.

To prevent problems, you should test your XSLT files as thoroughly as you would any other script or code module against corrupt or accidental input, such as unanticipated XML document types. Debug as necessary, and design and implement good error handling in your XSLT files. For more information, see the following topics:
Performing Error Handling with XSLT

In particular, safeguard your template designs against the possibility of an infinite recursion loop, in which two templates are written that match and point to each other. The XSLT processor in MSXML does not have a timeout, so when loops occur the application must be manually terminated to stop execution.

Be aware of inherited security contexts from Internet Explorer and other host applications.

MSXML inherits its first level of security from Internet Explorer, or from another immediate host application running under Windows. If that security is not set or in effect, MSXML imposes security based on the source context of the URL provided to locate a file.

For example, the following are three different contexts for loading a sample XML file, books.xml. The first is a local file system, the second is an intranet site, and the third is an Internet site.

C:\temp\books.xml
http://MyWorkgroupServer/books.xml
http://www.example.com/books.xml

For the first URL, MSXML assumes complete trust of the local file system. Access and control of the file are determined solely by the currently configured Windows file security settings, or by the system defaults.

For the second URL, the file is browseable (read-only), because the source is a local Web server on the same local intranet.

For the third URL, the source is an external Web server located using a DNS domain name on the Internet. In this case, MSXML blocks cross-domain interaction. For example, if example.com was the DNS domain requested in the URL, you would not be able to interact with another domain, such as microsoft.com.

For more information about the Internet Explorer security model, see the following topics in Internet Explorer Help:
Check the length of character input and validate against a permitted range of characters.

Many attacks on applications have occurred when string input goes unchecked or a buffer used to store it is overrun. In the worst case, Windows returns an access violation and the application stops responding. In the case of an intentional attack by a malicious user or application attempting to overrun a text input control on an application form, you should know that the MSXML parser fails without an error if more than 32 kilobytes of character or string input is passed to it. However, you might want to implement additional safeguards in your own form validation code for validating user input.

Implement parse error handling in your code.

Many simple applications that can be written using MSXML assume that DOM documents load successfully. For example, consider the following Visual Basic code. This code loads two documents, an XML file and an XSLT style sheet, and then performs a transformation using both files.

```
Begin Sub LoadButDoNotCheck
    Dim xmlDoc As New Msxml2.DOMDocument30
    Dim xslDoc As New Msxml2.DOMDocument30
    xmlDoc.load "books.xml"
    xslDoc.load "stylesheet.xsl"
    MsgBox xmlDoc.transformNode(xslDoc)
End Sub
```

In many cases this code might run without problems. However, it makes two assumptions that might not always be correct:

1. Both the sample XML file (books.xml) and XSLT style sheet (stylesheet.xsl) are assumed to be available at the same path as the executing VBScript (.vbs) file or compiled Visual Basic application (.exe) file that contains this subroutine.
2. Both the XML and XSLT documents are assumed to load successfully as well-formed XML before the call to the `transformNode` method. This method call requires both documents.
If any of these conditions are untrue, the subsequent lines of code fail, but in some instances they are unnecessarily executed anyway. You can rewrite this subroutine as follows, so that it handles errors as they occur:

```vbscript
Begin Sub LoadButCheckAndReportParseErrors
    Dim xmlDoc As New MSXML2.DOMDocument30
    Dim xslDoc As New MSXML2.DOMDocument30
    xmlDoc.Load "books.xml"
    If xmlDoc.parseError.errorCode = 0 Then
        xslDoc.Load "stylesheet.xsl"
        If xslDoc.parseError.reason = "" Then
            MsgBox xmlDoc.transformNode(xslDoc)
        Else
            MsgBox "Stylesheet.xsl did not load. " & xslDoc.parseError.reason
        End If
    Else
        MsgBox "Books.xml did not load. " & xmlDoc.parseError.reason
    End If
End Sub
```

Whenever possible, you should include this kind of parse error handling in code that loads and works with DOMDocument objects. Robust code takes longer to write, but it is easier and more efficient to maintain.
XML Client Security

This topic describes how you can use the Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office and the security zones of Microsoft Internet Explorer 5.0 and later.
Accessing Data Across Domains

The security scheme for MSXML is based on Internet Explorer security zones and settings. The Access data sources across domains setting allows the user three options: Disable, Enable, or Prompt. The following are the four security zones and their defaults for the Access data sources across domains setting.

Internet
  Disable
Local intranet
  Prompt
Trusted
  Enable
Restricted
  Disable

If a user loads a page in the Local intranet zone that uses MSXML to access data from another site in the Local intranet zone, the user will be prompted that data is being accessed across domains.
Accessing Data Across Protocols

Access is denied if either of the following cross-protocol access is attempted:

- https to http access
- http to https access
Accessing Data Across Zones

Cross-zone access of data is permitted only in cases in which more trusted zones access data from less trusted zones. This allows intranet applications to access data on the intranet and Internet.

The following table describes which zones have access to other zones when the Access data sources across domains setting is set to Enable or Prompt.

<table>
<thead>
<tr>
<th>From Zone</th>
<th>Local</th>
<th>Internet</th>
<th>Local Intranet</th>
<th>Trusted Sites</th>
<th>Restricted Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>Access Denied</td>
<td>Access Granted</td>
<td>Access Denied</td>
<td>Access</td>
<td>Access Denied</td>
</tr>
<tr>
<td>Local</td>
<td>Access</td>
<td>Access Granded</td>
<td>Access Denied</td>
<td>Access</td>
<td>Access Denied</td>
</tr>
<tr>
<td>Trusted Sites</td>
<td>Access</td>
<td>Access</td>
<td>Access</td>
<td>Access</td>
<td>Access</td>
</tr>
<tr>
<td>Restricted Sites</td>
<td>Access</td>
<td>Access Denied</td>
<td>Access Denied</td>
<td>Access</td>
<td>Access Denied</td>
</tr>
</tbody>
</table>

**Note**  Although Local is not one of the four security zones, the parser treats files on the local computer as if they are in their own zone.
XML Server Security

During processing, XML files can rely on XML schemas, document type definitions (DTDs), external entity files, and style sheets. The contents of these supporting files can affect the content and presentation of the XML documents and must receive the same level of security protection as the XML files themselves.

XML files and their supporting files can be managed using conventional permissions and authentication strategies. Information entering or leaving a database through an XML file can be managed with both database security and security on the XML conversion mechanisms. For the most part, these issues are handled in the same way they are for HTML files and databases that store information for distribution over the Web.

In some cases, developers rely on supporting files from sources they do not control. For example, the World Wide Web Consortium (W3C) hosts the DTDs for XHTML, the XML version of HTML. Consortia sites and schema repositories provide many sources of ready-to-use XML document structures and supporting files. Developers should check that the hosting organization is providing adequate security and that the files cannot be modified without permission.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
XML Data Islands

There is an increasing need to be able to embed "islands" of data inside HTML pages. In Microsoft® Internet Explorer 5.0 and later, these data islands can be written in XML.

The following topics describe the syntax used for embedding these data islands within a page, and detail the object model exposed by the browser to enable them to be used. This method of embedding XML in HTML follows the note published by the World Wide Web Consortium (W3C) as the "XML in HTML Meeting Report." The W3C expects to evolve the HTML specification to include the capability of embedding XML in HTML documents.
Embedding an XML Data Island into an HTML Page

An XML data island can be embedded using one of the following methods.

- Using the Dynamic HTML (DHTML) `<xml>` element within the HTML document
- Overloading the HTML `<script>` element

Using the XML Element Within the HTML Document

This syntax is valid for Internet Explorer 5.0.

There are two syntactically correct ways of using the `<xml>` element within the HTML document.

- The XML data can exist inline, surrounded by `<xml></xml>` start and end tags.

  ```xml
  <xml id="XMLID">
  <xmldata>
    <data>TEXT</data>
  </xmldata>
  </xml>
  ```

- The `<xml>` element can have a `src` attribute, the value of which is the URL for an XML data source.

  ```xml
  <xml src="http://localhost/xmlFile.xml"></xml>
  ```

The `<xml>` element is present in the HTML Document Object Model. It is in the DHTML a11 collection and is seen by the browser as just a regular node. The XML data within the `<xml>` element can then be accessed by calling the DHTML `XMLDocument` property on the `<xml>` element.

The `XMLDocument` property returns the root node of the XML within the `<xml>` element or the root node of the XML referenced by the value of the `src` attribute. From this root, the XML data island can be navigated using the XML Document Object Model (DOM). The following function returns the data from the data island with the ID of "XMLID".
function returnXMLData()
{
    return document.all("XMLID").XMLDocument.nodeValue;
}

The <XML> element can also be referenced by ID alone. For example, the following function has the identical functionality as the preceding example.

function returnXMLData()
{
    return XMLID.documentElement.text;
}

Because the XMLDocument property was not used, the `documentElement` property must be called to retrieve the root element of the XML.

**Overloading the HTML <SCRIPT> Element**

This syntax has been deprecated and is intended only for down-level cases.

There are three syntactically correct ways of overloading the HTML <SCRIPT> element.

- The `LANGUAGE` attribute can be given the value "XML".

  `<SCRIPT LANGUAGE="XML">`

- The `TYPE` attribute can be given the value "text/xml".

  `<SCRIPT TYPE="text/xml">`

- As with the <XML> element, a `SRC` attribute can be added, the value of which is the URL for an XML data source.

  `<SCRIPT LANGUAGE="XML" SRC="http://localhost/xmlFile.xml"></SCRIPT>`

The following HTML fragment illustrates how to embed data by overloading the <SCRIPT> element.

```
<SCRIPT ID="XMLID" LANGUAGE="XML">
  <XMLDATA>
    <DATA>TEXT</DATA>
  </XMLDATA>
</SCRIPT>
```

The <SCRIPT> element is present in the HTML page's object model. (It is in the
DHTML `all` collection and is seen by the browser as a regular script node.) The XML data within the `<SCRIPT>` elements can be accessed by calling the `XMLDocument` property on the `<SCRIPT>` object.

The following script accesses the XML data island in the preceding HTML fragment, and returns the name of the root node of the XML data island.

```javascript
function returnIslandRootName()
{
    var islandRoot = document.all.("SCRIPT").XMLDocument;
    return islandRoot.nodeName;
}
```

**Note** A tag that uses the name "XML" cannot be nested within an XML data island.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Binding the XML Data Source Object to Data

The XML Data Source Object (DSO) allows you to bind HTML elements to an XML data set. By doing so, you can display a table of elements and attributes contained in the XML data. The contents of the table will change as the XML data changes.

The following topics show how you can use the XML DSO for data binding.

- Data Islands
- Using the XML Data Source Object
- Additional Data Binding and XML Resources
Data Islands

In Microsoft® Internet Explorer 5.0, you can use `<XML>` element to create a data island. Data islands are XML data referenced or included in an HTML page. The XML data can be included within the HTML or it can be in an external file.

Using the `<XML>` element allows you to avoid using script; if the user has disabled scripting for security reasons, the object tag version will not work because script is required to initialize the XML.
**Inline XML**

Inline XML is enclosed within the `<XML>` and `</XML>` tags, as shown in the following example.

```xml
<XML ID="xmlData">
  <?xml version="1.0" ?>
  <trees>
    <tree><name>Birch</name><height unit="foot">10</height></tree>
    <tree><name>Aspen</name><height unit="foot">4</height></tree>
  </trees>
</XML>
```
References to XML Files

External XML uses the src attribute on the <XML> tag to reference an external file. The src attribute can refer to a local file or specify a URL. By using URLs that point to external servers, data can be integrated from several sources. The following code uses the src attribute with a local file.

<XML ID="xmlData" SRC="xmlData.xml"></XML>

The file xmlData.xml contains the XML. The src attribute can refer to a URL as well, as in the following example.

Reading Data from Within a Data Island

The ID attribute of the <XML> element is used to reference the data island. By using HTML tags that can accept data source tags (binding the HTML to the data), you can format and display the data in the XML data island. The following example displays the XML items tagged with <meeting>.

```
<HTML><HEAD></HEAD><TITLE></TITLE>
<BODY>
<XML ID="xmlMeetings">
<?xml version="1.0" ?>
<meetings>
<meeting>
<date>1/1/99</date>
<time>9:00</time>
<location>104</location>
</meeting>
<meeting>
<date>2/1/99</date>
<time>13:00</time>
<location>34</location>
</meeting>
<meeting>
<date>3/1/99</date>
<time>14:30</time>
<location>20</location>
</meeting>
</meetings>
</XML>
<table datasrc="#xmlMeetings">
<tr>
<td><div datafld="date"></div></td>
<td><div datafld="time"></div></td>
<td><div datafld="location"></div></td>
</tr>
</table>
</BODY>
</HTML>
```

The <table> element uses the datasrc attribute to reference the inline XML. The datasrc refers to the ID attribute used to identify the XML, preceded by a pound sign (#). The TD element cannot be bound to data, so a tag that binds to data must be used. In this case, it is the <DIV> element. The <DIV> element also allows the datafld attribute to refer to the XML element to place in this cell of
the table. In this case, it is datafld="date" for the <date> XML element.

As the XML is read, additional rows are created for each element tagged with the <meeting> tag.
Using the XML Data Source Object

You can either use the `<OBJECT>` element to refer to the XML data source object, or you can use data islands and the `<XML>` element.

The XML data source Microsoft® ActiveX® object can be inserted into an HTML page as follows.

```html
<OBJECT width=0 height=0
classid="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
id="xmlldso">
</OBJECT>
```

This can be used as an XML data provider in conjunction with the data binding features of Microsoft Internet Explorer 5.0 for binding XML data to HTML elements on the page.

To load XML into the data source object, use the Dynamic HTML (DHTML) `XMLDocument` property to get a Document Object Model (DOM), and then call the `load` method as follows:

```html
<SCRIPT for=window event=onload>
    var doc = xmlldso.XMLDocument;
    doc.load("books.xml");
    if (doc.documentNode == null)
    {
        HandleError(doc);
    }
</SCRIPT>
```
**Inline XML**

You can also provide the XML inline inside the `<OBJECT>` element, as shown in the following example.

```xml
<OBJECT width=0 height=0
classid="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
    id="xmldso">
    <favorites>
        <favorite>
            <name>Microsoft</name>
            <url>http://www.microsoft.com</url>
        </favorite>
    </favorites>
</OBJECT>
```

You use script to load the data source object, as follows:

```javascript
<script for=window event=onload>
    var doc = xmldso.XMLDocument;
    doc.loadXML(xmldso.altHtml);
    if (doc.documentNode == null) {
        HandleError(doc);
    }
</script>
```
Events Used with the XML Data Source Object

The XML data source object triggers events when the underlying XML data changes. These events are common among the XML data source object and the other supplied data source objects. For more information about events and the data source object, see "DHTML Event Model Support for Data Binding", in the DHTML documentation.
Viewing and Navigating a Subset of the Data

If you want a table to display a small portion of your XML data set, use the DATAPAGESIZE attribute on your <TABLE> element. The DATAPAGESIZE attribute indicates how many records to display in the table.

To navigate the table, you can use the nextPage, previousPage, firstPage, and lastPage methods to display different pages of the data.

Usually, you can provide buttons to view different pages of the data set with nextPage, previousPage, firstPage, and lastPage. For example, a button to view the next page can be written as follows.

```html
<INPUT TYPE="button" VALUE="Next" ONCLICK="tbl.nextPage();">
```

To specify the table, you can use the following code.

```html
<TABLE DATAPAGESIZE=1 ID=tbl DATASRC=#xmlData>
...Table...
</TABLE>
```

This example sets the table to display one record (DATAPAGESIZE=1), identifies itself as tbl (ID=tbl), and uses a data source called xmlData.

To indicate which table the button refers to, use the ID attribute used with the <TABLE> element.

The ONCLICK attribute can also specify "previousPage", "firstPage", or "lastPage". For example, to create a button to display the first page, use the following.

```html
<INPUT TYPE="button" VALUE="First Page" ONCLICK="tbl.firstPage();">
```
The $Text Data Field

When you bind data using the XML data source object, an automatic field called $Text is created. It contains the items in that record, concatenated. The following example demonstrates the $Text data field.

```html
<HTML><HEAD></HEAD><TITLE></TITLE>
<BODY>
<XML ID="xmlParts">
<?xml version="1.0" ?>
<parts>
<part>
<partnumber>A1000</partnumber>
<description>Flat washer</description>
<quantity>1000</quantity>
</part>
<part>
<partnumber>S2300</partnumber>
<description>Machine screw</description>
<quantity>1000</quantity>
</part>
<part>
<partnumber>M2400</partnumber>
<description>Nail</description>
<quantity>500</quantity>
</part>
</parts>
</XML>
<table datasrc=#xmlParts>
<tr>
<td><div datafld="partnumber"></div></td>
<td><div datafld="$Text"></div></td>
</tr>
</table>
</BODY>
</HTML>
```

In this example, the table will consist of a column of part numbers (where datafld is equal to "partnumber") and a column containing the part number, description, and quantity concatenated (where datafld is equal to "$Text"). For example, the first row of the partnumber column will contain "S2300", while the second row of the $Text column will contain "S2300 Machine screw 1000". The $Text column contains the part number.
Rules for Assigning XML Elements and Attributes to Columns and Rows

The XML data source object follows a procedure for assigning elements and attributes to columns and rows in databound applications. XML is modeled as a tree with one element containing the entire hierarchy. For example, an XML description of a book can contain <chapter> tags, <figure> tags, and <section> tags. A <book> element can contain the subelements of <chapter>, <figure>, and <section>. When the XML data source object assigns rows and columns, the subelements, not the top-level element, are converted.

The XML data source object uses this procedure for converting the subelements.

Each subelement and attribute corresponds to a column in some rowset in the hierarchy.

The name of the column is the same as the name of the subelement or attribute, unless the parent element has an attribute and a subelement with the same name, in which case a "!" is prepended to the subelement's column name.

Each column is either a simple column containing scalar values, usually strings, or a rowset column containing subrowsets.

Columns corresponding to attributes are always simple.

Columns corresponding to subelements are rowset columns if either the subelement has its own subelements and/or attributes, or the subelement's parent has more than one instance of the subelement as a child. Otherwise, the column is simple.

When there are multiple instances of a subelement (under different parents), its column is a rowset column if any of the instances imply a rowset column; its column is simple only if all instances imply a simple column.

All rowsets have an additional column named $Text.

A simpler conversion takes place if you have set the JavaDSOCompatible flag to true. The JavaDSOCompatible flag makes the Internet Explorer 5.0 XML data
source object compatible with the Java data source object supplied with Internet Explorer 4.0. To set the JavaDSOCompatible flag, you can use the <XML> element as follows:

<xml id="xmldata" JavaDSOCompatible=true>
  ...XML data
</xml>

Or, you can use the following with the <OBJECT> element (using the <XML> element is recommended).

<Object width=0 height=0
  classid="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
  id="xmldso">
  <param name="JavaDSOCompatible" value="true">
</OBJECT>

The following method is used for creating rows and columns when JavaDSOCompatible is true.

Any element that contains another element is automatically a rowset.

Elements that contain only text are columns.

Values stored in attributes are ignored.

For more information about the Java XML data source object, see "XML Data Source", in the DHTML documentation.
Using DTDs

If you use a document type definition (DTD) with your XML, the XML data source object uses the following method for converting elements and attributes to rows and columns.

Each subelement and attribute named by the DTD corresponds to a column in some rowset in the hierarchy.

The name of the column is the same as the name of the subelement or attribute, unless the parent element has an attribute and a subelement with the same name, in which case a "!" is prepended to the subelement's column name.

Each column is either a simple column containing scalar values, usually strings, or a rowset column containing subrowsets.

Columns corresponding to attributes are always simple.

Columns corresponding to subelements are rowset columns if either the DTD allows the subelement to have its own subelements and/or attributes, or the DTD allows the subelement's parent to have more than one instance of the subelement as a child. Otherwise the column is simple.

All rowsets have an additional column named $Text.

Content corresponding to the content model "ANY" is not included in the rowset hierarchy.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Additional Data Binding and XML Resources

For more information about data binding, see the following topics in the DHTML documentation.

- Data Binding
- Data Binding Architecture
- Adding a Data Source Object to a Page
- Binding HTML Elements to Data
- DHTML Object Model Support for Data Binding
- DHTML Event Model Support for Data Binding

For tutorials and demos, see the following resources.

- [Use the C++ XML Data Source Object](#)
- [Use the Master/Detail Feature with the C++ XML Data Source Object](#)
How Do I?

The following tutorials demonstrate common Extensible Markup Language (XML) authoring and development tasks.

You will be able to read the lessons in Microsoft® Internet Explorer 3.0, 4.0, and other browsers, but the interactive elements and demos require Internet Explorer 5.0 or later.

- [Author XML Elements](#)
- [Author XML Documents](#)
- [Author XML Data Islands](#)
- [Use the XML Object Model](#)
- [Use XML Namespaces](#)
- [Author XML Schemas](#)
- [Use Data Types Within XML Documents](#)
- [Access Typed XML Values](#)
- [Use the C++ XML Data Source Object](#)
- [Use the Master/Detail Feature with the C++ XML Data Source Object](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Author XML Elements

What is an XML element?

XML is a meta-markup language, a set of rules for creating semantic tags used to describe data. An XML element is made up of a start tag, an end tag, and data in between. The start and end tags describe the data within the tags, which is considered the value of the element. For example, the following XML element is a <director> element with the value Matthew Dunn.

<director>Matthew Dunn</director>

The element name <director> allows you to mark up the value Matthew Dunn semantically, so you can differentiate that particular bit of data from another, similar bit of data. For example, there might be another element with the value Matthew Dunn:

<actor>Matthew Dunn</actor>

Because each element has a different tag name, you can easily tell that one element refers to Matthew Dunn, the director, while the other refers to Matthew Dunn, the actor. If there were no way to mark up the data semantically, having two elements with the same value might cause confusion.

In addition, XML tags are case-sensitive, so the following are each a different element.

<City> <CITY> <city>

Attributes

An element can optionally contain one or more attributes. An attribute is a name-value pair separated by an equal sign (=).

<CITY ZIP="01085">Westfield</CITY>

In this example, ZIP="01085" is an attribute of the <CITY> element. Attributes are used to attach additional, secondary information to an element, usually meta
information. Attributes can also accept default values, while elements cannot. Each attribute of an element can be specified only once, but in any order.
Try it!

In the following text box, type the title of a favorite movie and then click Continue.
Check the syntax

Because XML is a highly structured language, it is important that all XML be well-formed. That is, the XML must have both a start tag and end tag, and must be authored using the proper syntax. In the following box, create an XML element with a start tag, an end tag, and a value on a single line. Click the Well-formed? button to see if your XML is correct.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel...
Author XML Documents

What is an XML document?

A basic XML document is simply an XML element that can, but might not, include nested XML elements.

For example, the XML <books> element is a valid XML document:

```xml
<books>
  <book isbn="0345374827">
    <title>Maeve Ascendant</title>
    <author>Eva Corets</author>
  </book>
</books>
```
Authoring guidelines

There are some things to remember when constructing a basic XML document.

- All elements must have an end tag.
- All elements must be cleanly nested (overlapping elements are not allowed).
- All attribute values must be enclosed in quotation marks.
- Each document must have a unique first element, the root node.
Try it!

In the following text box, create an XML document that contains both 
<element> and <attribute> nodes. Click the **Well-formed?** button to see 
whether your XML document conforms to the XML specification.
Author XML Data Islands

What is an XML data island?

A data island is an XML document that exists within an HTML page. It allows you to script against the XML document without having to load it through script or through the `<OBJECT>` tag. Almost anything that can be in a well-formed XML document can be inside a data island.

The `<XML>` element marks the beginning of the data island, and its `ID` attribute provides a name that you can use to reference the data island.

The XML for a data island can be either inline:

```
<XML ID="XMLID">
  <customer>
    <name>Mark Hanson</name>
    <custID>81422</custID>
  </customer>
</XML>
```

or referenced through a `SRC` attribute on the `<XML>` tag:

```
<XML ID="XMLID" SRC="customer.xml"></XML>
```

You can also use the `<SCRIPT>` tag to create a data island.

```
<SCRIPT LANGUAGE="xml" ID="XMLID">
  <customer>
    <name>Mark Hanson</name>
    <custID>81422</custID>
  </customer>
</SCRIPT>
```
Authoring guidelines

Simply author an XML document, place that XML document within an `<XML>` element, and give that `<XML>` element an ID attribute.
Try it!

In the following text box, type a well-formed XML document.

Type an ID for the data island.

Click the **Insert Data Island** button to display an HTML page with your data island inserted.
Use the XML Object Model

What is the XML object model?

The XML object model is a collection of objects that you use to access and manipulate the data stored in an XML document. The XML document is modeled after a tree, in which each element in the tree is considered a node. Objects with various properties and methods represent the tree and its nodes. Each node contains the actual data in the document.
How do I access the nodes in the tree?

You access nodes in the tree by scripting against their objects. These objects are created by the XML parser when it loads and parses the XML document. You reference the tree, or document object, by its ID value. In the following example, "MyXMLDocument" is the document object's ID value. The document object's properties and methods give you access to the root and child node objects of the tree. The root, or document element, is the top-level node from which its child nodes branch out to form the XML tree. The root node can appear in the document only once.

Run the mouse over the following data island to reveal the code required to access each node. The root node is <class>, and its child node is <student>, which has child nodes of <name> and <GPA>.

```xml
<XML ID="MyXMLDocument">
  <class>
    <student studentID="13429">
      <name>James Smith</name>
      <GPA>3.8</GPA>
    </student>
  </class>
</XML>
```

The following list is a sample of the properties and methods that you use to access nodes in an XML document.

<table>
<thead>
<tr>
<th>Property/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLDocument</td>
<td>Returns a reference to the XML Document Object Model (DOM) exposed by the object.</td>
</tr>
<tr>
<td>documentElement</td>
<td>Returns the document root of the XML document.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Returns a node list containing the children of a node (if any).</td>
</tr>
<tr>
<td>item</td>
<td>Accesses individual nodes within the list through an index. Index values are zero-based, so <code>item(0)</code> returns the first child node.</td>
</tr>
<tr>
<td>text</td>
<td>Returns the text content of the node.</td>
</tr>
</tbody>
</table>
The following code shows an HTML page containing an XML data island. The data island is contained within the <XML> element.

```html
<HTML>
  <HEAD>
    <TITLE>HTML with XML Data Island</TITLE>
  </HEAD>
  <BODY>
    <P>Within this document is an XML data island.</P>
    <XML ID="resortXML">
      <resorts>
        <resort>Adventure Works</resort>
        <resort>Alpine Ski House</resort>
      </resorts>
    </XML>
  </BODY>
</HTML>

Adventure Works Alpine Ski House

You access the data island through the ID value, "resortXML", which becomes the name of the document object. In the preceding example, the root node is <resorts>, and the child nodes are <resort>.

The following code accesses the second child node of <resorts> and returns its text, Alpine Ski House.

resortXML.XMLDocument.documentElement.childNodes.item(1).text
How do I persist XML DOM tree information?

Several methods and interfaces are available for persisting DOM information.

If you are using a script language, the DOMDocument object exposes the load, loadXML, and save methods, and the xml property.

For Microsoft® Visual Basic® and C or C++ programmers, the IXMLDOMDocument interface exposes the same members as the DOMDocument object. The IXMLDOMDocument interface also implements standard COM interfaces such as IPersistStreamInit, IPersistMoniker, and IStream.
Try it!

In the following text box, enter code to access a part of either of the preceding documents. Assume that the first document object is "MyXMLDocument" and the second is "resortXML". Then click the **Access XML** button to reveal the node you have referenced.

For an example, you can cut and paste this sample line of code:

```javascript
resortXML.XMLDocument.documentElement.childNodes.item(1).text
```

| Access XML | James Smith 3.8 |

**See Also**

[DOMDocument](#)
Use XML Namespaces

What is an XML namespace?

An XML namespace is a collection of names that can be used as element or attribute names in an XML document. The namespace qualifies element names uniquely on the Web in order to avoid conflicts between elements with the same name. The namespace is identified by some Uniform Resource Identifier (URI), either a Uniform Resource Locator (URL), or a Uniform Resource Name (URN), but it doesn't matter what, if anything, it points to. URIs are used simply because they are globally unique across the Internet.

Namespaces can be declared either explicitly or by default. With an explicit declaration, you define a shorthand, or prefix, to substitute for the full name of the namespace. You use this prefix to qualify elements belonging to that namespace. Explicit declarations are useful when a node contains elements from different namespaces. A default declaration declares a namespace to be used for all elements within its scope, and a prefix is not used.
How do I declare an explicit namespace?

The following explicit declaration declares `bk` and `money` to be shorthand for the full names of their respective namespaces. The `xmlns` attribute is an XML keyword for a namespace declaration.

```xml
<BK>
  <bk:BOOK xmlns:bk="urn:example.microsoft.com:BookInfo"
           xmlns:money="urn:Finance:Money">
    <bk:TITLE>Creepy Crawlies</bk:TITLE>
    <bk:PRICE money:currency="US Dollar">22.95</bk:PRICE>
  </bk:BOOK>
</BK>
```

All elements beginning with `bk:` or `money:` are considered to be from the namespace `urn:example.microsoft.com:BookInfo` or `urn:Finance:Money`, respectively. Run the mouse over the XML to reveal information about the different elements.
How do I declare a default namespace?

A namespace declared without a prefix becomes the default namespace for the document. All elements and attributes in the document that do not have a prefix will then belong to the default namespace. The following example declares that the `<BOOK>` element and all elements and attributes within it (`<TITLE>`, `<PRICE>`, currency) are from the namespace `urn:example.microsoft.com:BookInfo`.

```xml
<BOOK xmlns="urn:example.microsoft.com:BookInfo">
  <TITLE>Creepy Crawlies</TITLE>
  <PRICE currency="US Dollar">22.95</PRICE>
</BOOK>
```
Try it!

In the following text box, add an explicit namespace declaration to the `<book>` element of the following XML document for the namespace urn:example.microsoft.com:AuthorInfo with a prefix of author. Click the Well-formed? button to see whether your XML document conforms to the XML specification.

```xml
<books>
  <book xmlns:bk="urn:example.microsoft.com:BookInfo" xmlns:money="urn:Finance:Money">
    <title>Creepy Crawlies</title>
    <price money:currency="US Dollar">22.95</price>
  </book>
</books>
```

Now, add an element to the XML above called `<author>`, qualified with the namespace prefix `author`, containing the book's author, Stefan Knorr. Click the Well-formed? button to see whether your XML document conforms to the XML specification.
Author XML Schemas

What is an XML Schema?

An XML Schema is an XML-based syntax for defining how an XML document is marked up. XML Schema is a schema specification recommended by Microsoft and it has many advantages over document type definition (DTD), the initial schema specification for defining an XML model. DTDs have many drawbacks, including the use of non-XML syntax, no support for datatyping, and non-extensibility. For example, DTDs do not allow you to define element content as anything other than another element or a string. For more information about DTDs, see the World Wide Web Consortium (W3C) XML Recommendation. XML Schema improves upon DTDs in several ways, including the use of XML syntax, and support for datatyping and namespaces. For example, an XML Schema allows you to specify an element as an integer, a float, a Boolean, a URL, and so on.

Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office in Microsoft Internet Explorer 5.0 and later can validate an XML document with both a DTD and an XML Schema.
How can I create an XML Schema?

Run the mouse over the following XML document to reveal the schema declarations for each node.

```xml
<class xmlns="x-schema:classSchema.xml">
  <student studentID="13429">
    <name>James Smith</name>
    <GPA>3.8</GPA>
  </student>
</class>
```

You'll notice in the preceding document that the default namespace is `x-schema:classSchema.xml`. This tells the parser to validate the entire document against the schema (x-schema) at the following URL (classSchema.xml).

The following is the entire schema for the preceding document. The schema begins with the `<Schema>` element containing the declaration of the schema namespace and, in this case, the declaration of the datatypes namespace as well. The first, `xmlns="urn:schemas-microsoft-com:xml-data"`, indicates that this XML document is an XML Schema. The second, `xmlns:dt="urn:schemas-microsoft-com:datatypes"`, allows you to type element and attribute content by using the `dt` prefix on the `type` attribute within their `ElementType` and `AttributeType` declarations.

```xml
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <AttributeType name='studentID' dt:type='string' required='yes'/>
  <ElementType name='name' content='textOnly'/>
  <ElementType name='GPA' content='textOnly' dt:type='float'/>
  <ElementType name='student' content='mixed'>
    <attribute type='studentID'/>
    <element type='name'/>
    <element type='GPA'/>
  </ElementType>
  <ElementType name='class' content='eltOnly'>
    <element type='student'/>
  </ElementType>
</Schema>
```

The declaration elements that you use to define elements and attributes are described as follows.
Element          Description
(ElementType)     Assigns a type and conditions to an element, and what, if any, child elements it can contain.
(AttributeName)   Assigns a type and conditions to an attribute.
(attribute)        Declares that a previously defined attribute type can appear within the scope of the named <ElementType> element.
(element)         Declares that a previously defined element type can appear within the scope of the named <ElementType> element.

The content of the schema begins with the <AttributeName> and <ElementType> declarations of the innermost elements.

<AttributeName name='studentID' dt:type='string' required='yes'/>
<ElementType name='name' content='textOnly'/>
<ElementType name='GPA' content='textOnly' dt:type='float'/>

The next <ElementType> declaration is followed by its attribute and child elements. When an element has attributes or child elements, they must be included this way in its <ElementType> declaration. They must also be previously declared in their own <ElementType> or <AttributeName> declaration.

<ElementType name='student' content='mixed'>
  <attribute type='studentID'/>
  <element type='name'/>
  <element type='GPA'/>
</ElementType>

This process is continued throughout the rest of the schema until every element and attribute has been declared.

Unlike DTDs, XML Schemas allow you to have an open content model, allowing you to do such things as type elements and apply default values without necessarily restricting content.

In the following schema, the <GPA> element is typed and has an attribute with a default value, but no other nodes are declared within the <student> element.

<Schema xmlns="urn:schemas-microsoft-com:xml-data"
The preceding schema allows you to validate only the area with which you are concerned. This gives you more control over the level of validation for your document and allows you to use some of the features provided by the schema without having to employ strict validation.
Try it!

Try authoring a schema for the following XML document.

```xml
<order>
  <customer>
    <name>Fidelma McGinn</name>
    <phone_number>425-655-3393</phone_number>
  </customer>
  <item>
    <number>5523918</number>
    <description>shovel</description>
    <price>39.99</price>
  </item>
  <date_of_purchase>1998-10-23</date_of_purchase>
  <date_of_delivery>1998-11-03</date_of_delivery>
</order>
```

After you have completed the schema, run it through the XML Validator.

MSDN® Online Downloads provides a set of XML sample files, including an XML document with an accompanying schema. Download these samples to work with the XML document and the schema. To test the validity of your XML against a schema, you can load the document through the XML Validator or simply view the XML file in the MIMETYPE Viewer.

The following are some considerations.

- `<ElementType>` and `<AttributeType>` declarations must precede `<element>` and `<attribute>` content declarations that refer to these types. For example, in the preceding schema, the `<ElementType>` declaration for the `<GPA>` element must precede the `<ElementType>` declaration for the `<student>` element.
- The default value of the `order` attribute depends on the value of the `content` attribute. When the content is set to "eltOnly", the order defaults to `seq`. When the content is set to "mixed", the order defaults to `many`.

Other Resources

[XML Validator](#)
Access Typed XML Values

What is a typed XML value?

A typed XML value is an XML element value that has been assigned a data type in an XML Schema, a formal definition of an XML document. The XML parser uses the schema to validate the document. The XML Schema information resides either in a schema file or within the XML document itself.
How do I access typed XML values?

It is possible to access typed data through the XML object model. Just as you can retrieve the value of an element by calling the nodeValue property on that element's node, you can retrieve the typed value of an element by calling the nodeTypedValue property on the element itself.

For example, consider the following XML document:

```xml
<?xml version="1.0"?>
<weather xmlns="x-schema:weatherSchema.xml">
    <date>1970-09-30</date>
    <degrees>67.5</degrees>
</weather>
```

where `weatherSchema.xml` is the following file:

```xml
<Schema xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <ElementType name="date" content="textOnly" dt:type="date"/>
    <ElementType name="degrees" content="textOnly" dt:type="float"/>
    <ElementType name="weather" content="eltOnly">
        <element type="date"/>
        <element type="degrees"/>
    </ElementType>
</Schema>
```

If you navigate to the `<degrees>` element
`(xmlDocument.documentElement.childNodes.item(1))`, you can access its typed value by calling nodeTypedValue, as follows:

`xmlDocument.documentElement.childNodes.item(1).nodeTypedValue`
Try it!

In the following text box, enter the code that will access the typed value of the `<date>` element. Assume that `xmlDocument` is the document object. Click the **Check Code** button.

To experience the validation capability of data types, see what happens when you enter a date that is not valid. In the following text box, enter an invalid date using the `yyyy-mm-dd` format. For example, enter "1998-13-02", and then click the **Check Date Validation** button.
Use the C++ XML Data Source Object

What is the C++ XML Data Source Object?

Microsoft® Internet Explorer 5.0 and later is released with a C++ Data Source Object (DSO) that can be used to bind XML to HTML. Microsoft Internet Explorer 4.0 was released with a Java XML DSO that is still supported in Internet Explorer 5.0 and later; however, the new C++ DSO gives you better performance and the ability to bind directly to an XML data island.

The C++ DSO allows you to create XML-driven Web applications in a completely declarative fashion, although it is still possible to write scripts against the XML document object. With the C++ DSO, both the Microsoft ActiveX® Data Objects (ADO) and XML object models are available to you.
How do I bind XML to HTML?

Consider the following XML file.

```xml
<universities>
    <university>
        <name>UCSB</name>
        <location>Santa Barbara, CA</location>
    </university>
    <university>
        <name>University of Texas at Arlington</name>
        <location>Arlington, TX</location>
    </university>
    <university>
        <name>USM</name>
        <location>Hattiesburg, MS</location>
    </university>
    <university>
        <name>Baylor</name>
        <location>Waco, TX</location>
    </university>
</universities>
```

You can bind this XML to a repeating table with the following HTML.

```html
<XML ID=xmlDoc src="universities.xml"></XML>
<TABLE DATASRC="#xmlDoc" BORDER=1>
    <THEAD><TH>NAME</TH><TH>LOCATION</TH></THEAD>
    <TR>
        <TD><SPAN DATAFLD="name"></SPAN></TD>
        <TD><SPAN DATAFLD="location"></SPAN></TD>
    </TR>
</TABLE>
```

Click the button below to view the table created.

UCSB Santa Barbara, CA University of Texas at Arlington Arlington, TX USM Hattiesburg, MS Baylor Waco, TX

Show Table

There is no need for an `<APPLET>` or `<OBJECT>` tag; the XML DSO does all the work for you.
Try it!

For more information, see the XML Data Source Object demo in MSDN® Online Downloads. Download the sources for that demo and see if you can change the display format from a form to a repeated table.

Other Resources

[XML Data Source Object Demo](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Devel
Use the Master/Detail Feature with the C++ XML Data Source Object

What is the master/detail feature?

The new master/detail feature allows you to bind to the current record of a hierarchical record set. This means that you can now bind the child elements of the current record to a distinct table. For example, consider the following XML.

```xml
<orders>
  <order order_number="2233">
    <customer>
      <name>James Smith</name>
      <custID>192883</custID>
    </customer>
    <item>
      <name>Fly Swatter</name>
      <price>9.99</price>
    </item>
  </order>
  <order order_number="2234">
    <customer>
      <name>Marea Angela Castaneda</name>
      <custID>827145</custID>
    </customer>
    <item>
      <name>Fly Paper</name>
      <price>15.99</price>
    </item>
  </order>
  <order order_number="2235">
    <customer>
      <name>Amy Jones</name>
      <custID>998022</custID>
    </customer>
    <item>
      <name>Mosquito Netting</name>
      <price>38.99</price>
    </item>
  </order>
</orders>
```
You can allow your user to navigate through the orders by ID, displaying only the customer and item information for the current order. Your user would not have to view the information for all orders—only for the one in which he or she is interested.
How do I bind to the details?

The key to binding to lower levels in the hierarchy (the details) is to understand the structure of your data. The preceding XML has three elements within the root element, <orders>. Based on the heuristic employed by the XML Data Source Object (DSO), each order will be mapped to a rowset containing an "order_number", "customer", and "item" field. The "order_number" column will contain the value of the order_number attribute. The "customer" and "item" columns will contain pointers to respective "customer" and "item" recordsets. The "customer" recordset will then contain a "name" and "custID" field with the values of those elements within. The "item" recordset will contain a "name" and "price" field with the values of those elements within.

So, with this in mind, note that within the top-level (the "orders") recordset, you can get to the value of the "order_number." You will then allow your user to navigate through the orders by "order_number."

<P>ORDER NUMBER: <SPAN DATASRC="#xmlDoc" DATAFLD="order_number"></SPAN></P>

Now add some buttons to help users move throughout the "orders" recordset.

<INPUT TYPE=BUTTON VALUE="Previous Order" onclick="xmlDoc.recordset.movePrevious()">
<INPUT TYPE=BUTTON VALUE="Next Order" onclick="xmlDoc.recordset.moveNext()"/>

To retrieve the values within the child elements of the current record, create a table and set that table's DATASRC attribute to "#xmlDoc", exactly as was done in the preceding example. However, this time also set its DATAFLD attribute to "customer". This tells the table to bind to data within the "customer" recordset pointed at within the "customer" field of the "orders" recordset.

<TABLE DATASRC="#xmlDoc" DATAFLD="customer" BORDER>
   <THEAD><TH>NAME</TH><TH>ID</TH></THEAD>
   <TR>
      <TD><SPAN DATAFLD="name"></SPAN></TD>
      <TD><SPAN DATAFLD="custID"></SPAN></TD>
   </TR>
</TABLE>

Then do the same for the data within the <item> element.

<TABLE DATASRC="#xmlDoc" DATAFLD="item" BORDER=1>
Now, as the user clicks the buttons and moves to the next and previous records in the recordset, the data in the tables will change to correspond to the current record.

Click the **Show Example** button to view the page created above.
Try it!

See if you can use the preceding XML to create a Web page that will allow a user to navigate through the orders by customer name.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
DOM Developer's Guide

This guide introduces the Microsoft COM implementation of the Document Object Model (DOM). The DOM provides a navigable set of classes that directly reflect the W3C Document Object Model (DOM) Level 1 specification. These classes enable you to construct an XML document in memory. You can then compile and validate your XML documents against a DTD or schema. The DOM Developer's Guide, along with the reference material that accompanies it, describes the Microsoft COM implementation of the DOM standard. The Microsoft DOM implementation provides both Microsoft® JScript®, Visual Basic® and Microsoft Visual C++® interfaces.

The following sections provide more information about creating applications with the DOM.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter Kit</td>
<td>Explains the benefits of the DOM, and offers ideas for building DOM applications.</td>
</tr>
<tr>
<td>Concepts</td>
<td>Introduces some of the fundamental concepts that relate to the DOM.</td>
</tr>
<tr>
<td>How Do I?</td>
<td>Helps you build DOM applications. This section provides tutorials for using the DOM with other XML application programming interfaces, such as the Schema Object Model (SOM) and the Simple API for XML (SAX).</td>
</tr>
</tbody>
</table>

See Also

DOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Starter Kit

This starter kit is designed to provide a quick orientation to help you understand and use the XML Document Object Model (DOM), as implemented by Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office.

- [What is XML DOM?](#)
- [What are the MSXML Helper APIs?](#)
What is XML DOM?

The Document Object Model (DOM) as implemented in MSXML provides a programmatic representation of XML documents, fragments, nodes, or node-sets. It also provides an application programming interface for working with XML data. As an XML representation, it conforms to the W3C DOM specification. As a set of API, XML DOM objects are COM objects that implement interfaces and can be used in XML applications written in programming languages such as C/C++, Visual Basic, VBScript, and JScript.

The following JScript fragments outline the basic process of programming with XML DOM.

- To work with XML data programmatically, you first create an XML DOM object. The following JScript code fragment is an example.
  ```javascript
  var xmlDoc = new ActiveXObject("Msxml2.DOMDocument");
  ```

- Next, you can load XML data from a file into the DOM object, as follows:
  ```javascript
  xmlDoc.load("file.xml");
  ```

- Alternatively, you can load data from an XML stream that originated from another application, or that was created dynamically:
  ```javascript
  strXML = "<a><a1>1</a1><a2>2</a2></a>";
  xmlDoc.loadXML(strXML);
  ```

- To navigate to a node in the XML document, you can specify an XPath expression in the call to one of several methods on the DOM instance, for example:
  ```javascript
  var node = xmlDoc.selectSingleNode("/a2");
  ```

- To insert a new element into a DOM object, you set properties and call methods on the object, and possibly on its child objects. For example, the following code fragment appends an empty <a3> element as a new child of the document element, <a>:
  ```javascript
  xmlDoc.documentElement.appendChild( xmlDoc.createElement("a3") );
  ```
To persist a DOM object, call the `save` method on the object:

```javascript
xmlDoc.save( "new.xml" );
```

To perform XSL Transformations (XSLT) on an XML document, you can create another DOM object to hold the XSLT style sheet, and then call the `transformNode` method on the DOM object for the XML document:

```javascript
var xslt = new ActiveXObject("msxml2.DOMDocument");
xslt.load("transform.xsl");
strXML = xmlDoc.transformNode(xslt);
```

These are just a few simple examples to show you how DOM can be used to work with an XML document. More detailed discussions and demonstrations are provided in the later sections of this guide.

See Also

[Concepts](#) | [How Do I?](#) | [DOM Reference](#)

Other Resources

[W3C Document Object Model (DOM) Level 1 Specification (Second Edition)](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
What are the MSXML Helper APIs?

The MSXML Helper APIs are a set of COM objects that provide support functionality. For example, these APIs:

- Simplify certain DOM tasks, such as managing namespaces
- Enhance XSLT performance by caching compiled style sheets
- Enable direct interaction with a Web server

See Also

Concepts | How Do I? | MSXML Helper APIs

Other Resources

W3C Document Object Model (DOM) Level 1 Specification (Second Edition)
Concepts

The Document Object Model (DOM) provides a standardized way to access and manipulate the information stored in XML documents. DOM application programming interfaces (APIs) serve as a bridge between applications and XML documents.

- Introduction to the DOM
- How the DOM Works
- Working with XML Document Parts
- Persistence and the DOM
- White Space and the DOM
- Enforcing Character Encoding with DOM
Introduction to the DOM

The Document Object Model (DOM) presents an easily processed standardized interpretation of an XML document to applications and scripts. The DOM implementation in Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office allows you to load or create a document; gather errors, if any; access and manipulate the information and structures contained within the document; and save the document back out to an XML file, if necessary.

This introduction contains the following topics:

- **DOM and MSXML**
- **Modeling Documents as Node Trees**
- **Working with the DOM**
- **Additional MSXML Features**

**See Also**

[IXMLDOMNamedNodeMap](#) | [selectNodes Method](#) | [transformNode Method](#) | [XML DOM Enumerated Constants](#) | [Template-Driven Transformations](#) | [Using XPath Expressions to Select Nodes](#) | [Using Namespaces in Documents](#)
DOM and MSXML

The DOM implementation is just one part of the MSXML parser. The following diagram shows the tasks involved in parsing an XML document and presenting the information to an application or script.

For another approach to XML document parsing, the MSXML parser also provides a different application programming interface, Simple API for XML (SAX2).

The DOM approach creates an object tree that is managed by the MSXML parser. This allows developers to take advantage of the logic built into MSXML for managing XML content, rather than having to create their own. Developers who need to build applications for specific purposes may find it more efficient to use SAX2 to read the document and then manage the information in their internal data structures.

For more information about SAX2, see the [SAX2 Developer's Guide](SAX2 Developer's Guide).
Modeling Documents as Node Trees

The DOM provides you with an interface for loading, accessing, manipulating, and serializing XML documents. The DOM provides a representation of a complete XML document stored in memory, providing random access to the contents of the entire document. The DOM allows applications to rely on the logic provided by the MSXML parser to handle XML-based information, using its facilities rather than writing custom code to read and process XML.

When the MSXML parser loads an XML document into a DOM, it reads it from start to finish and creates a logical model of nodes from the structures and content within the XML document. The document itself is considered a single node that contains all of the other nodes, including a node representing the root element, which, in turn, contains all of the element, attribute, and text nodes in the document.

Example

The following XML document has a simple multi-tier structure.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
<!DOCTYPE catalog [  
<!NOTATION XLS PUBLIC "http://www.microsoft.com/office/excel/">  
<!ELEMENT COLLECTION (DATE?, BOOK+) >  
<!ATTLIST COLLECTION xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">  
<!ELEMENT BOOK (TITLE, AUTHOR, PUBLISHER) >  
<!ELEMENT DATE (PCDATA) >  
<!ELEMENT TITLE (PCDATA) >  
<!ELEMENT AUTHOR (PCDATA) >  
<!ELEMENT PUBLISHER (PCDATA) >  
]>  
<!--catalog last updated 2000-11-01-->  
<catalog xmlns="http://www.example.com/catalog/"  
  <book id="bk101">  
    <author>&#71;ambardella, Matthew</author>  
    <title>XML Developer's &#x47;uide</title>  
    <genre>Computer</genre>  
    <price>44.95</price>  
    <publish_date>2000-10-01</publish_date>  
```
After MSXML parsing, the top two levels of the node structure representing this document will look like this.

The topmost node is the document itself, which contains all of the other nodes. Immediately within the document are nodes representing the XML declaration, the style sheet processing instruction, the DOCTYPE declaration, and the root element for the document, in this case, catalog.

The catalog element contains the real content of the document, and its structure is shown below.

This part of the DOM contains element, attribute, text, and CDATA nodes. (The character references and built-in entities are converted to ordinary text by the parser, but the CDATA section has its own node.)
Working with the DOM

The DOM allows applications to work with XML document structures and information as program structures rather than text streams. Applications and scripts can read and manipulate these structures without knowing the details of XML syntax, taking advantage of the facilities built into the DOM API of MSXML.

The DOM uses two key abstractions: a tree-like hierarchy and nodes that represent document content and structures. The hierarchy is composed of these nodes, which may contain or be contained by other nodes. For developers, this means that much of the work of XML processing requires navigating this tree structure to find or modify the information it contains. Working with XML requires thinking of information in terms of nested containers, and making sure that information is put into or retrieved from the right container.

The DOM treats nodes as generic objects, making it possible to create a script that loads a document and then traverses all of the nodes, reporting what it finds in the tree.

The following are exposed by the XML DOM.

- DOMDocument
- IXMLDOMNode
- IXMLDOMNodeList
- IXMLDOMNamedNodeMap

The DOM programming interfaces enable applications to traverse the tree and manipulate its nodes. Each node is defined as a specific node type, according to the XML DOM enumerated constants, which also define valid parent and child nodes for each node type. For most XML documents, the most common node types are element, attribute, and text. Attributes occupy a special place in the model because they are not considered child nodes of a parent, and are treated more like properties of elements. An additional programming interface, the IXMLDOMNamedNodeMap, is provided for attributes.

Examples
This sample Active Server Pages (ASP) script uses the MSXML parser to parse a document into a DOM tree, then move down the tree from the root node and report the kinds of nodes it encounters and their content.

**VBScript**

The first version uses Microsoft Visual Basic® Scripting Edition (VBScript) to load the document and walk the tree. If no form input is provided, it presents the user with a form to gather the URL of an XML document. The user then submits that back to the script, which parses the document and presents a tree.

```vbnet
<%@LANGUAGE=VBScript%>
<html>
<head>
<title>Tree walk test - VBScript</title>
</head><body>
<% function attribute_walk(node) For i=1 to indent Response.Write("&nbsp;") Next For Each attrib In node.attributes Response.Write("|--") Response.Write(attrib.nodeTypeString) Response.Write(:) Response.Write(attrib.name) Response.Write("--") Response.Write(attrib.nodeValue) Response.Write("<br />") Next end function

function tree_walk(node) dim nodeName indent=indent+2

For Each child In node.childNodes
    For i=1 to indent
        Response.Write("&nbsp;")
    Next
    Response.Write("|--")
    Response.Write(child.nodeTypeString)
    Response.Write("--")
Next
</body>
</html>
```
If child.nodeType<3 Then
    Response.Write(child.nodeName)
    Response.Write("<br />")
End If
If (child.nodeType=1) Then
    If (child.attributes.length>0) Then
        indent=indent+2
        attribute_walk(child)
        indent=indent-2
    End If
End If
If (child.hasChildNodes) Then
    tree_walk(child)
Else
    Response.Write(child.text
    Response.Write("<br />")
End If
Next
indent=indent-2
end function
xmlFile=Request.Form("fileURI")
Dim root
Dim xmlDoc
Dim child
Dim indent
indent=0
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmDoc.async = False
xmlDoc.validateOnParse=False
xmlDoc.load(xmlFile)
If xmlDoc.parseError.errorcode = 0 Then
    'Walk from the root to each of its child nodes:
    Response.Write("<pre>")
    tree_walk(xmlDoc)
    Response.Write("</pre>"
Else
%>
<h1>XML Parsing - DOM Tree Walk Demo</h1>
<form id="location" method="post" action="">
<input type="text" name="fileURI" maxlength="255" size="20" id="XMLurl"
<br />
<input type="submit" name="submit" value="submit" />
</form>
<% End If%>
</body></html>

At the bottom, the script contains a main routine that either loads a document
and passes it to the tree walker or presents a form asking which document to
load. This script relies on the tree_walk function, a recursive function that
moves from node to node in the tree and presents a suitably formatted version of
the contents. That function in turn relies on an attribute_walk function to
present attribute content, because attribute nodes are not considered children of
element nodes within the DOM.
JScript
The Microsoft JScript® version is similar to the VBScript version. However, it
requires extra lines of code to avoid overwriting variables during the recursive
tree walking.
<%@LANGUAGE=JScript%>
<html>
<head>
<title>Tree walk test - JScript</title>
</head><body>
<%
function attribute_walk(node) {
for (k=1; k<indent; k++) {
Response.Write("&nbsp;");
}
for (m=0; m<node.attributes.length; m++){
attrib = node.attributes.item(m);
Response.Write("|--");
Response.Write(attrib.nodeTypeString);
Response.Write(":");
Response.Write(attrib.name);
Response.Write("--");
Response.Write(attrib.nodeValue);
Response.Write("<br />");
}
} //end attribute_walk
function tree_walk(node) {
indent=indent+2;
for (current=0; current<node.childNodes.length; current++) {
child=node.childNodes.item(current);
for (j=1; j<indent; j++){


Response.Write("&nbsp;");
}
Response.Write("|--");
Response.Write(child.nodeTypeString);
Response.Write("--");
if (child.nodeType<3) {
    Response.Write(child.nodeName);
    Response.Write("<br/>");
}
if (child.nodeType==1) {
    if (child.attributes.length>0) {
        indent=indent+2;
        attribute_walk(child);
        indent=indent-2;
    }
}
if (child.hasChildNodes) {
    //store information so recursion is possible
    depthList[depth]=current;
    depth=depth+1;
    tree_walk(child);
    //return from recursion
    depth=depth-1;
    current=depthList[depth];
}
else {
    Response.Write(child.text);
    Response.Write("<br/>");
}
}
indent=indent-2;

//recursion-tracking variables
depth=0;
depthList=new Array();
indent=0;
xmlFile=new String();
xmlFile=Request.Form("fileURI");
xmlFile=""+xmlFile; //makes string clean for passing to MSXML
xmlPresented=false;

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.validateOnParse=false;
//xmlFile="http://127.0.0.1/ms/local.xml"
if ((xmlFile)) {
    xmlDoc.load(xmlFile);
    if (xmlDoc.parseError.errorcode == null) {
        Response.Write("<pre>");
        tree_walk(xmlDoc);
        Response.Write("</pre>");
        xmlPresented=true;
    }
}
if (xmlPresented==false){
    %>
    <h1>XML Parsing - DOM Tree Walk Demo</h1>
    <form id="location" method="post" action="">
        <input type="text" name="fileURI" maxlength="255" size="20" id="fileURI" />
        <br />
        <input type="submit" name="submit" value="submit" />
    </form>
    %>
</body></html>
Additional MSXML Features


**Fundamental**
The W3C fundamental interfaces include those required to write applications that manipulate XML documents.

**Extended**
The W3C extended interfaces include those that make programming more convenient for developers.

The MSXML DOM implements both fundamental and extended interfaces and also provides additional methods to support XSL Transformations (XSLT), XPath, namespaces, and data types. This approach allows developers to work with a single consistent API for document processing and transformations.

For example, the `selectNodes` method supports XPath syntax to enable sophisticated queries for nodes within a particular context or subtree of the overall tree structure. The `transformNode` method supports the use of XSLT to perform transformations.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
How the DOM Works

MSXML reads an XML document and parses its contents into a set of abstract information containers called nodes. These nodes represent the structure and content of the document, allowing applications to read and manipulate the information in the document without explicitly understanding XML syntax. After a document is parsed, its nodes can be explored in any direction; they are not limited to straight-through text file processing.

For script developers, the most important programming object is DOMDocument. The DOMDocument object exposes properties and methods that allow you to navigate, query, and modify the content and structure of an XML document. Each of the following objects exposes methods and properties that enable you to gather information about the instance of the object, manipulate the value and structure of the object, and navigate to other objects within the tree.

For developers using C, C++, or Microsoft Visual Basic®, these objects are exposed as the following COM interfaces.

- IXMLDOMDocument/DOMDocument
- IXMLDOMNode
- IXMLDOMNodeList
- IXMLDOMNamedNodeMap
- IXMLDOMParseError
Reference Documentation

For complete reference material about each object, and the methods and properties exposed by that object, see XML DOM Objects/Interfaces.

For more information about integrating MSXML 5.0 with existing environments using MSXML, see Installing and Registering the MSXML 5.0 SDK and GUID and ProgID Information.

The following topics explain how to use the XML DOM objects and collections. They discuss the methods and properties exposed by each object, and also address DOM persistence.

- Use DOMDocument
- Use IXMLDOMNode
- Use IXMLDOMNodeList
- Use IXMLDOMNodeNamedNodeMap
- Persistence and the DOM
- Character Encoding, XML, and MSXML
Working with XML Document Parts

The Document Object Model (DOM) provides a generic container, the node, that can be used to represent elements, attributes, textual content, comments, processing instructions, entities, CDATA sections, and document fragments. Because each node type has different access methods and content limitations, it is sometimes easier to work with nodes belonging to a particular type.

This section provides information about how to work with the following types of nodes.

- Document
- Elements
- Attributes
- Text Nodes
- CDATA Sections
- Processing Instructions
- Comments
- Entities
- Entity References
- Document Fragments
- Document Type
- Namespaces
- Notations

See Also

Attributes | CDATA Sections | Comments | Document Map | DOCTYPE Declaration | Elements | Character and Entity References | IXMLDOMNodeNamedNodeMap | Processing Instructions | XML Declaration
Working with the Document

The document map is represented by the `DOMDocument` object, which contains all of the information about the document. The `DOMDocument` object acts as the root node for all of the other nodes in the DOM tree. It contains the root element of the document, as well as information from before the root element—the prolog—and after the end of the root element.

The `DOMDocument` node can contain multiple `IXMLDOMProcessingInstruction` and `IXMLDOMComment` nodes, and one each of the `IXMLDOMElem`ent and `IXMLDOMDocumentType` nodes. These nodes are treated as children, and will appear in the same sequence in which they appeared in the original XML document.

Examples

To create a new document object, you must create a new instance of `Msxml2.DOMDocument`.

**JScript**

The following code is in Microsoft® JScript®.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
```

**VBScript**

The following code is in Microsoft Visual Basic® Scripting Edition (VBScript).

```vbscript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
```

After you create the `DOMDocument` object, you can set flags on it for different kinds of parsing and processing behavior, load XML documents, create new nodes inside the document to build an XML document from within your program, or persist your `DOMDocument` object as an XML file.

For more information about properties and methods of `DOMDocument` objects, see [IXMLDOMDocument/DOMDocument](#).
Working with Elements

Elements within an XML document are represented by IXMLDOMElement objects. Just as elements can contain elements, text, comments, processing instructions, CDATA sections, and entity references within XML documents, these objects can contain IXMLDOMElement, IXMLDOMText, IXMLDOMComment, IXMLDOMProcessingInstruction, IXMLDOMCDATASection, and IXMLDOMEntityReference objects.

Elements do not store attributes directly as children, however. Attributes can be retrieved or modified by name using the getAttribute and setAttribute methods, or manipulated as an IXMLDOMNamedNodeMap through the attributes property.

Creating an element requires that you already have created a DOMDocument object, as the createElement method belongs to DOMDocument. Although you can create and manipulate elements without adding them to the node list of DOMDocument, these elements will disappear as soon as the program completes and will not be persisted in XML documents. Generally, you'll want to assign elements a place in the document tree shortly after creation.

Examples

JScript

The following JScript code creates a DOMDocument object, and then uses that DOMDocument object to create an IXMLDOMElement object, which is then appended to the DOMDocument to be the root element of the document.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var rootElement=xmlDoc.createElement("memo");
xmlDoc.appendChild(rootElement);
```

VBScript

```vbscript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set rootElement=xmlDoc.createElement("memo")
xmlDoc.appendChild(rootElement)
```
For more information about properties and methods of IXMLDOMElement, see IXMLDOMElement.
Working with Attributes

Although attributes belong to a particular element, they are not considered child nodes of element nodes. Instead, they behave more like properties of IXMLDOMNode.

Most of the methods for working with attributes come from IXMLDOMNode. Attributes can be manipulated in the following ways.

- Directly, through the `getAttribute` and `setAttribute` methods of IXMLDOMNode.
- As named IXMLDOMNodeAttribute nodes, with `getAttributeNode` and `setAttributeNode`.
- As a set of nodes accessible through the `attributes` property and returned as an IXMLNamedNodeMap.

Examples

**JScript**

The following JScript example creates a new document containing a `<memo>` element, and then creates an attribute named `author` with a value of "Pat Coleman".

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var rootElement=xmlDoc.createElement("memo");
rootElement.setAttribute("author", "Pat Coleman");
xmlDoc.appendChild(rootElement);
```

**VBScript**

```vbscript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set rootElement=xmlDoc.createElement("memo")
rootElement.setAttribute("author", "Pat Coleman")
xmlDoc.appendChild(rootElement)
```

If you prefer to work with attribute nodes, you can create the attribute, and then create a text node to store its value. Attribute nodes can only contain text nodes and entity reference nodes. (If you need to create an attribute containing an
entity reference, you must use this approach.)

Working with attribute nodes requires using the DOMDocument object to create attribute and text (and entity reference, if necessary) nodes before assigning the nodes to the element.

**JScript**

The following JScript code uses this approach to perform the same work as the preceding examples, creating a `<memo>` element with an `author` attribute holding the value "Pat Coleman".

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var rootElement = xmlDoc.createElement("memo");
var memoAttribute = xmlDoc.createAttribute("author");
var memoAttributeText = xmlDoc.createTextNode("Pat Coleman");
memoAttribute.appendChild(memoAttributeText);
rootElement.setAttributeNode(memoAttribute);
xmlDoc.appendChild(rootElement);
```

**VBScript**

```vbscript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set rootElement = xmlDoc.createElement("memo")
Set memoAttribute = xmlDoc.createAttribute("author")
Set memoAttributeText = xmlDoc.createTextNode("Pat Coleman")
memoAttribute.appendChild(memoAttributeText)
rootElement.setAttributeNode(memoAttribute)
xmlDoc.appendChild(rootElement)
```

Developers who want to work with attributes as a set can also use the IXMLDOMNodeNamedNodeMap collection returned by the attribute property of IXMLDOMElement.

For more information about properties and methods of IXMLDOMAttribute, see [IXMLDOMAttribute](#).
Working with Text Nodes

Text nodes store the character sequences that make up the content of XML elements and attributes. Text nodes cannot have child nodes because they represent content, not structure. Text nodes must be contained by element, attribute, document fragment, or entity reference nodes—they cannot be contained by the top-level document node, though the DOMDocument object is used to create text nodes. For more information, see Textual Content.

Examples

The createTextNode method of DOMDocument object is used to create new text nodes, instances of XMLDOMText.

JScript

In the following JScript example, createTextNode is called twice—once to create a text node for an attribute value, and once to create a text node for element content.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var rootElement=xmlDoc.createElement("memo");
var memoAttribute=xmlDoc.createAttribute("author");
var memoAttributeText=xmlDoc.createTextNode("Pat Coleman");
var toElement=xmlDoc.createElement("to");
var toElementText=xmlDoc.createTextNode("Carole Poland");
memoAttribute.appendChild(memoAttributeText);
xmlDoc.appendChild(rootElement);
rootElement.setAttributeNode(memoAttribute);
rootElement.appendChild(toElement);
toElement.appendChild(toElementText);
```

VBScript

```vbscript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set rootElement=xmlDoc.createElement("memo")
Set memoAttribute=xmlDoc.createAttribute("author")
Set memoAttributeText=xmlDoc.createTextNode("Pat Coleman")
Set toElement=xmlDoc.createElement("to")
Set toElementText=xmlDoc.createTextNode("Carole Poland")
```
memoAttribute.appendChild(memoAttributeText)
xmlDoc.appendChild(rootElement)
rootElement.setAttributeNode(memoAttribute)
rootElement.appendChild(toElement)
toElement.appendChild(toElementText)

Although you can read the textual content of a text node using its nodeValue property, it is more common to use the text property of the IXMLDOMElement object to avoid problems created by elements containing multiple text, CDATA section, and entity reference nodes. The normalize method of IXMLDOMElement can also simplify text processing.

For more information on properties and methods of IXMLDOMText, see IXMLDOMText.
Working with CDATA Sections

CDATA sections allow developers to include the markup characters <, >, and & within element content without using character or entity references. Scripts, style sheets, program code, and sample XML code are frequently contained in CDATA sections. The IXMLDOMCDATASEction object behaves like a text node, but preserves knowledge of its special status, making it easy to preserve CDATA sections through multiple load-and-save cycles.

CDATA sections behave like text nodes, except that they cannot be used inside of attribute nodes. CDATA sections may be mixed with text nodes, entity references, and other content containers, and can appear as child nodes of elements, document fragments, entity references, and entities.

Examples

JScript

The following JScript fragment creates an element named <example> whose contents are protected by a CDATA section using the createCDATASection method of DOMDocument.

```javascript
var demoElement=xmlDoc.createElement("example");
var demoContent=xmlDoc.createCDATASection("<sample>This is an element</sample>");
demoElement.appendChild(demoContent);
```

VBScript

```vbscript
Set demoElement=xmlDoc.createElement("example")
Set demoContent=xmlDoc.createCDATASection("<sample>This is an element</sample>")
demoElement.appendChild(demoContent)
```

CDATA sections also have an impact on MSXML white space handling. For more information, see Preserving Markup Characters by Using CDATA Sections.

For more information about properties and methods of IXMLDOMCDATASEction, see the IXMLDOMCDATASEction.
Working with Processing Instructions

Processing instructions provide a loosely-structured mechanism for conveying application-specific information within a document. Processing instructions can appear within document, element, document fragment, entity reference, or entity nodes. They have only two components—a name (also called the target) and a value. The value may contain content that looks like attribute values (often called pseudo-attributes), but this content is stored as simple text.

Processing instructions can appear in the document content—within the root element or an element that is the descendant of the root element—or they can appear before or after the root element. The `<?xml-stylesheet?>` processing instruction, used to connect XML documents to cascading style sheets or XSL Transformations (XSLT) style sheets, usually appears in the prolog.

Examples

**JScript**

The following JScript example identifies which style sheet to use in a processing instruction in the prolog.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var stylePI = xmlDoc.createProcessingInstruction("xml-stylesheet",
    'type="text/xsl" href="show_book.xsl"');
xmlDoc.appendChild(stylePI);
```

**VBScript**

```vbnet
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set stylePI = xmlDoc.createProcessingInstruction("xml-stylesheet",
    "type="text/xsl" href="show_book.xsl""")
xmlDoc.appendChild(stylePI)
```

The XML document begins with the following.

```xml
<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
```

The target of a processing instruction can be retrieved through its `nodeName`
property while the rest of the processing instruction is stored in the `nodeValue` property. MSXML treats the XML declaration as a processing instruction within its DOM representation. This can have a significant effect on character encoding.

For more information about properties and methods of `XMLDOMProcessingInstruction` objects, see the `IXMLDOMProcessingInstruction`. 
Working with Comments

While processing instructions provide support for unstructured application-oriented information, comments provide a place for unstructured human-readable information. Comments do not have names, only content, and generally represent information that applications should ignore.

Comments are represented by IXMLDOMComment objects, which are created using the createComment method of DOMDocument. Comments can appear before, after, or within the root element.

Examples

**JScript**

The following code creates a comment that says "catalog last updated 2000-11-01".

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var dateComment = xmlDoc.createComment("catalog last updated 2000-11-01");
xmlDoc.appendChild(dateComment);
```

**VBScript**

```vbscript
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set dateComment = xmlDoc.createComment("catalog last updated 2000-11-01")
xmlDoc.appendChild(dateComment)
```

The contents of an IXMLDOMComment object are always stored in its nodeValue property.

For more information about properties and methods of IXMLDOMComment, see [IXMLDOMComment](#).
Working with Entities

The entities attribute is the plural form of the entity attribute. Entities allow a reference to an unparsed external entity to appear within a document. The entities attribute contains a list of such entities, delimited by white space. The entities property of the DOM is used to retrieve the list of entities declared in the DOCTYPE declaration.

For more information, see the entities property and the IXMLDOMDocumentType object/interface.
Working with Entity References

Entity references are used to insert references to entities declared in an XML document type declaration. MSXML will expand all entity references used in a document as it parses them, but applications can insert entity references into document object models intended for export as XML documents.

Examples

JScript

The following JScript fragment creates a header element that contains the text Jay Fluegel & Mindy Martin, expressed as "Jay Fluegel ", the entity reference "amp", and " Mindy Martin".

```javascript
var headerElement=xmlDoc.createElement("header");
var headerStart=xmlDoc.createTextNode("Jay Fluegel ");
var headerMiddle=xmlDoc.createEntityReference("amp");
var headerEnd=xmlDoc.createTextNode(" Mindy Martin");
headerElement.appendChild(headerStart);
headerElement.appendChild(headerMiddle);
headerElement.appendChild(headerEnd);
```

VBScript

```vbscript
Set headerElement=xmlDoc.createElement("header")
Set headerStart=xmlDoc.createTextNode("Jay Fluegel ")
Set headerMiddle=xmlDoc.createEntityReference("amp")
Set headerEnd=xmlDoc.createTextNode(" Mindy Martin")
headerElement.appendChild(headerStart)
headerElement.appendChild(headerMiddle)
headerElement.appendChild(headerEnd)
```

For more information about properties and methods of IXMLDOMEntityReference, see IXMLDOMEntityReference.
Working with Document Fragments

Document fragments are DOM objects that do not correspond precisely to a particular construct in XML 1.0. They represent a portion, not necessarily well-formed, of an XML document. Document fragments provide programmers with a tool for storing nodes like a DOMDocument object with fewer restrictions and overhead.

IXMLDOMDocumentFragment must be created using the createDocumentFragment method of the DOMDocument object. The content stored in the document fragment remains, but the document fragment placeholder disappears when inserted into a document tree.

For more information about properties and methods of XMLDOMDocumentFragment, see IXMLDOMDocumentFragment.
Working with the Document Type

The document type node represents the information provided by the DOCTYPE declaration, if one appears. The IXMLDOMDocumentType object, accessible through the doctype property of the DOMDocument object, is read-only. This object provides access to the entities and notations that have been declared in the document type definition (DTD) of the document. Its name property identifies the root element of the document.

For more information about properties and methods of IXMLDOMDocumentType, see the IXMLDOMDocumentType.
MSXML provides full support for XML namespaces. Because the World Wide Web Consortium (W3C) Namespaces in XML Recommendation appeared after the DOM Level 1 was complete, namespace support in the MSXML DOM is provided through extensions to the core DOM model. Namespace support properties are provided for all nodes in the DOM, but are generally meaningful only for element and attribute nodes. For more information, see Using Namespaces in Documents.

The name property of a node is its qualified name, including the namespace prefix, if any. If you want just the namespace prefix, the prefix property includes that information, while the base name, leaving off the prefix, is available through the baseName property. The namespace URI to which the prefix refers is available through the namespaceURI property.

To create namespace declarations, you must create attributes beginning with xmlns as defined by the Namespaces in XML specification.
**Working with Notations**

Notation declarations associate a name with an identifier for a notation. The notation declarations are not validated. They are referenced when validating strings that are members of NOTATION simple type definitions are called in the XML document.

For more information, see [Notation Declarations](#).
Persistence and the DOM

Information stored in an XML Document Object Model (DOM) can be serialized to an XML file, and information stored in an XML file can be opened into a DOM. XML files provide an easily exchanged, stored, and processed format for persisting information.

For scripting purposes, persistence of object models as files is handled by the following property and methods.

- load method
- loadXML method
- save method
- xml property

For Microsoft® Visual Basic® and C/C++ programmers, DOM persistence is handled by the following methods and objects/interfaces.

- load method
- loadXML method
- save method
- xml property
- IPersistStreamInit interface (see IXMLDOMDocument/DOMDocument)
- IPersistMoniker interface (see IXMLDOMDocument/DOMDocument)
- IStream interface (see IXMLDOMDocument/DOMDocument)
White Space and the DOM

To achieve maximum performance, when a text file is opened with the `xmlDoc.load` or `xmlDoc.loadXML` methods (where `xmlDoc` is an XML DOM document), the parser strips most white space from the file unless specifically directed otherwise; the parser notes within each node whether one or more spaces, tabs, newlines, or carriage returns follow the node in the text by setting a flag.

As a result, a node in an XML DOM document "knows" that at least one occurrence of white space follows or precedes it, but does not know exactly how much white space there was. This method is efficient, reducing both the size of each XML file and the number of calculations required to redisplay the XML in a browser.

However, because this information is lost, an XML document stored in this manner can lose formatting information. Tabs, in particular, can be lost because they are not formally recognized in the default mode as anything but white space.

XSL Transformations (XSLT) uses the XML Document Object Model (DOM), not the source document, to guide its transformation. Because the white space has already been stripped to process the XML into the DOM, white space characters are lost even before the transformation takes place. Most of the XSLT-related methods for specifying white space in the source data document or style sheets are applied too late to make a difference in formatting.

The `preserveWhiteSpace` property tells the XML parser whether or not to convert white space from the initial source file that acts against the XML DOM. If explicitly set (the default is False), it must always be set prior to loading a file; otherwise, the default is to strip white space characters and reduce the file to the smallest possible stream. When `preserveWhiteSpace` is set to True, the XML document retains all of the characters within the file when converted into a DOM.

If you set the `preserveWhiteSpace` property from True to False then back to True for a given DOM document, the spaces will not reappear—setting the
property to False actually removes the space from the DOM, which cannot reconstruct it.

If you are working with XML as a data format streamed to some other process, disable `preserveWhiteSpace` by setting it, or allowing it to default, to False. If retaining positional information is important, for example, in conversions to non-XML formats like tab-separated data, set `preserveWhiteSpace` to True. Be aware this option increases the number of characters and places more demands on the browser.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Enforcing Character Encoding with DOM

In some cases, an XML document is passed to and processed by an application—for example, an ASP page—that cannot properly decode rare or new characters. When this happens, you might be able to work around the problem by relying on DOM to handle the character encoding. This bypasses the incapable application.

For example, the following XML document contains the character entity ("&amp;#8364;") that corresponds to the Euro currency symbol (€). The ASP page, incapable.asp, cannot process currency.xml.

XML Data (currency.xml)

```xml
<?xml version="1.0" encoding="utf-8"?>
<currency>
  <name>Euro</name>
  <symbol>&#8364;</symbol>
  <exchange>
    <base>US$</base>
    <rate>1.106</rate>
  </exchange>
</currency>
```

ASP Page (incapable.asp)

```jsp
<%@language = "javascript"%>
<%
    var doc = new ActiveXObject("Msxml2.DOMDocument");
    doc.async = false;
    if (doc.load(Server.MapPath("currency.xml"))==true) {
        Response.ContentType = "text/xml";
        Response.Write(doc.xml);
    }
%>
```

When limited.asp attempts to process currency.xml, an error results in the Response.Write(doc.xml) instruction. However, you can replace this instruction with the following line:
With this line, the error does not occur. The ASP code produces the correct output in a Web browser, as follows:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<currency>
  <name>Euro</name>
  <symbol>€</symbol>
  <exchange>
    <base>US$</base>
    <rate>1.106</rate>
  </exchange>
</currency>
```

The effect of the change in the ASP page is to let the DOM object (doc)—instead of the Response object on the ASP page—handle the character encoding.
How Do I?

This section provides a series of exercises in programming with XML DOM APIs in C/C++, Visual Basic, and JScript. The following table describes these hands-on tutorials.

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program with DOM in C/C++</strong></td>
<td>A series of task-oriented Win32 console applications written in C/C++. These samples demonstrate how to call the DOM APIs to perform basic XML-related tasks, such as loading and saving a DOM object; performing XLS Transformations (XSLT); creating a DOM object dynamically in an application; querying a node or node-set using XPath expressions; and validating an XML document against a specified XML Schema definition.</td>
</tr>
<tr>
<td><strong>Program with DOM in C/C++ Using Smart Pointer Class Wrappers</strong></td>
<td>A series of task-oriented Win32 console applications in C/C++ that use smart pointer classes to work with interfaces. These samples demonstrate many of the same tasks demonstrated in the C/C++ tutorial. These samples demonstrate how to use DOM APIs to perform common tasks in JScript.</td>
</tr>
</tbody>
</table>
Program with DOM in C/C++

This tutorial is intended for C/C++ developers interested in writing XML applications using the DOM APIs as implemented in Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office. It includes a series of simple examples written in C and tested using Microsoft Visual C++ 6.0. The objective is to provide a quick and practical orientation, so that you can start to write your own XML applications.

These demonstrations are task-oriented, and are simplified for clarity. Therefore, their implementation is not always optimized.

This tutorial consists of the following demonstrations.

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Ready to Work with MSXML (C/C++)</td>
<td>Discusses the requirements for using MSXML, and describes how to install the MSXML components.</td>
</tr>
<tr>
<td></td>
<td>Demonstrates how to set up a Visual C++ project to write your own XML applications in C/C++ using the XML DOM.</td>
</tr>
<tr>
<td></td>
<td>Performs the same functions as the Load an XML DOM Object from an XML File example, but incorporate headers and libraries manually.</td>
</tr>
<tr>
<td>Set Up My Visual C++ Project</td>
<td></td>
</tr>
<tr>
<td>Include Headers and Libraries Manually</td>
<td></td>
</tr>
<tr>
<td>Load an XML DOM Object from an XML File (C/C++)</td>
<td>Demonstrates how to create an XML DOM instance and load its content from an external XML data file.</td>
</tr>
<tr>
<td>Save an XML DOM Object to a File (C/C++)</td>
<td>Demonstrates how to serialize an XML DOM object in a text file.</td>
</tr>
<tr>
<td>Perform XSL Transformations (C/C++)</td>
<td>Demonstrates how to perform XLS Transformations.</td>
</tr>
<tr>
<td>Create an XML DOM Object</td>
<td>Demonstrates how to create an XML DOM object programmatically,</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Dynamically (C/C++)</strong></td>
<td>Including processing instructions, comments, elements, attributes, CDATA sections, and text nodes.</td>
</tr>
<tr>
<td><strong>Query XML DOM Nodes (C/C++)</strong></td>
<td>Demonstrates how to query a DOM node or node-set using XPath expressions.</td>
</tr>
<tr>
<td><strong>Validate an XML Document or Fragment (C/C++)</strong></td>
<td>Demonstrates how to validate an XML document and/or fragment against an XML Schema.</td>
</tr>
<tr>
<td><strong>Manage Object Lifetime Using Reference Counting on Interface Pointers (C/C++)</strong></td>
<td>Demonstrates how to manage the lifetime of DOM objects when using raw interface pointers.</td>
</tr>
</tbody>
</table>
Get Ready to Work with MSXML (C/C++)

To work with MSXML, you must have:

- An appropriate version of MSXML installed on your machine.
- The MSXML libraries and headers incorporated into your project.

The following is a detailed discussion of these requirements.
Installing MSXML Components

To work with MSXML, you must have the module installed on your machine. To verify this, search for the mxmlX.dll libraries in the $sysRoot\system32 directory, where X is the version number and $sysRoot is the system directory. By default, this is c:\windows for machines running Windows XP/ME/98/95, and c:\WinNT for machines running Windows 2000/NT. For example, installing MSXML 5.0 copies the msxml5.dll file (as well as msxml5a.dll and msxml5r.dll) to the system directory, and registers the DLL in the system's registry.

If you are accessing the required DLLs from another local host, instead of downloading them from the Web site, you can install MSXML manually.

To install MSXML manually

1. Copy the DLLs to the system directory on your working machine.
2. Register the DLLs by typing the following at a command prompt:

   cd c:\windows\system32
   regsvr32 msxml5.dll

You can install other versions of MSXML in the same way.
Importing Headers and Libraries

After MSXML is installed, you need to set up your application project so that calls to the MSXML-supported APIs are resolved properly when the application is built. In Microsoft Visual C++, you must import the MSXML headers and libraries into your project.

There are two ways to instruct your application to use the DOM interfaces: You can include headers and libraries automatically, or manually. The first approach below is simple; Visual Studio does all the work for you. The second approach is more traditional.

**To include headers and libraries automatically**

- Add the following two lines to your source code:

  ```
  #import <msxml5.dll> raw_interfaces_only
  using namespace MSXML2;
  ```

**To include headers and libraries manually**

- Follow the instructions in the [Include Headers and Libraries Manually](#) demonstration.
Set Up My Visual C++ Project

For simplicity and clarity, the C/C++ applications presented in this tutorial are constructed as Win32 console application projects in Microsoft Visual Studio. The procedure below outlines the steps to build and run such a project.

To create a new Win32 console application project

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type the name of the project (in this case, "trivialProj") in the Project name field. For the project Location field, either accept the default setting or choose another location (e.g., c:\examples\HowDoI). Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.

Now you are ready to add source, header, and resource files to the project. For a clear illustration, each task in the How Do I Program with DOM in C++ tutorial is implemented as a standalone Win32 console application. The following skeleton code shows the general layout of these applications.

Source Listing for a Trivial Application (trivial.cpp)

```c
#include <stdio.h>
#include <msxm5.dll> raw_interfaces_only
using namespace MSXML2;
int main(int argc, char* argv[])
{
    CoInitialize(NULL);
    printf("start using msxm5.dll... \n");
    // Add your comments here.
    ...CoUninitialize();
    return 0;
}
```

To add source code to your project

1. Select FileView on the project browser, and highlight your project files (in
this case, trivialProj files). From the File menu, select New.

2. On the Files tab of the New dialog box, highlight C++ Source File. Then type the name of the C++ source file (in this case, "trivial.cpp") in the File name text box.

3. Click OK.

4. Copy the source code snippet above and paste it into the file you've just created.

   **Note**  Before using any COM module, you must initialize the COM library by calling the CoInitialize(NULL) function in your application. You then must close the COM library by calling the CoUninitialize() function before the application exits.

To build and run the application

1. To test the application while in the development environment, select Build TrivialProj.exe from the Build menu.

2. When the application is built without errors, run the application by selecting !Execute TrivialProj.exe from the Build menu.

When you build and run trivial.cpp, you should get the following output:

   start using msxml5.dll...

Now we are ready start working with XML DOM. First, we'll demonstrate how to include headers and libraries manually.
Include Headers and Libraries Manually

In this exercise, you will practice how to explicitly include headers and libraries into your project. This is also called loading DOM "raw". This example is functionally identical to the Load an XML DOM Object from an XML File (C/C++) example that follows it. The only difference is the way these headers and libraries are included.

This project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source</strong>: loadDOMRaw.cpp</td>
<td>Creates an XML DOM object and loads the resource file into the project's main directory.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>This is the output you should get when you build and run the loadDOMRaw project.</td>
</tr>
</tbody>
</table>

To create the loadDOMRaw Visual C++ Project

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project LoadDOMRawProj.

To include headers and libraries manually

1. Locate the SDK directory installed by the appropriate MSXML package. For example, your directory might be `C:\msxml5\sdk`, with subdirectories named **inc** and **lib**.
2. Add your directory to the standard include path. To do this in Visual C++, open the Options... panel from the Tools menu. Click the Directories tab, then select Include files from the Show directories for drop-down menu. Add your directory name, such as "C:\msxml5\sdk\inc", to the Directories
Add your directory to the system library path. To do this in Visual C++, open the **Options...** panel from the **Tools** menu. Click the **Directories** tab. Select **Library files** from the **Show directories for** drop-down menu. Add your directory name to the **Directories** textbox.

Include the required MSXML header file by inserting the following macro into the source code:

```c
#include <msxml2.h>
```

Link **msxml2.lib** to your project. To do this in Visual C++, open **Project Settings** by clicking **Settings** from the **Project** menu. Select the **Link** tab. Append "msxml2.lib" to the **Object/library modules:** textbox.

Next, we'll add the source code for this project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
**Source: loadDOMRaw.cpp**

This C/C++ source code performs the following steps:

1. Creates an XML DOM object (pXMLDom) and sets it to synchronous mode.
2. Calls the load method on pXMLDom, specifying the path to *stocks.xml*.

**C/C++ Source File (loadDOMRaw.cpp)**

```c
#include<stdio.h>
#include<msxml2.h>

// You might need to add the msxml5/sdk/(inc, lib) directories
// to the Tools->Options...->Directories in Visual Studio.
//
// You might also need to append "msxml2.lib" to the
// Project->Settings...->Link->Object/Libray Modules field.

int main(int argc, char* argv[]) {
    HRESULT hr;
    IXMLDOMDocument3 *pXMLDoc = NULL;
    IXMLDOMParseError * pObjError = NULL;
    BSTR bstr = NULL;
    VARIANT_BOOL status;
    VARIANT vSrc;

    CoInitialize(NULL);

    hr = CoCreateInstance(CLSID_DOMDocument50,
                           NULL,
                           CLSCTX_INPROC_SERVER,
                           IID_IXMLDOMDocument3,
                           (void**)&pXMLDoc);

    if (FAILED(hr)) {
        printf("Failed to CoCreate an instance of an XML DOM\n");
        printf("Error code: %x\n", hr);
        goto clean;
    }

    hr = pXMLDoc->put_async(VARIANT_FALSE);
    if (FAILED(hr))
```
{  printf("Failed to set async property\n");
  goto clean;
}

hr = pXMLDoc->put_validateOnParse(VARIANT_FALSE);
if (FAILED(hr))
{  printf("Failed to set validateOnParse\n");
  goto clean;
}

hr = pXMLDoc->put_resolveExternals(VARIANT_FALSE);
if (FAILED(hr))
{  printf("Failed to disable resolving externals.\n");
  goto clean;
}

VariantInit(&vSrc);
V_BSTR(&vSrc) = SysAllocString(L"stocks.xml");
V_VT(&vSrc) = VT_BSTR;

hr = pXMLDoc->load(vSrc, &status);
if(status!=VARIANT_TRUE)
{  hr = pXMLDoc->get_parseError(&pObjError);
    hr = pObjError->get_reason(&bstr);
    printf("Failed to load DOM from books.xml. %S\n",bstr);
    goto clean;
}

hr = pXMLDoc->get_xml(&bstr);
printf("stocks.xml:\n%S\n", bstr);

clean:
  if (bstr)
    SysFreeString(bstr);
  if (&vSrc)
    VariantClear(&vSrc);
  if (pObjError)
    pObjError->Release();
  if (pXMLDoc)
    pXMLDoc->Release();
  CoUninitialize();
  return 0;
}
To add loadDOMRaw.cpp to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file loadDOMRaw.cpp.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, we'll add the resource file, stocks.xml.
Resource: stocks.xml

XML Data File (stocks.xml)

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

To add stocks.xml to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file `stocks.xml`
2. Copy the resource listing above and paste it into the source file you just created.

   **Note** You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

Next, build and run the `loadDOMRaw` project. The result should be the output shown in the following topic.
Output

When you build and run the loadDOMRaw project, you should get the following output in a console window.

XML DOM loaded from stocks.xml:
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl"href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
Load an XML DOM Object from an XML File (C/C++)

This project is functionally identical to the previous project, Include Headers and Libraries Manually. The difference is that in this example, the headers and libraries are loaded automatically by Visual Studio.

To load an XML DOM object from an XML data file, you use the load method. The following Microsoft Visual C++ 6.0 project illustrates this procedure.

This project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: loadDOM.cpp</td>
<td>Creates an XML DOM object and loads the resource file into the project's main directory.</td>
</tr>
<tr>
<td>Output</td>
<td>This is the output you should get when you build and run the loadDOM project.</td>
</tr>
</tbody>
</table>

To create the loadDOM Visual C++ Project

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project LoadDOMFromFileProj.

Next, we'll add the source code for the loadDOM project.
This C/C++ source code performs the following steps:

1. Creates an XML DOM object (pXMLDom) and sets it to synchronous mode.
2. Calls the load method on pXMLDom, specifying the path to stocks.xml.

C/C++ Source File (loadDom.cpp)

```c++
#include <stdio.h>
#include <windows.h>
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;

#define HRCALL(a, errmsg) do { 
    hr = (a); 
    if (FAILED(hr)) { 
        dprintf( "%s:%d HRCALL Failed: %s\n 0x%.8x = %s\n", __FILE__, __LINE__, errmsg, hr, #a ); 
        goto clean; 
    }
} while (0)

void dprintf( char * format, ...) 
{ 
    static char buf[1024];
    va_list args;
    va_start( args, format );
    vsprintf( buf, format, args );
    va_end( args);
    OutputDebugStringA( buf);
    printf("%s", buf);
}
```

// Helper function to create a DOM instance:
IXMLDOMDocument * DomFromCOM()
{
    HRESULT hr;
    IXMLDOMDocument *pxmldoc = NULL;
```
HRCALL( CoCreateInstance(__uuidof(DOMDocument50),
    NULL,
    CLSCTX_INPROC_SERVER,
    __uuidof(IXMLDOMDocument),
    (void**)&pxmldoc),
"Create a new DOMDocument");

HRCALL( pxmldoc->put_async(VARIANT_FALSE),
"should never fail");
HRCALL( pxmldoc->put_validateOnParse(VARIANT_FALSE),
"should never fail");
HRCALL( pxmldoc->put_resolveExternals(VARIANT_FALSE),
"should never fail");

return pxmldoc;
clean:
  if (pxmldoc)
    { pxmldoc->Release(); }
  return NULL;
}

int main(int argc, char* argv[])
{
    IXMLDOMDocument *pXMLDom=NULL;
    IXMLDOMParseError *pXMLErr=NULL;
    BSTR bstr = NULL;
    VARIANT_BOOL status;
    VARIANT var;
    HRESULT hr;
    CoInitialize(NULL);
    pXMLDom = DomFromCOM();
    if (!pXMLDom) goto clean;
    VariantInit(&var);
    V_BSTR(&var) = SysAllocString(L"stocks.xml");
    V_VT(&var) = VT_BSTR;
    HRCALL(pXMLDom->load(var, &status), "");
    if (status!=VARIANT_TRUE) {
        HRCALL(pXMLDom->get_parseError(&pXMLErr,"");
        HRCALL(pXMLErr->get_reason(&bstr),");
        dprintf("Failed to load DOM from stocks.xml. %S
",
            bstr);
        goto clean;
    }
HRCALL(pXMLDom->get_xml(&bstr), "");
dprintf("XML DOM loaded from stocks.xml:\n\n", bstr);

clean:
  if (bstr) SysFreeString(bstr);
  if (&var) VariantClear(&var);
  if (pXMLErr) pXMLErr->Release();
  if (pXMLDom) pXMLDom->Release();

  CoUninitialize();
  return 0;
}

To add loadDOM.cpp to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file loadDOM.cpp.
2. Copy the C/C++ source code above, and paste it into the source file you just created.

Next, we'll add the resource file, stocks.xml.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: stocks.xml

XML Data File (stocks.xml)

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <stock exchange="nasdaq">
  <name>new</name>
  <symbol>zzzz</symbol>
  <price dt:dt="number">20.313</price>
 </stock>
 <stock exchange="nyse">
  <name>zacx corp</name>
  <symbol>ZCXM</symbol>
  <price dt:dt="number">28.875</price>
 </stock>
 <stock exchange="nasdaq">
  <name>zaffymat inc</name>
  <symbol>ZFFX</symbol>
  <price dt:dt="number">92.250</price>
 </stock>
 <stock exchange="nasdaq">
  <name>zysmergy inc</name>
  <symbol>ZYSZ</symbol>
  <price dt:dt="number">20.313</price>
 </stock>
</portfolio>
```

To add stocks.xml to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file stocks.xml
2. Copy the resource listing above and paste it into the source file you just created.

   **Note** You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

Next, build and run the loadDOM project. The result should be the output shown in the following topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
When you build and run the **loadDOM** project, you should get the following output in a console window.

```xml
XML DOM loaded from stocks.xml:
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

The next topic in this tutorial demonstrates how to save an XML DOM object to an XML file.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Save an XML DOM Object to a File (C/C++)

Saving an XML DOM object is also known as serializing the DOM. It involves a call to the `save` method on the DOM object. To demonstrate this, we will continue from the previous exercise, and save the DOM object that was loaded from `stocks.xml` to a different file, `myStocks.xml`.

This project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>When you build and run the <code>saveDOM</code> project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

**To create the loadDOM Visual C++ project**

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the project `saveDOMProj`.

Next, we'll add the `source code` for the `saveDOM` project.
This C/C++ source code performs the following steps:

1. Creates an XML DOM object (pXMLDOM) and sets it to synchronous mode.
2. Loads an XML string to pXMLDOM.
3. Calls the loadXML method on pXMLDOM, specifying the XML data as the following string:
   \[<r>
   \t<top/>
   \t<b>bottom</b>
   \n</r>\]
4. Displays the resulting XML DOM in the console window.
5. Calls the save method on pXMLDOM to serialize the DOM content to a file (myData.xml).

### C/C++ Source File (saveDOM.cpp)

```c
#include <stdio.h>
#include <windows.h>
#include <msxml5.dll> raw_interfaces_only
using namespace MSXML2;

// Macro that calls a COM method returning HRESULT value:
#define HRCALL(a, errmsg) \
do { \
    hr = (a); \
    if (FAILED(hr)) { \
        dprintf( "%s:%d HRCALL Failed: %s\n 0x%.8x = %s\n", \
                 __FILE__, __LINE__, errmsg, hr, #a ); \
        goto clean; \
    } \
} \
} while (0)

// Helper function that puts output in stdout and debug window 
// in Visual Studio:
void dprintf( char * format, ... )
{
    static char buf[1024];
    va_list args;
    va_start( args, format );
    vsprintff( buf, format, args );
    va_end( args);
    OutputDebugStringA( buf);
```
printf("%s", buf);

// Helper function to create a DOM instance:
IXMLDOMDocument * DomFromCOM()
{
    HRESULT hr;
    IXMLDOMDocument *pxmldoc = NULL;

    HRCALL( CoCreateInstance(__uuidof(DOMDocument50),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(IXMLDOMDocument),
        (void**)&pxmldoc),
        "Create a new DOMDocument");

    HRCALL( pxmldoc->put_async(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_validateOnParse(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_resolveExternals(VARIANT_FALSE),
        "should never fail");

    return pxmldoc;

    clean:
    if (pxmldoc)
    {
        pxmldoc->Release();
    }
    return NULL;
}

int main(int argc, char* argv[])
{
    IXMLDOMDocument *pXMLDom=NULL;
    IXMLDOMParseError *pXMLErr=NULL;
    BSTR bstr = NULL;
    VARIANT_BOOL status;
    VARIANT var;
    HRESULT hr;

    CoInitialize(NULL);

    pXMLDom = DomFromCOM();
    if (!pXMLDom) goto clean;

    bstr = SysAllocString(L"<r><t>top</t><b>bottom</b></r>"bohydrfastcheautri);
    HRCALL(pXMLDom->loadXML(bstr, &status),
"dom->loadXML(): ");
SysFreeString(bstr);

if (status!=VARIANT_TRUE) {
    HRCALL(pXMLDom->get_parseError(&pXMLErr),
        "dom->get_parseError: ");
    HRCALL(pXMLErr->get_reason(&bstr),
        "parseError->get_reason: ");
    dprintf("Failed to load DOM from xml string. %S\n", bstr);
    goto clean;
}
HRCALL(pXMLDom->get_xml(&bstr), "dom->get_xml: ");
dprintf("XML DOM loaded from stocks.xml:\n\n", bstr);

VariantInit(&var);
V_BSTR(&var) = SysAllocString(L"myData.xml");
V_VT(&var) = VT_BSTR;
HRCALL(pXMLDom->save(var), "dom->save: ");
dprintf("XML DOM saved to myData.xml\n");

clean:
    if (bstr) SysFreeString(bstr);
    if (&var) VariantClear(&var);
    if (pXMLErr) pXMLErr->Release();
    if (pXMLDom) pXMLDom->Release();
    CoUninitialize();
    return 0;
}

To add saveDOM.cpp to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file saveDOM.cpp.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, build and run the saveDOM project. The result should be the output shown in the following topic.
Output

The **saveDOM** project outputs a file, **myData.xml**, in the project’s main directory. In addition to this file, this application produces the following output in a console window.

XML DOM loaded from app:
<r>
    <t>top</t>
    <b>bottom</b>
</r>

XML DOM saved to myData.xml.

The next topic in this tutorial demonstrates how to use DOM to perform XSLT.
Perform XSL Transformations (C/C++)

In general, performing XSL Transformations (XSLT) requires two DOM objects: one for the XML data file, and one for the XSLT style sheet. It also involves calling either the transformNode method or the transformNodeToObject method on the DOM object that corresponds to the XML data. The DOM object holding the XSLT style sheet is the input for the method. The first method returns the result as a string, and the second method returns the result as an object.

In the following C/C++ Win32 Console Application project, we demonstrate how to do this. The XSLT project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: XSLT.cpp</td>
<td>Transforms an XML data file with an XSLT style sheet.</td>
</tr>
<tr>
<td>Resource: stocks.xml and stocks.xsl</td>
<td>An XML data file and an XSLT style sheet.</td>
</tr>
<tr>
<td>Output</td>
<td>When you build and run the XSLT project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

To create the XSLT project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project XSLTProj.

Next, we'll add the source code for the XSLT project.
Source: XSLT.cpp

The source code listed below demonstrates how to call the transformNode method and the transformNodeToObject method in a C/C++ program. Specifically, the code performs the following steps:

1. Loads the XML data file (stocks.xml) into an XML DOM object (pXMLDom).
2. Loads the XSLT style sheet (stocks.xsl) into an XML DOM object (pXSLDoc).
3. Calls the transformNode(pXSLDoc) method on pXMLDom to do the transformation, holds the result in a string, and prints the output to the console.
4. Creates an XML DOM object (pXMLOut) to hold the output of the transformNodeToObject method.
5. Calls the transformNodeToObject method on pXMLDom to do the transformation; holds the resulting object in pXMLOut; prints out the XML result; and serializes the output object in an HTML file, stocks.htm.

C/C++ Source File (XSLT.cpp)

#include <stdio.h>
#include <windows.h>
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;

// Macro that calls a COM method returning HRESULT value:
#define HRCALL(a, errmsg) \
do {
    hr = (a); \
    if (FAILED(hr)) { \
        dprintf( "%s:%d HRCALL Failed: %s
  0x%.8x = %s\\n", __FILE__, __LINE__, errmsg, hr, #a ); \
        goto clean; \
    }
}\
} while (0)

// Helper function that puts output in stdout and debug window
// in Visual Studio:
void dprintf( char * format, ...)
{
    static char buf[1024];
va_list args;
va_start( args, format );
vsprintf( buf, format, args );
va_end( args);
OutputDebugStringA( buf);
printf("%s", buf);
}

// Helper function to create a DOM instance:
IXMLDOMDocument * DomFromCOM()
{
    HRESULT hr;
    IXMLDOMDocument *pxmldoc = NULL;

    HRCALL( CoCreateInstance(__uuidof(DOMDocument50),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(IXMLDOMDocument),
        (void**)&pxmldoc),
        "Create a new DOMDocument");

    HRCALL( pxmldoc->put_async(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_validateOnParse(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_resolveExternals(VARIANT_FALSE),
        "should never fail");

    return pxmldoc;

    clean:
    if (pxmldoc)
    {
        pxmldoc->Release();
    }
    return NULL;
}

int main(int argc, char* argv[])
{
    IXMLDOMDocument *pXMLDom=NULL;
    IXMLDOMDocument *pXSLDoc=NULL;
    IXMLDOMParseError *pXMLErr=NULL;
    IXMLDOMDocument *pXMLOut = NULL;
    IDispatch *pDisp=NULL;
    BSTR bstr = NULL;
    long ecode;
    VARIANT_BOOL status;
    VARIANT var;
    HRESULT hr;
CoInitialize(NULL);

pXMLDom = DomFromCOM();
if (!pXMLDom) goto clean;

VariantInit(&var);
V_BSTR(&var) = SysAllocString(L"stocks.xml");
V_VT(&var) = VT_BSTR;
HRCALL(pXMLDom->load(var,&status),"dom->load(): ");

if (status!=VARIANT_TRUE) {
    HRCALL(pXMLDom->get_parseError(&pXMLErr),
        "dom->get_parseError: ");
    HRCALL(pXMLErr->get_reason(&bstr),
        "parseError->get_reason: ");
    dprintf("Failed to load DOM from stocks.xml. %S\n", bstr);
    goto clean;
}

pXSLDoc = DomFromCOM();
if (!pXSLDoc) goto clean;

VariantClear(&var);
V_BSTR(&var) = SysAllocString(L"stocks.xsl");
V_VT(&var) = VT_BSTR;
HRCALL(pXSLDoc->load(var,&status), "xsl->load(): ");
if (status != VARIANT_TRUE) {
    HRCALL(pXMLDom->get_parseError(&pXMLErr),
        "dom->get_parseError: ");
    HRCALL(pXMLErr->get_reason(&bstr),
        "parseError->get_reason: ");
    dprintf("Failed to load DOM from stocks.xsl. %S\n", bstr);
    goto clean;
}

// Transform dom to a string:
HRCALL(pXMLDom->transformNode(pXSLDoc, &bstr),"" WARRANTIES="");
HRCALL(pXMLDom->get_parseError(&pXMLErr), ");
HRCALL(pXMLDom->transformNode(pXSLDoc, &bstr),"" WARRANTIES="");
HRCALL(pXMLDom->get_parseError(&pXMLErr), ");
if (ecode != 0) {
    if (bstr) SysFreeString(bstr);
    HRCALL(pXMLErr->get_reason(&bstr),"");
    dprintf("Failed to transformNode:\n%S\n", bstr);
    goto clean;
}

dprintf("output from transformNode:\n%S\n", bstr);
// Transform dom to another dom object:
pXMLOut = DomFromCOM();
if (!pXMLOut) goto clean;

HRCALL(pXMLOut->QueryInterface(IID_IDispatch,(void**)&pDisp),
    "QI(IDispatch): ");
VariantClear(&var);
var.vt = VT_DISPATCH;
var.pdispVal = pDisp;

HRCALL(pXMLDom->transformNodeToObject(pXSLDoc, var),
    "transformNodeToObject: ");

if (bstr) SysFreeString(bstr);
HRCALL(pXMLOut->get_xml(&bstr), "");
dprintf("output from transformNodeToObject:\n%S\n", bstr);

VariantClear(&var);
pDisp=NULL;
V_VT(&var) = VT_BSTR;
V_BSTR(&var) = SysAllocString(L"stocks.htm");
HRCALL(pXMLOut->save(var), "XMLOut->save: ");

dprintf("The above output is also saved in stocks.htm.\n");

clean:
    if (bstr) SysFreeString(bstr);
    if (&var) VariantClear(&var);
    if (pXMLErr) { pXMLErr->Release();}
    if (pXMLDom) { pXMLDom->Release();}
    if (pXMLOut) { pXMLOut->Release();}
    if (pXSLDoc) { pXSLDoc->Release();}

    CoUninitialize();
    return 0;
}

**To add XSLT.cpp to the project**

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file **XSLT.cpp**.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, we'll add the **resource files** to the XSLT project.
Resource: stocks.xml and stocks.xsl

XML File (stocks.xml)

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <stock exchange="nasdaq">
        <name>new</name>
        <symbol>zzzz</symbol>
        <price dt:dt="number">20.313</price>
    </stock>
    <stock exchange="nyse">
        <name>zacx corp</name>
        <symbol>ZCXM</symbol>
        <price dt:dt="number">28.875</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zaffymat inc</name>
        <symbol>ZFFX</symbol>
        <price dt:dt="number">92.250</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zysmergy inc</name>
        <symbol>ZYSZ</symbol>
        <price dt:dt="number">20.313</price>
    </stock>
</portfolio>
```

XSLT Style Sheet (stocks.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="/">
        <HTML>
            <BODY>
                <TABLE BORDER="2">
                    <TR>
                        <TD>Symbol</TD>
                        <TD>Name</TD>
                        <TD>Price</TD>
                    </TR>
                    <!-- Use xsl:apply-templates. -->
                    <xsl:apply-templates select="portfolio/stock"/>
                </TABLE>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
To add stocks.xml and stocks.xsl to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file stocks.xml
2. Copy stocks.xml, above, and paste it into the source file you just created.
3. Create another new C++ source file.
4. Copy the XSD file above and paste it into the source file you just created.

Note  You can also copy the files into the project's main directory using Windows Explorer (or a command prompt).

Next, build and run the XSLT project. The result should be the output shown in the following topic.
Output

The XSLT project outputs an HTML file, `stocks.xml`, in the project's main directory. In addition to this file, the application produces the following XSLT-related output in a console window. Note that the formatting of the `transformNode` output is different from that of the `transformNodeToObj`ect output.

```xml
output from transformNode:
<HTML>
<BODY>
<TABLE BORDER="2">
<TR>
<TD>Symbol</TD>
<TD>Name</TD>
<TD>Price</TD>
</TR>
<TR STYLE="color: green" Title="ZCXM is listed on the nyse stock exchange.">
<TD>ZCXM</TD>
<TD>zacx corp</TD>
<TD>28.875</TD>
</TR>
<TR STYLE="color: red" Title="ZFFX is listed on the nasdaq stock exchange.">
<TD>ZFFX*</TD>
<TD>zaffymat inc</TD>
<TD>92.250</TD>
</TR>
<TR STYLE="color: green" Title="ZYSZ is listed on the nasdaq stock exchange.">
<TD>ZYSZ*</TD>
<TD>zysmergy inc</TD>
<TD>20.313</TD>
</TR>
<TR STYLE="color: green" Title="zzzz is listed on the nasdaq stock exchange.">
<TD>zzzz*</TD>
<TD>new</TD>
<TD>20.313</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```
The next topic in this tutorial demonstrates how to create an XML DOM object dynamically.
Create an XML Document Dynamically (C/C++)

We have already demonstrated how to create an XML DOM object from an existing XML data file. In this section, we will show you how to create an XML document dynamically in an application. Specifically, we will demonstrate how to add processing instructions, comments, elements, and text nodes. We will also show how to use extra white space text nodes to provide a structured format when the DOM object is serialized. Such tasks are useful for creating an XML document that is also easy for humans to read.

This tutorial uses of creates the following files:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: dynamDOM.cpp</td>
<td>Creates an XML document dynamically.</td>
</tr>
<tr>
<td></td>
<td>When you build and run the dynamDOM project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

Output

To create the dynamDOM project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project dynamDOMProj.

Next, we'll add the source code for the dynamDOM project.
This application creates a simple, but complete, XML DOM object, with `<root>` as the document element. This element contains three child elements: `<node1>`, `<node2>`, and `<node3>`. The first child element contains character data. The second child element contains a CDATA section. The last child element contains three empty child elements: `<subnode1>`, `<subnode2>`, and `<subnode3>.

Programmatically, the **dynamDOM** application performs the following steps:

1. Creates an XML DOM instance (pXMLDom).
2. Calls the `createProcessInstruction` method on pXMLDom. This creates a processing instruction node (pi) targeted for XML 1.0.
3. Calls the `appendChild` method on pXMLDom. This adds the processing instruction node (pi) to pXMLDom.
4. Calls the `createComment` method on the DOM object (pXMLDom) to create a comment node (pc) and then append it pXMLDom.
5. Creates a `<root>` element as the document element, with a created attribute whose value is set to a string value of "using DOM". Adds this element (`<root>`) to the DOM object (pXMLDom).
6. Creates a `<node1>` element with some character data as its content. Appends this element (pe) to the `<root>` element of the DOM object (pXMLDom).
7. Creates a `<node2>` element that contains a CDATA section (pcd) with markup text. Appends this element (pe) to the `<root>` element of the DOM object (pXMLDom).
8. Creates a `<node3>` element that contains a DOM document fragment (pdf). This fragment contains three other empty child elements: `<subNode1>`, `<subNode2>`, and `<subNode3>`. The code then appends this element (pe) to the `<root>` element of the DOM object (pXMLDom).
9. Saves this dynamically created DOM object to the project's main directory, and prints the XML data in the application console.

**C/C++ Source File (dynamDOM.cpp)**

```c
#include <stdio.h>
#include <windows.h>
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;
```
// Macro that calls a COM method returning HRESULT value:
#define HRCALL(a, errmsg) \
  do { \
    hr = (a); \
    if (FAILED(hr)) { \
      dprintf("%s:%d HRCALL Failed: %s\n 0x%.8x = %s\n", \n        __FILE__, __LINE__, errmsg, hr, #a ); \
        goto clean; \
    } \
  } while (0)

// Helper function that put output in stdout and debug window
// in Visual Studio:
void dprintf( char * format, ... )
{
  static char buf[1024];
  va_list args;
  va_start( args, format );
  vsprintf( buf, format, args );
  va_end( args);
  OutputDebugStringA( buf);
  printf("%s", buf);
}

// Helper function to create a DOM instance.
IXMLDOMDocument * DomFromCOM()
{
  HRESULT hr;
  IXMLDOMDocument *pxmldoc = NULL;

  HRCALL( CoCreateInstance(__uuidof(DOMDocument50),
      NULL,
      CLSCTX_INPROC_SERVER,
      __uuidof(IXMLDOMDocument),
      (void**)&pxmldoc),
      "Create a new DOMDocument");

  HRCALL( pxmldoc->put_async(VARIANT_FALSE),
      "should never fail");
  HRCALL( pxmldoc->put_validateOnParse(VARIANT_FALSE),
      "should never fail");
  HRCALL( pxmldoc->put_resolveExternals(VARIANT_FALSE),
      "should never fail");
  HRCALL( pxmldoc->put_preserveWhiteSpace(VARIANT_TRUE),
      "should never fail");

  return pxmldoc;
  clean:
    if (pxmldoc)
{  
pxmldoc->Release();  
}  
return NULL;
}

VARIANT VariantString(BSTR str)
{
    VARIANT var;  
    VariantInit(&var);  
    V_BSTR(&var) = SysAllocString(str);  
    V_VT(&var) = VT_BSTR;  
    return var;
}

// Helper function to display xml parse error:  
void ReportParseError(IXMLDOMDocument *pDom, char *desc) {
    IXMLDOMParseError *pXMLErr=NULL;  
    BSTR bstrReason = NULL;  
    HRESULT hr;  
    HRCALL(pDom->get_parseError(&pXMLErr),
"dom->get_parseError: ");  
    HRCALL(pXMLErr->get_reason(&bstrReason),
"parseError->get_reason: ");  
    dprintf("%s %S\n",desc, bstrReason);
    clean:  
        if (pXMLErr) pXMLErr->Release();  
        if (bstrReason) SysFreeString(bstrReason);
    }

// Helper function to append a whitespace text node to a  
// specified element:  
void AddWhiteSpaceToNode(IXMLDOMDocument* pDom,
    BSTR bstrWs,
    IXMLDOMNode *pNode)
{
    HRESULT hr;  
    IXMLDOMText *pws=NULL;  
    IXMLDOMNode *pBuf=NULL;  
    HRCALL(pDom->createTextNode(bstrWs,&pws), " ");  
    HRCALL(pNode->appendChild(pws,&pBuf)," ");
    clean:  
        if (pws) pws->Release();  
        pws=NULL;  
        if (pBuf) pBuf->Release();  
        pBuf=NULL;  
}
// Helper function to append a child to a parent node:
void AppendChildToParent(IXMLDOMNode *pChild, IXMLDOMNode *pParent)
{
    HRESULT hr;
    IXMLDOMNode *pNode=NULL;
    HRCALL(pParent->appendChild(pChild, &pNode), "");
clean:
    if (pNode) pNode->Release();
pNode=NULL;
}

int main(int argc, char* argv[])
{
    IXMLDOMDocument *pXMLDom=NULL;
    IXMLDOMProcessingInstruction *pi=NULL;
    IXMLDOMComment *pc=NULL;
    IXMLDOMElement *pe=NULL;
    IXMLDOMElement *pRoot=NULL;
    IXMLDOMAttribute *pa=NULL;
    IXMLDOMAttribute *pa1=NULL;
    IXMLDOMCDATASection *pcd=NULL;
    IXMLDOMElement *peSub=NULL;
    IXMLDOMDocumentFragment *pdf=NULL;

    BSTR bstr = NULL;
    BSTR bstr1 = NULL;
    BSTR bstr_wsn = SysAllocString(L"\n");
    BSTR bstr_wsnt = SysAllocString(L"\n\t");
    BSTR bstr_wsntt = SysAllocString(L"\n\t\t");

    VARIANT var;
    HRESULT hr;

    CoInitialize(NULL);
    VariantInit(&var);

    pXMLDom = DomFromCOM();
    if (!pXMLDom) goto clean;

    // Create a processing instruction element.
    bstr = SysAllocString(L"xml");
    bstr1 = SysAllocString(L"version='1.0'");
    HRCALL(pXMLDom->createProcessingInstruction( bstr, bstr1, &pi),
    "createProcessingInstruction:");
    AppendChildToParent(pi, pXMLDom);
    pi->Release();
    pi=NULL;
    SysFreeString(bstr);
bstr=NULL;
SysFreeString(bstr1);
bstr1=NULL;

// Create a comment element.
bstr = SysAllocString(L"sample xml file created using XML DOM obj 
HRCALL(pXMLDom->createComment(bstr, &pc), "");
AppendChildToParent(pc, pXMLDom);
SysFreeString(bstr);
bstr=NULL;
 pc->Release();
pc=NULL;

// Create the root element.
bstr = SysAllocString(L"root");
HRCALL(pXMLDom->createElement(bstr, &pRoot), "");
SysFreeString(bstr);
bstr = NULL;

// Create a "created" attribute for the <root> element, and 
// assign the "using dom" character data as the attribute value.
bstr = SysAllocString(L"created");
var = VariantString(L"using dom");
HRCALL(pXMLDom->createAttribute(bstr,&pa), "");
HRCALL(pa->put_value(var), "");
HRCALL(pRoot->setAttributeNode(pa, &pa1), "");
AppendChildToParent(pRoot, pXMLDom);
SysFreeString(bstr);
bstr=NULL;
if (pa1) {
   pa1->Release();
   pa1=NULL;
}
pa->Release();
pa=NULL;
VariantClear(&var);

// Next, we will create and add three nodes to the <root> element

// Add NEWLINE+TAB for indentation before <node1>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsnt, pRoot);

// Create a <node1> to hold text content.
bstr = SysAllocString(L"node1");
HRCALL(pXMLDom->createElement(bstr,&pe),"");
SysFreeString(bstr);
bstr=NULL;

bstr=SysAllocString(L"some character data");
HRCALL(pe->put_text(bstr), "");
SysFreeString(bstr);
bstr=NULL;

// Append <node1> to <root>.
AppendChildToParent(pe, pRoot);
pe->Release();
pe=NULL;

// Add NEWLINE+TAB for indentation before <node2>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsnt, pRoot);

// Create a <node2> to hold a CDATA section.
bstr = SysAllocString(L"node2");
HRCALL(pXMLDom->createElement(bstr,&pe,"create <node2> ");
SysFreeString(bstr);
bstr=NULL;

if (!pcd) goto clean;

AppendChildToParent(pcd, pe);
pcd->Release();
pcd=NULL;

// Append <node2> to <root>.
AppendChildToParent(pe, pRoot);
pe->Release();
pe=NULL;

// Add NEWLINE+TAB for indentation before <node3>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsnt, pRoot);

// Create <node3> to hold a doc fragment with three sub-elements.
bstr = SysAllocString(L"node3");
HRCALL(pXMLDom->createElement(bstr,&pe,""));
SysFreeString(bstr);
bstr=NULL;

// Create a document fragment to hold three sub-elements.
HRCALL(pXMLDom->createDocumentFragment(&pdf), "");

// Add NEWLINE+TAB+TAB for indentation before <subnode1>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsntt, pdf);

// Create <subnode1>.
bstr = SysAllocString(L"subnode1");
HRCALL(pXMLDom->createElement(bstr,&peSub), "");
SysFreeString(bstr);
bstr=NULL;

// Add <subnode1> to pdf.
AppendChildToParent(peSub, pdf);
peSub->Release();
peSub=NULL;

// Add NEWLINE+TAB+TAB for indentation before <subnode2>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsntt, pdf);

// Create and append <subnode2>.
bstr = SysAllocString(L"subnode2");
HRCALL(pXMLDom->createElement(bstr,&peSub), "");
SysFreeString(bstr);
bstr=NULL;

// Add <subnode1> to pdf.
AppendChildToParent(peSub, pdf);
peSub->Release();
peSub=NULL;

// Add NEWLINE+TAB+TAB for indentation before <subnode3>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsntt, pdf);

// Create and append <subnode3>.
bstr = SysAllocString(L"subnode3");
HRCALL(pXMLDom->createElement(bstr,&peSub), "");
SysFreeString(bstr);
bstr=NULL;
if (!peSub) goto clean;

// Add <subnode1> to pdf.
AppendChildToParent(peSub, pdf);
peSub->Release();
peSub=NULL;

// Add NEWLINE+TAB after </subnode> in pdf.
AddWhiteSpaceToNode(pXMLDom, bstr_wsnt, pdf);
// Append pdf to <node3> (pe).
AppendChildToParent(pdf, pe);

// Append <node3> to <root>.
AppendChildToParent(pe, pRoot);

// Add NEWLINE for indentation before </root>.
AddWhiteSpaceToNode(pXMLDom, bstr_wsn, pRoot);
HRCALL(pXMLDom->get_xml(&bstr), "dom->get_xml");
dprintf("Dynamically created DOM:\n%S\n", bstr);

VariantClear(&var);
var = VariantString(L"dynamDom.xml");
HRCALL(pXMLDom->save(var), "");
dprintf("DOM saved to dynamDom.xml\n");

clean:
    if (bstr) SysFreeString(bstr);
    if (bstr1) SysFreeString(bstr1);
    if (&var) VariantClear(&var);

    if (pXMLDom) pXMLDom->Release();
    if (pRoot) pRoot->Release();
    if (pe) pe->Release();
    if (peSub) peSub->Release();
    if (pi) pi->Release();
    if (pa) pa->Release();
    if (pa1) pa1->Release();
    if (pc) pc->Release();
    if (pcd) pcd->Release();
    if (pdf) pdf->Release();

    CoUninitialize();
    return 0;
}

**To add dynamDOM.cpp to the project**

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file `dynamDOM.cpp`.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, [build and run](#) the `dynamDOM` project. The result should be the output shown in the following topic.
Output

The **dynamDOM** application outputs an XML data file, **dynamDom.xml**, in the project's main directory. In addition to this file, you should see the following output in a console window.

Dynamically created DOM:
```xml
<?xml version="1.0"?>
<?xml-stylesheet type='text/xml' href='dom.xsl'?>
<!--sample xml file created using XML DOM object.-->
<root created="using dom">
  <node1>some character data</node1>
  <node2><![CDATA[<some mark-up text>]]></node2>
  <node3>
    <subNode1/>
    <subNode2/>
    <subNode3/>
  </node3>
</root>
```

DOM saved to dynamDom.xml

The next topic demonstrates how to query nodes from DOM.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Query XML DOM Nodes (C/C++)

MSXML provides two DOM methods that allow you to query a node or a set of nodes in an XML DOM object: `SelectNodes` and `SelectSingleNode`. You specify the node or nodes of interest in an XPath expression, which is taken as the input to these methods.

In this C/C++ demonstration, we assume that you have some understanding of XPath expressions.

This project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: queryNodes.cpp</td>
<td>Selects both a single node and a node-set from a DOM object.</td>
</tr>
<tr>
<td>Output</td>
<td>When you build and run the <code>queryNodes</code> project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

To create the queryNodes project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project `queryNodesProj`.

Next, we'll add the source code for the `queryNodes` project.

See Also

XPath Developer's Guide | XPath Reference
This example application calls both the `selectSingleNode` method and the `selectNodes` method on an XML DOM object loaded from the `stocks.xml` file. The same XPath expression is input to both methods:

```
//stock[1]/*
```

This expression specifies all the child elements of the first `<stock>` element in the XML document. In MSXML, the `selectSingleNode` method returns the first element of the resultant node-set, and the `selectNodes` method returns all the elements in the node-set.

Programmatically, this source code performs the following steps:

1. Creates an instance of an XML DOM object (`pXMLDom`).
2. Calls the `load` method on `pXMLDom` to load XML data from a file (`stocks.xml`).
3. Calls the `selectSingleNode` method on `pXMLDom` to select a DOM node according to the input XPath expression (`//stock[1]/*`). Then displays the resulting node in the console if the query is successful.
4. Calls the `selectNodes` method on `pXMLDom` to select a DOM node-set according to the input XPath expression (`"//stock[1]/*"`). Then displays the resulting nodes in the console if the query is successful.

### C/C++ Source File (queryNodes.cpp)

```c
#include <stdio.h>
#include <windows.h>
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;

// Macro that calls a COM method returning HRESULT value:
#define HRCALL(a, errmsg) \
  do { \
    hr = (a); \n    if (FAILED(hr)) { \
      dprintf("%s:%d HRCALL Failed: %s\n 0x%.8x = %s\n", __FILE__, __LINE__, errmsg, hr, #a ); \
      goto clean; \
```
while (0)

// Helper function that put output in stdout and debug window
// in Visual Studio:
void dprintf( char * format, ...)
{
    static char buf[1024];
    va_list args;
    va_start( args, format );
    vsprintf( buf, format, args );
    va_end( args);
    OutputDebugStringA( buf);
    printf("%s", buf);
}

// Helper function to create a DOM instance:
IXMLDOMDocument * DomFromCOM()
{
    HRESULT hr;
    IXMLDOMDocument *pxmldoc = NULL;

    HRCALL( CoCreateInstance(__uuidof(DOMDocument50),
                            NULL,
                            CLSCTX_INPROC_SERVER,
                            __uuidof(IXMLDOMDocument),
                            (void**)&pxmldoc),
            "Create a new DOMDocument");

    HRCALL( pxmldoc->put_async(VARIANT_FALSE),
            "should never fail");
    HRCALL( pxmldoc->put_validateOnParse(VARIANT_FALSE),
            "should never fail");
    HRCALL( pxmldoc->put_resolveExternals(VARIANT_FALSE),
            "should never fail");

    return pxmldoc;

    clean:
    if (pxmldoc)
    {
        pxmldoc->Release();
    }
    return NULL;
}

VARIANT VariantString(BSTR str)
{
    VARIANT var;
    VariantInit(&var);
    V_BSTR(&var) = SysAllocString(str);
V_VT(&var) = VT_BSTR;
return var;
}

void ReportParseError(IXMLDOMDocument *pDom, char *desc) {
    IXMLDOMParseError *pXMLErr=NULL;
    BSTR bstrReason = NULL;
    HRESULT hr;
    HRCALL(pDom->get_parseError(&pXMLErr),
        "dom->get_parseError: ");
    HRCALL(pXMLErr->get_reason(&bstrReason),
        "parseError->get_reason: ");

dprintf("%s %S
",desc, bstrReason);
clean:
    if (pXMLErr) pXMLErr->Release();
    if (bstrReason) SysFreeString(bstrReason);
}

int main(int argc, char* argv[]) {
    IXMLDOMDocument *pXMLDom=NULL;
    IXMLDOMNodeList *pNodes=NULL;
    IXMLDOMNode *pNode=NULL;
    BSTR bstr = NULL;
    VARIANT_BOOL status;
    VARIANT var;
    HRESULT hr;
    long length;

    CoInitialize(NULL);

    pXMLDom = DomFromCOM();
    if (!pXMLDom) goto clean;

    VariantInit(&var);
    var = VariantString(L"stocks.xml");
    HRCALL(pXMLDom->load(var, &status), "dom->load(): ");

    if (status!=VARIANT_TRUE) {
        ReportParseError(pXMLDom,
            "Failed to load DOM from stocks.xml");
        goto clean;
    }

    // Query a single node.
    if (bstr) SysFreeString(bstr);
    bstr = SysAllocString(L"//stock[1]/");
    HRCALL(pXMLDom->selectSingleNode(bstr, &pNode),
        "selectSingleNode: ");

    V_VT(&var) = VT_BSTR;
    return var;
}
dom->selectSingleNode: ");
if (!pNode) {
    ReportParseError(pXMLDom, "Calling selectSingleNode ");
} else {
    dprintf("Result from selectSingleNode: 
");
    if (bstr) SysFreeString(bstr);
    HRCALL(pNode->get_nodeName(&bstr), " get_nodeName ");
    dprintf("Node, <%S>:
", bstr);
    if (bstr) SysFreeString(bstr);
    HRCALL(pNode->get_xml(&bstr), " get_xml: ");
    dprintf("\t%S
", bstr);
}

// Query a node-set.
if (bstr) SysFreeString(bstr);
bstr = SysAllocString(L"//stock[1]/*");
HRCALL(pXMLDom->selectNodes(bstr, &pNodes), "selectNodes ");
if (!pNodes) {
    ReportParseError(pXMLDom, "Error while calling selectNodes ");
} else {
    dprintf("Results from selectNodes: 
");
    HRCALL(pNodes->get_length(&length), " get_length: ");
    for (long i=0; i<length; i++) {
        if (pNode) pNode->Release();
        HRCALL(pNodes->get_item(i, &pNode), " get_item: ");
        if (bstr) SysFreeString(bstr);
        HRCALL(pNode->get_nodeName(&bstr), " get_nodeName: ");
        dprintf("Node (%d), <%S>:
", i, bstr);
        SysFreeString(bstr);
        HRCALL(pNode->get_xml(&bstr), " get_xml: ");
        dprintf("\t%S
", bstr);
    }
}

clean:
if (bstr) SysFreeString(bstr);
if (&var) VariantClear(&var);
if (pXMLDom) pXMLDom->Release();
if (pNodes) pNodes->Release();
if (pNode) pNode->Release();

CoUninitialize();
return 0;

To add queryNodes.cpp to the project
1. Select **FileView** on the project browser, and highlight **queryNodeProj files**. From the **File** menu, select **New**.
2. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "queryNodes.cpp" in the **File name** text box, and click **OK**.
3. Copy the C/C++ source code above and paste it into the source file you just created.

Next, we'll add the **resource file** to the **queryNodes** project.
Resource: stocks.xml

XML File (stocks.xml)

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <stock exchange="nasdaq">
        <name>new</name>
        <symbol>zzzz</symbol>
        <price dt:dt="number">20.313</price>
    </stock>
    <stock exchange="nyse">
        <name>zacx corp</name>
        <symbol>ZCXM</symbol>
        <price dt:dt="number">28.875</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zaffymat inc</name>
        <symbol>ZFFX</symbol>
        <price dt:dt="number">92.250</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zysmergy inc</name>
        <symbol>ZYSZ</symbol>
        <price dt:dt="number">20.313</price>
    </stock>
</portfolio>
```

To add stocks.xml to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file queryNodes.
2. Copy stocks.xml, above, and paste it into the source file you just created.

Next, build and run the queryNodes project. The result should be the output shown in the following topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Output

After you build and run the application, you should see the following output displayed in the application's console.

Result from selectSingleNode:
Node, <name>:
    <name>new</name>

Results from selectNodes:
Node (0), <name>:
    <name>new</name>
Node (1), <symbol>:
    <symbol>zzzz</symbol>
Node (2), <price>:
    <price xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="number">20.313</price>

Notice that the output above confirms that the selectSingleNode method picks out only the first node from the node set as specified by the XPath expression (//stock[1]/*). The selectNodes method returns the entire set of nodes specified by this XPath expression.

Next, we'll demonstrate how to validate XML.
Validate an XML Document or Fragment (C/C++)

In MSXML 5.0 and later, there are two ways to validate XML data against a set of XML Schema definitions. One uses the validate method on an XML DOM instance to validate the entire DOM object. The other uses the validateNode(node) method on a DOM instance to validate a DOM fragment. A DOM fragment is a sub-tree rooted at the specified node.

The example in this tutorial illustrates both of these methods. This project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: validateDOM.cpp</td>
<td>Validates an XML file against an XSD schema.</td>
</tr>
</tbody>
</table>

When you build and run the validateDOM project, you should get this output in a console window.

To create the validateDOM project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project validateDOMProj.

Next, we'll add the source code for the validateDOM project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
**Source: validateDOM.cpp**

The source code performs the following basic steps:

1. Creates a DOM instance (pXMLDoc) to hold the XML data.
2. Creates a DOM instance (pXSDDoc) to hold the XML Schema definition.
3. Creates an IXMLSchemaCollection or IXMLSchemaCollection2 object (pSCache). This object is also called a schema cache. The application then adds the XML Schema definition (pXSDDoc) to the pSCache.
4. Associates pSCache with the schemas property of the DOM object for the XML data (pXMLDoc).
5. Calls the following validation methods on the DOM object for XML data (pXMLDoc):
   - Calls the ValidateDocument function. This function then calls the validate method on pXMLDoc to validate the data set as a whole.
   - Calls the ValidateDocumentNodes function. This function then calls the validateNode(pNode) method on pXMLDoc to validate a node object (pNode) selected from pXMLDoc.

Checks the error returned from validate method and/or the validateNode(pNode) method, to determine if the specified XML data set is valid against the given XML Schema definition.

**C/C++ Source File (validateDOM.cpp)**

```c
#include <stdio.h>
#include <windows.h>
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;

// Macro that calls a COM method returning HRESULT value:
#define HRCALL(a, errmsg) \
   do { \
      hr = (a); \
      if (FAILED(hr)) { \
         dprintf( "%s:%d HRCALL Failed: %s\n 0x%.8x = %s\n", __FILE__, __LINE__, errmsg, hr, #a ); \
```

goto clean; \
} \
} while (0)

// Helper function that put output in stdout and debug window 
// in Visual Studio.
void dprintf( char * format, ...)
{
    static char buf[1024];
    va_list args;
    va_start( args, format );
    vsprintf( buf, format, args );
    va_end( args);
    OutputDebugStringA( buf);
    printf("%s", buf);
}

// Helper function to create a DOM instance:
IXMLDOMDocument3 * DomFromCOM()
{
    HRESULT hr;
    IXMLDOMDocument3 *pxmldoc = NULL;

    HRCALL( CoCreateInstance(__uuidof(DOMDocument50),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(IXMLDOMDocument),
        (void**)&pxmldoc),
        "Create a new DOMDocument");

    HRCALL( pxmldoc->put_async(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_validateOnParse(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_resolveExternals(VARIANT_FALSE),
        "should never fail");
    HRCALL( pxmldoc->put_preserveWhiteSpace(VARIANT_TRUE),
        "should never fail");

    return pxmldoc;
    clean:
        if (pxmldoc)
            { 
                pxmldoc->Release();
            }
        return NULL;
    }

// Helper function packaging a BSTR into a variant:
VARIANT VariantString(BSTR str)
{ 
    VARIANT var;
    VariantInit(&var);
    V_BSTR(&var) = SysAllocString(str);
    V_VT(&var) = VT_BSTR;
    return var;
}

// Helper function packaging an object into a variant:
VARIANT VariantObject(IUnknown *pUnk) 
{
    VARIANT var;
    HRESULT hr;
    IDispatch *pDisp=NULL;

    VariantInit(&var);
    HRCALL(pUnk->QueryInterface(IID_IDispatch,(void**)&pDisp),"");
    var.pdispVal = pDisp;
    var.vt = VT_DISPATCH;
    return var;
    clean:
    VariantClear(&var);
    return var;
}

// Helper function to display parse error:
void ReportParseError(IXMLDOMDocument *pDom, char *desc) {
    IXMLDOMParseError *pXMLErr=NULL;
    BSTR bstrReason = NULL;
    HRESULT hr;
    HRCALL(pDom->get_parseError(&pXMLErr),
    "dom->get_parseError: ");
    HRCALL(pXMLErr->get_reason(&bstrReason),
    "parseError->get_reason: ");

    dprintf("%s %S\n",desc, bstrReason);
    clean:
    if (pXMLErr) pXMLErr->Release();
    if (bstrReason) SysFreeString(bstrReason);
}

// Validate XML document as a whole.
void ValidateDocument(IXMLDOMDocument2 *pDom) {

    IXMLDOMParseError *pErr=NULL;
    BSTR bstr = NULL;
    HRESULT hr;
    long eCode;
if (!pDom) {
    dprintf("Can't validate document. Invalid DOM\n");
    return;
}

dprintf("Validating DOM...\n");
HRCALL(pDom->validate(&pErr), "");
HRCALL(pErr->get_errorCode(&eCode), "");
if (eCode != 0) {
    HRCALL(pErr->get_reason(&bstr), "");
    dprintf("XMLDoc is not valid because\n%S\n", bstr);
    SysFreeString(bstr);
    bstr = NULL;
} else {
    HRCALL(pDom->get_xml(&bstr),"" );
    dprintf("XMLDoc is validated: \n%S\n", bstr);
    SysFreeString(bstr);
    bstr = NULL;
}
clean:
    if (pErr) pErr->Release();
    if (bstr) SysFreeString(bstr);
}

// Validate Document nodes, node by node.
void ValidateDocumentNodes(IXMLDOMNodeDocument3 *pDom, BSTR xpath) {
    IXMLDOMNode * pNode = NULL;
    IXMLDOMNodeList *pNodelist = NULL;
    IXMLDOMParseError *pError = NULL;
    BSTR bstr = NULL;
    long length, eCode, i;
    HRESULT hr;

    if (!pDom) {
        dprintf("Can't validate document nodes. Invalid DOM\n");
        return;
    }

    HRCALL(pDom->selectNodes(xpath,&pNodelist),"");
    HRCALL(pNodelist->get_length(&length), "");
    for (i=0; i<length; i++) {
        HRCALL(pNodelist->get_item(i, &pNode), "");
        HRCALL(pDom->validateNode(pNode, &pError),"");
        HRCALL(pError->get_errorCode(&eCode), "");
        HRCALL(pNode->get_nodeName(&bstr), "");
        if (eCode != 0) {
            BSTR bstr1 = NULL;
            HRCALL(pError->get_reason(&bstr1), "");
        }
    }
}
dprintf("\t<%S> (%d) is not valid because\n%S\n",
    bstr, i, bstr1);
SysFreeString(bstr1);
bstr1=NULL;
}
else {
    dprintf("\t<%S> (%d) is a valid node\n",bstr,i);
}
SysFreeString(bstr);
bstr=NULL;
}
clean:
if (bstr) SysFreeString(bstr);
if (pError) pError->Release();
if (pNode) pNode->Release();
if (pNodelist) pNodelist->Release();
return;
}

int main(int argc, char* argv[])
{
    IXMLDOMDocument3  *pXMLDoc = NULL;
    IXMLDOMDocument3  *pXSDDoc = NULL;
    IXMLDOMSchemaCollection *pSCache = NULL;
    BSTR bstr = NULL;
    VARIANT_BOOL status;
    VARIANT var;
    HRESULT hr;

    CoInitialize(NULL);
    VariantInit(&var);

    // Create a DOM and load a document from books.xml
    pXMLDoc = DomFromCOM();
    if (!pXMLDoc) goto clean;

    VariantClear(&var);
    var = VariantString(L"books.xml");
    HRCALL(pXMLDoc->load(var, &status), "");
    if (status!=VARIANT_TRUE) {
        ReportParseError(pXMLDoc,
            "Failed to load DOM from books.xml");
        goto clean;
    }

    // Create a DOM and load a schema from books.xsd.
    pXSDDoc = DomFromCOM();
    if (!pXSDDoc) goto clean;


VariantClear(&var);
var = VariantString(L"books.xsd");
HRCALL(pXSDDoc->load(var, &status), "");
if (status!=VARIANT_TRUE) {
    ReportParseError(pXSDDoc,
                       "Failed to load DOM from books.xsd");
    goto clean;
}

// Create a schema collection object.
HRCALL( CoCreateInstance(__uuidof(XMLSchemaCache50),
                          NULL,
                          CLSCTX_INPROC_SERVER,
                          __uuidof(IXMLDOMSchemaCollection),
                          (void**)&pSCache),
      "Create a new Schema collection object");

// Add schema to the schema collection.
bstr = SysAllocString(L"urn:books");
VariantClear(&var);
var = VariantObject(pXSDDoc);
HRCALL(pSCache->add(bstr,var), "add schema");
SysFreeString(bstr);
bstr = NULL;
VariantClear(&var);

// Attaching the schema to the XML document.
var = VariantObject(pSCache);
HRCALL(pXMLDoc->putref_schemas(var),"");
VariantClear(&var);

// Validate the document as a whole.
ValidateDocument(pXMLDoc);

// Validate all //book nodes, node by node.
bstr = SysAllocString(L"//book");
ValidateDocumentNodes(pXMLDoc, bstr);
SysFreeString(bstr);
bstr = NULL;

// Validate all //book/* nodes, node by node.
bstr = SysAllocString(L"//book/*");
ValidateDocumentNodes(pXMLDoc, bstr);
SysFreeString(bstr);
bstr = NULL;

clean:
    if (bstr) SysFreeString(bstr);
    if (&var) VariantClear(&var);
if (pXMLDoc) pXMLDoc->Release();
if (pXSDDoc) pXSDDoc->Release();
if (pSCache) pSCache->Release();

CoUninitialize();
return 0;
}

To add validateDOM.cpp to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file validateDOM.cpp.
2. Copy the C/C++ source code above, and paste it into the source file you just created.

Next, we'll add the resource files to the validateDOM project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: books.xml and books.xsd

This example uses two resources files. One is an XML data file (books.xml) and the other is the XML schema definition file (books.xsd). According to books.xsd, the second <book> element in books.xml is missing the required <pub_date> child element. Therefore, when we attempt to validate the XML file against the given schema, we should get a validation error.

XML File (books.xml)

```xml
<?xml version="1.0"?>
<books xmlns:x="urn:books">
  <book id="bk001">
    <author>Writer</author>
    <title>The First Book</title>
    <genre>Fiction</genre>
    <price>44.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>An amazing story of nothing.</review>
  </book>

  <book id="bk002">
    <author>Poet</author>
    <title>The Poet's First Poem</title>
    <genre>Poem</genre>
    <price>24.95</price>
    <review>Least poetic poems.</review>
  </book>
</books>
```

XSD File (books.xsd)

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:books"
            xmlns:bks="urn:books">

  <element name="books" type="bks:BooksForm"/>

  <complexType name="BooksForm">
    <sequence>
      <element name="book"
               type="bks:BookForm"
               minOccurs="0"/>
    </sequence>
  </complexType>
</xsd:schema>
```
To add books.xml and books.xsd to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file books.xml
2. Copy books.xml and paste it into the source file you just created.
3. Create another new C++ source file.
4. Copy books.xsd and paste it into the source file you just created.

Note  You can also copy the files into the project’s main directory using Windows Explorer (or a command prompt).

Next, build and run the validateDOM project. The result should be the output shown in the following topic.
After you build and run the project, the application should produce the following output in a console window:

Validating DOM...
    XMLDoc is not valid because
   Content for element 'review' is invalid according to the DTD/Schema. 
   Expecting: pub_date.

    <book> (0) is a valid node
    <book> (1) is not valid because
   Content for element 'review' is invalid according to the DTD/Schema. 
   Expecting: pub_date.

    <author> (0) is a valid node
    <title> (1) is a valid node
    <genre> (2) is a valid node
    <price> (3) is a valid node
    <pub_date> (4) is a valid node
    <review> (5) is a valid node
    <author> (6) is a valid node
    <title> (7) is a valid node
    <genre> (8) is a valid node
    <price> (9) is a valid node
    <review> (10) is a valid node

Next, we'll demonstrate how to manage the lifetime of a DOM object using reference counting.
Manage Object Lifetime Using Reference Counting on Interface Pointers (C/C++)

Like other COM technologies, MSXML uses reference counting to manage the lifetime of a DOM object. A reference count is a number indicating the state of an object. Whenever an object is created or its reference is duplicated, the reference count of the object is incremented. When an object is no longer needed, its reference count is decremented. A positive reference count means the object is usable. When the reference count becomes zero, the object is unusable, and most likely removed from the process memory.

When you use raw COM interface pointers in MSXML, you need to handle reference counting on the interface pointers explicitly. If you do not do this properly, your applications are likely to have memory leaks or to encounter access violations because objects or interface pointers were incorrectly made unusable. Even when you use smart pointer class wrappers that automate memory management, a good understanding of reference counting helps ensure that you use these wrapper classes and templates correctly.

The following are general guidelines for handling reference counting when using MSXML:

- Call the AddRef method on the interface pointer to increment the reference count when its reference is duplicated.
- Call the Release method on the interface pointer to decrement the reference count when the interface pointer is no longer needed, or is to be reused in a different context.

When you create or obtain an object by calling, for example, IXMLDOMDocument::createNode, IXMLDOMNode::appendChild, or CoCreateInstance, the object's reference count is already incremented before the function or method returns. It is your responsibility to release the object when you are done using it. If you duplicate the reference of an existing object, you need to call AddRef explicitly on the new interface pointer. This tutorial
presents a hands-on illustration of these issues.

This project uses or creates the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source: refCount.cpp</strong></td>
<td>Creates an XML document using several DOM objects.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>This is the output you should get when you build and run the <strong>refCount</strong> project.</td>
</tr>
</tbody>
</table>

**To create the refCount Visual C++ Project**

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see **Set Up My Visual C++ Project**. Name the project **LoadDOMRawProj**.

Next, we'll add the **source code** for this project.

**Other Resources**

**The Rules of the Component Object Model**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Source: refCount.cpp

This C/C++ code uses several DOM objects to generate the following simple XML document in memory.

<root>
  <A>
    <a>11</a>
  </A>
  <B>2</B>
</root>

The C/C++ file uses some IXMLDOMNode objects (pElem, pElemA, and pElemOut) as temporary stores for holding elements that are added later to their respective element containers. To help you follow the state changes while objects are created and released, diagnostic messages are displayed to show the reference count of various interface pointers. These diagnostic messages begin with "dmN:", where N is the sequence number of the message.

C/C++ Source File (refCount.cpp)

#include <stdio.h>
#include <windows.h>
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;

// The following macro executes an arbitrary method call, followed
// by error checking. Using it in the main() function below makes
// the code more readable.
#define HRCALL(a, errmsg) \
    do { \
      hr = (a); \
      if (FAILED(hr)) { \
        dprintf( "%s:%d HRCALL Failed: %s\n  0x%.8x = %s\n", \n          __FILE__, __LINE__, errmsg, hr, #a ); \
        goto clean; \
      } \
    } while (0)

// This helper function outputs a message to both stdout and
// debug-output in Visual Studio.
void dprintf( char * format, ...)
{

static char buf[1024];
va_list args;
va_start(args, format);
    //va_arg(args, char *); // Skip format.
vsprintf(buf, format, args);
va_end(args);
OutputDebugStringA(buf);
printf("%s", buf);
}

// Helper function to create a DOM instance:
IXMLDOMDocument * DomFromCOM()
{
    HRESULT hr;
    IXMLDOMDocument *pxmldoc = NULL;

    HRCall( CoCreateInstance(__uuidof(DOMDocument50),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(IXMLDOMDocument),
        (void**)&pxmldoc),
        "Create a new DOMDocument");
    // pxmldoc RefCount = 1

    HRCall( pxmldoc->put_async(VARIANT_FALSE),
        "should never fail");
    HRCall( pxmldoc->put_validateOnParse(VARIANT_FALSE),
        "should never fail");
    HRCall( pxmldoc->put_resolveExternals(VARIANT_FALSE),
        "should never fail");

    return pxmldoc;

    clean:
    if (pxmldoc)
    {
        // pxmldoc RefCount = 0 -> deleted
        pxmldoc->Release();
    }
    return NULL;
}

// Helper function creates an element and returns it as an
// XMLDOMNode.
IXMLDOMNode *ElementFromDom(BSTR name, IXMLDOMDocument *pDom)
{
    if (pDom==NULL) {
        dprintf("ElementFromDom: invalid pDom\n");
        return NULL;
    }
HRESULT hr;
IXMLDOMNode *pNode = NULL;
IXMLDOMElement *pElem = NULL;

// We use createElement() instead of createNode() to illustrate when to call Release on a DOM object.
HRCALL(pDom->createElement(name, &pElem), "can't create pElem");
// pElem RefCount = 1

HRCALL(pElem->QueryInterface(__uuidof(IXMLDOMNode), (void**)&pNode), "can't QI IXMLDOMNode from pElem.");
// pNode RefCount = 1

clean:
if (pElem)
{
  // We are done with pElem, so we release it.
  pElem->Release();
}

// pNode RefCount = 1
return pNode;

宣称: // Helper function to determine the reference count of an interface pointer:
ULONG refCountOf(IUnknown * pUnk)
{
  pUnk->AddRef();
  return pUnk->Release();
}

int output_index = 1;
// Helper function to display refcount of an interface pointer:
void ShowRefCountOf(char * title, IUnknown * pUnk)
{
  fprintf("dm%d: refCount(%s) = %d\n", output_index++, title, refC
}

// Helper function to display the specified refcount:
void ShowRefCount(char * title, ULONG count)
{
  fprintf("dm%d: refCount(%s) = %d\n", output_index++, title, coun
}

/////////////////////////////

int main(int argc, char* argv[])

HRESULT hr;
IXMLDOMDocument *pDom = NULL;
IXMLDOMNode *pRoot = NULL;
IXMLDOMNode *pElem = NULL;
IXMLDOMNode *pElemA = NULL;
IXMLDOMNode *pElemOut = NULL;
BSTR bstr = NULL;
ULONG count;

CoInitialize(NULL);

pDom = DomFromCOM();
if (!pDom)
    goto clean;
ShowRefCountOf("pDom ++", pDom);    // dm1:

// Create an empty <root> element.
bstr = SysAllocString(L"root");
pRoot = ElementFromDom(bstr, pDom);
if (!pRoot) {
    goto clean;
}
ShowRefCountOf("pRoot ++", pRoot);    // dm2:
    SysFreeString( bstr);
bstr = NULL;

// Create an empty <A> element.
bstr = SysAllocString(L"A");
pElem = ElementFromDom(bstr, pDom);
if (!pElem) {
    goto clean;
}
ShowRefCountOf("pElem ++", pElem);    // dm3:
    SysFreeString( bstr);
bstr = NULL;

// Append <A> to <root>.
HRCALL( pRoot->appendChild(pElem, &pElemOut),
    "failed to append <A> to <root>");
ShowRefCountOf("pElemOut ++", pElemOut);    // dm4:

// Keep a copy of the inserted <A> element and increment refcount
pElemA = pElemOut;
pElemA->AddRef();
ShowRefCountOf("pElemA ++", pElemA);    // dm5:

count = pElem->Release();
ShowRefCount("pElem --", count);  // dm6:
pElem = NULL;

count = pElemOut->Release();
pElemOut = NULL;
ShowRefCount("pElemOut --", count);  // dm7:

// The refCount of pElemA should be the same as that of pElemOut, because they both reference the same object.
ShowRefCountOf("pElemA 		", pElemA);  // dm8:

// Create an <a> element with a text value of "11".
bstr = SysAllocString(L"a");
pElem = ElementFromDom(bstr, pDom);
if (pElem==NULL) {
    goto clean;
}
SysFreeString( bstr);
bstr = NULL;

bstr = SysAllocString(L"11");
HRCALL( pElem->put_text(bstr),
    "put_text(...)"");
ShowRefCountOf("pElem ++", pElem);  // dm9:
SysFreeString( bstr);
bstr = NULL;

// Append <a> to <A>.
HRCALL( pElemA->appendChild(pElem, &pElemOut),
    "append <a> to <A>");
ShowRefCountOf("pElemOut ++", pElemOut);  // dm10:

// We're done with pElemA.
count = pElemA->Release();
pElemA = NULL;
ShowRefCount("pElemA --", count);  // dm11:

count = pElem->Release();
pElem = NULL;
ShowRefCount("pElem --", count);  // dm12:

count = pElemOut->Release();
pElemOut = NULL;
ShowRefCount("pElemOut --", count);  // dm13:

// Create a <B> element with a text value of "2".
bstr = SysAllocString(L"B");
pElem = ElementFromDom(bstr, pDom);
if (!pElem)
    goto clean;
SysFreeString(bstr);
bstr = NULL;

bstr = SysAllocString(L"2");
HRCALL(pElem->put_text(bstr),
    "put_text(...)");
ShowRefCountOf("pElem ++", pElem);  // dm14:
SysFreeString(bstr);
bstr = NULL;

// Append <B> to <root>.
HRCALL(pRoot->appendChild(pElem, &pElemOut),
    "append <B> to <root>");
ShowRefCountOf("pElemOut ++", pElemOut);  // dm15:

// We're done with pElem, and recycle pElemOut.
count = pElem->Release();
pElem = NULL;
ShowRefCount("pElem --", count);  // dm16:

count = pElemOut->Release();
pElemOut = NULL;
ShowRefCount("pElemOut --", count);  // dm17:

// Append <root> to dom.
HRCALL(pDom->appendChild(pRoot, &pElemOut),
    "append <root> to dom");
ShowRefCountOf("pElemOut ++", pElemOut);  // dm18:

// We're done with pElemOut.
count = pElemOut->Release();
pElemOut = NULL;
ShowRefCount("pElemOut --", count);  // dm19:

HRCALL(pDom->get_xml(&bstr),
    "get_xml()");
dprintf("\npDom->xml:%S\n", bstr);

clean:
    if (bstr)
        SysFreeString(bstr);

    if (pElem)
        pElem->Release();
    if (pElemA)
        pElemA->Release();
    if (pElemOut)
pElemOut->Release();

if (pDom) {
    count = pDom->Release();
    ShowRefCount("pDom --", count); // dm20:
}

if (pRoot) {
    count = pRoot->Release();
    ShowRefCount("pRoot --", count);  // dm21:
}

CoUninitialize();
return 0;
}

**To add refCount.cpp to the project**

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file `refCount.cpp`.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Now build and run the application. The output is listed next.
Output

When you build and run the `refCount` project, you should get the following output in a console window:

```
dm1:  refCount(pDom ++) = 1
dm2:  refCount(pRoot ++) = 1
dm3:  refCount(pElem ++) = 1
dm4:  refCount(pElemOut ++) = 2
dm5:  refCount(pElemA ++) = 3
dm6:  refCount(pElem --) = 2
dm7:  refCount(pElemOut --) = 1
dm8:  refCount(pElemA ) = 1
dm9:  refCount(pElem ++) = 1
dm10: refCount(pElemOut ++) = 2
dm11: refCount(pElemA --) = 0
dm12: refCount(pElem --) = 1
dm13: refCount(pElemOut --) = 0
dm14: refCount(pElem ++) = 1
dm15: refCount(pElemOut ++) = 2
dm16: refCount(pElem --) = 1
dm17: refCount(pElemOut --) = 0
dm18: refCount(pElemOut ++) = 2
dm19: refCount(pElemOut --) = 1

pDom->xml:

<root><A><a>11</a></A><B>2</B></root>

dm20:  refCount(pDom --) = 0
dm21:  refCount(pRoot --) = 0
```

Remarks

The following remarks explain the various reference counts shown above. However, because reference counting is implementation-dependent, these discussions might or might not apply to other objects.

Messages dm1: - dm3:

Show three newly created objects: `pDom`, which corresponds to the document element; `pRoot`, which corresponds to the `<root>` element; and `pElem`, which corresponds to the `<A>` element. Because these objects are
new and distinct, the reference count equals 1 on each interface pointer.

**Message dm4:**
Shows the reference count of pElemOut as returned from the call to pRoot->appendChild method. Because pElemOut is another reference to the pElem object, when the call succeeds its reference count is 1 plus the count on pElem—that is, 2.

**Message dm5:**
Shows the reference count of pElemA, which is a copy of pElemOut; pElemA is used to hold the <A> element, so that pElem and pElemOut can be recycled to represent other elements. So far, pElem, pElemOut, and pElemA all point to the same object. Therefore, the reference count on pElemA is 3.

**Messages dm6: - dm8:**
Show the reference counts on pElem, pElemOut, and pElemA after the Release method has been called on pElem and pElemOut. These two interface pointers need to be released, because they will be used to add another element, <a>, to the XML document. Remember that at this point in the code all three interface pointers still reference the same object. This means that the reference counts on them are correlated with each other. Failure to release pElem or pElemOut here will result in a memory leak.

**Messages dm9:**
Shows the reference count on the pElem interface pointer, which points to a newly created IXMLDOMNode object. This object, which represents the <a> element, is to be added as a child of <A>. Because this is a newly created object, the reference count on this interface pointer is 1.

**Message dm10:**
Shows the reference count of pElemOut as returned from the call to pElemA->appendChild method. This element is another reference to the pElem object, <a>. Therefore, when the call succeeds, its reference count is 1 plus the reference count on pElem—that is, 2.

**Messages dm11: - dm13:**
Show the reference counts on pElemA, pElem, and pElemOut after the Release method was called on each of the interface pointers. The system has released pElemA because we are done with <A>; when the reference count on pElemA reaches zero, the system frees the memory object pointed to by the pElemA interface pointer. Any attempt to access this object will result in access violation. The other two interface pointers are released because we are ready to move on to the next element, <B>. Failing to release pElem and pElemOut here would result in a memory leak.

**Messages dm14:**
Shows the reference count on the pElem interface pointer, which points to a newly created IXMLDOMNode object, <B>. This object is to be added as a child of <root>. Because pElem is not a copy of another interface pointer, the reference count is 1.

**Message dm15:**
Shows the reference count of pElemOut as returned from the call to pRoot->appendChild method. This element is another reference to the pElem object, <B>. Therefore, when the call succeeds, its reference count is 1 plus the reference count on pElem—that is, 2.

**Messages dm16: - dm17:**
Show the reference counts on pElem and pElemOut after the Release method was called on each of the interface pointers. These two interface pointers are released because we are done with <B>. Failing to release pElemOut here would result in memory leak.

**Message dm18:**
Shows the reference count of pElemOut as returned from the call to pDom->appendChild method. This element is another reference to the pRoot object, <root>. Therefore, when the call succeeds, its reference count is 1 plus the reference count on pRoot—that is, 2.

**Messages dm19: - dm21:**
Show the reference counts on pElemOut, pDom, and pRoot after the Release method has been called on each of the interface pointers. These interface pointers are released because they are no longer needed. All the memory objects allocated in this application are freed when the application ends.
Use OnReadyStateChange in Visual C/C++ Applications

In C++, use connection points to trap all XMLDOMDocument events. The DISPID of the connection point container is DIID_XMLDOMDocumentEvents. The DISPID for the connection point to trap the readystatechange event is DISPID_XMLDOMEVENT_ONREADYSTATECHANGE.

For a C/C++ example, see onreadystatechange Event.
Program with DOM in C/C++ Using Smart Pointer Class Wrapper

This tutorial is intended for C/C++ developers interested in writing XML applications using the DOM APIs as implemented in Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office. It includes a series of simple examples written in C and tested using Microsoft Visual C++ 6.0. The objective is to provide a quick and practical orientation, so that you can start to write your own XML applications.

These demonstrations are task-oriented, and are simplified for clarity. Therefore, their implementation is not always optimized.

This tutorial consists of the following demonstrations.

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Ready to Work with MSXML (Smart)</td>
<td>Discusses the requirements for using MSXML, and describes how to install the MXSML components.</td>
</tr>
<tr>
<td>Load an XML DOM Object from an XML File (Smart)</td>
<td>Demonstrates how to create an XML DOM instance and load its content from an external XML data file.</td>
</tr>
<tr>
<td>Save an XML DOM Object to a File (Smart)</td>
<td>Demonstrates how to serialize an XML DOM object in a text file.</td>
</tr>
<tr>
<td>Perform XSL Transformations (Smart)</td>
<td>Demonstrates how to perform XLS Transformations.</td>
</tr>
<tr>
<td>Create an XML DOM Object Dynamically (Smart)</td>
<td>Demonstrates how to create an XML DOM object programmatically, including processing instructions, comments, elements, attributes, CDATA sections, and text nodes.</td>
</tr>
<tr>
<td>Query DOM Nodes (Smart)</td>
<td>Demonstrates how to query a DOM node or node-set using XPath expressions.</td>
</tr>
<tr>
<td><strong>Make XML Requests Over HTTP (Smart)</strong></td>
<td>Demonstrates how to perform client requests of XML data from a web server.</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Validate an XML Document or (Smart)</strong></td>
<td>Demonstrates how to validate an XML document and/or fragment against an XML Schema.</td>
</tr>
</tbody>
</table>
Get Ready to Work with MSXML (Smart)

Programming with MSXML in C/C++ with smart pointer class wrappers requires the same basic components as programming without smart pointer class wrappers:

- An appropriate version of MSXML installed on your machine.
- The MSXML libraries and headers incorporated into your project.

See Get Ready to Work with MSXML (C/C++) and Set Up My Visual C++ Project for instructions on how to install these components and use them in Visual C++.

This topic describes how to incorporate headers and libraries into your project.
Importing Headers and Libraries

After MSXML is installed, you need to set up your application project so that calls to the MSXML-supported APIs are resolved properly when the application is built. In Microsoft Visual C++, you must import the MSXML headers and libraries into your project. This is a fairly simple task if you intend to use smart pointer classes to access interface pointers.

Calling Interfaces Using Smart Pointer Classes

If you use the DOM interfaces through their smart pointer class wrappers, you can simply insert the following statements at the beginning of your application code:

```cpp
#import <msxml5.dll>
using namespace MSXML2;
```

The first statement instructs Visual C++ to generate the type library information contained in the `msxml5.dll` library. As a result, two header files, `msxml5.tlh` and `msxml5.tli`, are created in one of your project's folders. These files contain the required type library information, with interfaces also wrapped in smart pointer classes. You might find it helpful to examine the `.tlh` and `.tli` files if you have not used them before. The second statement imports the namespace to your project, so that you can reference the DOM APIs without specifying their namespace explicitly.
Using Smart Pointer Classes

There are many advantages to using smart pointer classes. In addition to automating some object management tasks, such as calling the `AddRef` method or the `Release` method on an interface pointer, they make the API calling convention in C/C++ more consistent with that in script or Visual Basic. This is helpful to programmers who frequently use these languages.

You might want to use the raw COM interface pointers when performance is an important issue. The Program with DOM in C/C++ tutorial shows you how to do that.

The syntax used to call the XML DOM methods or properties using smart pointer classes differs from that using raw interface pointers. The following topics provide more information about using smart pointer classes.

- [Smart Pointer Classes and DOM Methods with an [out, retval] Parameter](#)
- [Smart Pointer Classes and DOM Properties](#)
Smart Pointer Classes and DOM Methods with an [out, retval] Parameter

When a method has an [out, retval] parameter, the syntax is different depending on whether you use smart pointer class wrappers or the raw interface method. We use the load method as an example.

**Raw Interface Syntax**

```csharp
HRESULT load(
    [in] VARIANT xmlSource,
    [out, retval] VARIANT_BOOL *isSuccessful
);
```

For example,

```csharp
hr = pXMLDom->load("myData.xml", &vbStatus);
```

where hr, pXMLDom, and vbStatus are of the HRESULT, IXMLDOMDocument*, and VARIANT_BOOL types, respectively.

**Smart Pointer Class Wrapper Syntax**

```csharp
VARIANT_BOOL load(
    [in] VARIANT xmlSource
);
```

For example,

```csharp
vbStatus = pXMLDom->load("myData.xml");
```

Notice that the DOM method call using smart pointer classes is similar to that in script or Visual Basic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Smart Pointer Classes and DOM Properties

The syntax for invoking DOM properties using raw interface pointers differs from that using smart pointer classes. We use the async property as an example.

**Raw Interface Syntax**

```c
HRESULT get_async(
    [out, retval] VARIANT_BOOL *isAsync
);
HRESULT put_async(
    [in] VARIANT_BOOL isAsync
);
```

For example,

```c
hr = pXMLDom->get_async(&vbAsync);
if (vbAsync == VARIANT_TRUE)
{
    hr = pXMLDom->put_async(VARIANT_FALSE);
}
```

where hr, pXMLDom, and vbAsync are of the HRESULT, IXMLDOMDocument*, and VARIANT_BOOL types, respectively.

**Smart Pointer Class Wrapper Syntax**

```c
VARIANT_BOOL async
```

For example,

```c
If (pXMLDom->async == VARIANT_TRUE)
{
    pXMLDom->async = VARIANT_FALSE;
}
```

Notice that invoking the DOM property using smart pointer classes is similar to using script or Visual Basic.
Now we are ready start working with XML DOM. First, we'll Load an XML DOM Object from an XML File.
Load an XML DOM Object from an XML File (Smart)

To load an XML DOM object from an XML data file, you use the load method. The following Microsoft Visual C++ 6.0 project illustrates this procedure.

This project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: loadDOMsmart.cpp</td>
<td>Creates an XML DOM object and loads the resource file into the project's main directory.</td>
</tr>
<tr>
<td>Output</td>
<td>This is the output you should get when you build and run the loadDOMsmart project.</td>
</tr>
</tbody>
</table>

Create the loadDOMsmart Visual C++ Project

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project LoadDOMsmartProj.

Next, we'll add the source code for the loadDOMsmart project.
Source: loadDOMsmart.cpp

This C/C++ source code performs the following steps:

1. Creates an XML DOM object (pXMLDom) and sets it to synchronous mode.
2. Calls the load method on pXMLDom, specifying the path to stocks.xml.

C/C++ Source File (loadDOMsmart.cpp)

```c
#include<stdio.h>
#import<msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[])
{
    IXMLDOMDocument3Ptr pXMLDom;
    HRESULT hr;
    CoInitialize(NULL);
    hr = pXMLDom.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr))
    {
        printf("Failed to instantiate an XML DOM.\n");
        return -1;
    }
    pXMLDom->async = VARIANT_FALSE; // default - true,
    if(pXMLDom->load("stocks.xml")!=VARIANT_TRUE)
    {
        printf("Failed to load stocks.xml: \n%s\n", (LPCSTR)pXMLDom->parseError->Getreason());
        return -1;
    } else
    {
        printf("XML DOM loaded from stocks.xml: \n%s\n", (LPCSTR)pXMLDom->xml);
        pXMLDom.Release();
        CoUninitialize();
        return 0;
    }
}
```

To add the loadDOMsmart source code to the project
1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file loadDOMsmart.cpp.

2. Copy the C/C++ source code above, and paste it into the source file you just created.

Next, we'll add the resource file, stocks.xml.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: stocks.xml

XML Data File (stocks.xml)

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

To add the stocks.xml resource file to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file `stocks.xml`
2. Copy the resource listing above and paste it into the source file you just created.

   **Note** You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

Next, **build and run** the `loadDOMsmart` project. The result should be the **output** shown in the following topic.
When you build and run the **loadDOMsmart** project, you should get the following output in a console window:

```xml
XML DOM loaded from stocks.xml:
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

The next topic in this tutorial demonstrates how to save an XML DOM object to a file.
Save an XML DOM Object to a File (Smart)

Saving an XML DOM object is also known as serializing the DOM. It involves a call to the save method on the DOM object. To demonstrate this, we will continue from the previous exercise, and save the DOM object that was loaded from stocks.xml to a different file, myStocks.xml.

This project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: saveDOMsmart.cpp</td>
<td>Saves an XML DOM object, pXMLDom, to a new file. When you build and run the saveDOMsmart project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

Create the loadDOMsmart Visual C++ project

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project saveDOMsmartProj.

Next, we'll add the source code for the saveDOMsmart project.
Source: saveDOMsmart.cpp

This C/C++ source code performs the following steps:

1. Creates an XML DOM object (pXMLDom) and sets it to synchronous mode.
2. Loads an XML string to pXMLDom.
3. Calls the loadXML method on pXMLDom, specifying the XML data as the following string:
   `<r>
   <t>top</t>
   <b>bottom</b>
   </r>`
4. Displays the resulting XML DOM in the console window.
5. Calls the save method on pXMLDom to serialize the DOM content to a file (myData.xml).

C/C++ Source File (saveDOMsmart.cpp)

```c++
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    IXMLDOMDocument3Ptr pXMLDom;
    HRESULT hr;

    CoInitialize(NULL);
    hr = pXMLDom.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to instantiate an XML DOM.\n");
        return -1;
    }

    pXMLDom->async = VARIANT_FALSE; // The default is true.

    if(pXMLDom->loadXML("<r>
    <t>top</t>
    <b>bottom</b>
    </r>"))!=VARIANT_TRUE) {
        printf("Failed to loadXML:\n%s\n", (LPCSTR)pXMLDom->parseError->Getreason());
        return -1;
    }

    ....
```

```c++
```

```c++
```
else
    printf("XML DOM loaded from app:\n%s\n", (LPCSTR)pXMLDom->xml);

hr = pXMLDom->save("myData.xml");
if (FAILED(hr))
{
    printf("Failed to save DOM to myStocks.xml.\n");
    return -1;
}
else
    printf("XML DOM saved to myData.xml.\n");

pXMLDom.Release();
CoUninitialize();
return 0;
}

**To add the saveDOMsmart source code to the project**

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file `saveDOMsmart.cpp`.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, **build and run** the `saveDOMsmart` project. The result should be the **output** shown in the following topic.
Output

The **saveDOMsmart** project outputs a file, **myData.xml**, in the project's main directory. In addition to this file, this application produces the following output in a console window.

XML DOM loaded from app:
<
   <t>top</t>
   <b>bottom</b>
</

XML DOM saved to myData.xml.

The next topic in this tutorial demonstrates how to use DOM to perform XSLT.
Perform XSL Transformations (Smart)

In general, performing XSL Transformations (XSLT) requires two DOM objects: one for the XML data file, and one for the XSLT style sheet. It also involves calling either the `transformNode` method or the `transformNodeToObject` method on the DOM object that corresponds to the XML data. The DOM object holding the XSLT style sheet is the input for the method. The first method returns the result as a string, and the second method returns the result as an object.

In the following C/C++ Win32 Console Application project, we demonstrate how to do this. The XSLT project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source:</strong> XSLTsmart.cpp</td>
<td>Transforms an XML data file with an XSLT style sheet.</td>
</tr>
<tr>
<td><strong>Resource:</strong> stocks.xml and stocks.xsl</td>
<td>An XML data file and an XSLT style sheet.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>When you build and run the XSLT project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

Create the XSLT project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the project XSLTsmartProj.

Next, we'll add the source code for the XSLTsmart project.
The source code listed below demonstrates how to call the transformNode method and the transformNodeToObject method in a C/C++ program. Specifically, the code performs the following steps:

1. Loads the XML data file (stocks.xml) into an XML DOM object (pXMLDom).
2. Loads the XSLT style sheet (stocks.xsl) into an XML DOM object (pXSLDoc).
3. Calls the transformNode(pXSLDoc) method on pXMLDom to do the transformation, holds the result in a string (xmlStr), and prints the output to the console.
4. Creates an XML DOM object (pXMLOut) to hold the output of the transformNodeToObject method.
5. Calls the transformNodeToObject method on pXMLDom to do the transformation; holds the resulting object in pXMLOut; prints out the XML result; and serializes the output object in an HTML file, stocks.htm.

C/C++ Source File (XSLTsmart.cpp)

```c
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    IXMLDOMDocument3Ptr pXMLDom, pXSLDoc, pXMLOut;
    HRESULT hr;

    CoInitialize(NULL);
    // Load the XML file.
    hr = pXMLDom.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to instantiate an XML DOM.\n");
        return -1;
    }

    pXMLDom->async = VARIANT_FALSE; // The default is true.

    if(pXMLDom->load("stocks.xml")!=VARIANT_TRUE)
```
printf("Failed to load stocks.xml:\n%s\n",
  (LPCSTR)pXMLDom->parseError->Getreason());
return -1;
}

// Load the XSLT style sheet.
hr = pXSLDoc.CreateInstance(__uuidof(DOMDocument50));
if (FAILED(hr))
{
  printf("Failed to instantiate an XML DOM.\n"");
  return -1;
}

pXSLDoc->async = VARIANT_FALSE; // The default is true.
if(pXSLDoc->load("stocks.xsl")!=VARIANT_TRUE)
{
  printf("Failed to load stocks.xsl:\n%s\n",
    (LPCSTR)pXMLDom->parseError->Getreason());
  return -1;
}

// Transform the XSLT to an XML string.
_bstr_t xmlStr = pXMLDom->transformNode(pXSLDoc);
// Always remember to check for parse errors.
if(pXMLDom->parseError->errorCode != 0)
{
  printf("Failed to transformNode:\n%s\n",
    (LPCSTR)pXMLDom->parseError->Getreason());
}
else
{
  printf("output from transformNode:\n%s\n",
    (LPCSTR)xmlStr);
}

// Instantiate a DOM for xmlOut object.
hr = pXMLOut.CreateInstance(__uuidof(DOMDocument50));
if (FAILED(hr))
{
  printf("Failed to instantiate an XML DOM.\n"");
  return -1;
}

// Transform the XSLT to a DOM object.
hr = pXMLDom->transformNodeToObject(pXSLDoc,
  pXMLOut.GetInterfacePtr());
if (FAILED(hr))
{

printf("Failed to transformNodeToObject:\n%s\n",
   (LPCSTR)pXMLDom->parseError->Getreason());
}
else
{
   hr = pXMLOut->save("stocks.htm");
   if (FAILED(hr))
   {
      printf("Failed to save output DOM to xslt_out.htm\n");
      return -1;
   }
   else
   {
      printf("Output from transformNodeToObject:\n%s\n",
             (LPCSTR)pXMLOut->xml);
      printf("The above output is also saved in stocks.htm\n");
   }
}
pXMLDom.Release();
pXSLDoc.Release();
pXMLOut.Release();
CoUninitialize();

   return 0;
}

To add the XSLT source code to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file XSLTsmart.cpp.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, we'll add the resource files to the XSLT project.
Resource: stocks.xml and stocks.xsl

**XML Data File (stocks.xml)**

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <stock exchange="nasdaq">
        <name>new</name>
        <symbol>zzzz</symbol>
        <price dt:dt="number">20.313</price>
    </stock>
    <stock exchange="nyse">
        <name>zacx corp</name>
        <symbol>ZCXM</symbol>
        <price dt:dt="number">28.875</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zaffymat inc</name>
        <symbol>ZFFX</symbol>
        <price dt:dt="number">92.250</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zysmergy inc</name>
        <symbol>ZYSZ</symbol>
        <price dt:dt="number">20.313</price>
    </stock>
</portfolio>
```

**XSLT Style Sheet (stocks.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="/">
        <HTML>
            <BODY>
                <TABLE BORDER="2">
                    <TR>
                        <TD>Symbol</TD>
                        <TD>Name</TD>
                        <TD>Price</TD>
                    </TR>
                    <!-- Use xsl:apply-templates... -->
                    <xsl:apply-templates select="portfolio/stock">
                    </xsl:apply-templates>
                </TABLE>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
To add stocks.xml and stocks.xsl to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the new file `stocks.xml`
2. Copy `stocks.xml` from above and paste it into the source file you just created.
3. Create another new C++ source file.
4. Copy the XSD file above and paste it into the source file you just created.

   **Note** You can also copy the files into the project's main directory using Windows Explorer (or a command prompt).

Next, **build and run** the XSLTsmart project. The result should be the **output** shown in the following topic.
Output

The XSLT project outputs an HTML file, stocks.xml, in the project's main directory. In addition to this file, the application produces the following XSLT-related output in a console window. Note that the formatting of the transformNode output is different from that of the transformNodeToObject output.

```
output from transformNode:
<HTML>
<BODY>
<TABLE BORDER="2">
<TR>
<TD>Symbol</TD>
<TD>Name</TD>
<TD>Price</TD>
</TR>
<TR STYLE="color:green" Title="ZCXM is listed on the nyse stock exchange."><TD>ZCXM</TD>
<TD>zacx corp</TD>
<TD>28.875</TD>
</TR>
<TR STYLE="color:red" Title="ZFFX is listed on the nasdaq stock exchange."><TD>ZFFX*</TD>
<TD>zaffymat inc</TD>
<TD>92.250</TD>
</TR>
<TR STYLE="color:green" Title="ZYSZ is listed on the nasdaq stock exchange."><TD>ZYSZ*</TD>
<TD>zysmergy inc</TD>
<TD>20.313</TD>
</TR>
<TR STYLE="color:green" Title="zzzz is listed on the nasdaq stock exchange."><TD>zzzz*</TD>
<TD>new</TD>
<TD>20.313</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```
The above output is also saved in stocks.htm.

The next topic in this tutorial demonstrates how to create an XML DOM object dynamically.
Create an XML Document Dynamically (Smart)

We have already demonstrated how to create an XML DOM object from an existing XML data file. In this section, we will show you how to create an XML document dynamically in an application. Specifically, we will demonstrate how to add processing instructions, comments, elements, and text nodes. We will also show how to use extra white space text nodes to provide a structured format when the DOM object is serialized. Such tasks are useful for creating an XML document that is also easy for humans to read.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: dynamDOMsmart.cpp</td>
<td>Creates an XML document dynamically.</td>
</tr>
<tr>
<td></td>
<td>When you build and run the dynamDOMsmart project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

Create the dynamDOMsmart project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project dynamDOMsmartProj.

Next, we'll add the source code for the dynamDOMsmart project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
**Source: dynamDOMsmart.cpp**

This application creates a simple, but complete, XML DOM object, with `<root>` as the document element. This element contains three child elements: `<node1>`, `<node2>`, and `<node3>`. The first child element contains character data. The second child element contains a CDATA section. The last child element contains three empty child elements: `<subnode1>`, `<subnode2>`, and `<subnode3>.

Programmatically, the **dynamDOMsmart** application performs the following steps:

1. Creates an XML DOM instance (pXMLDom).
2. Calls the `createProcessInstruction` method on pXMLDom. This creates a processing instruction node (pi) targeted for XML 1.0.
3. Calls the `appendChild` method on pXMLDom. This adds the processing instruction node (pi) to pXMLDom.
4. Calls the `createComment` method on the DOM object (pXMLDom) to create a comment node (pc) and then append it pXMLDom.
5. Creates a `<root>` element as the document element, with a created attribute whose value is set to a string value of "using DOM". Adds this element (<root>) to the DOM object (pXMLDom).
6. Creates a `<node1>` element with some character data as its content. Appends this element (pe) to the document element (documentElement) of the DOM object (pXMLDom).
7. Creates a `<node2>` element that contains a CDATA section (pcd) with markup text. Appends this element (pe) to the document element (documentElement) of the DOM object (pXMLDom).
8. Creates a `<node3>` element that contains a DOM document fragment (pdf). This fragment contains three other empty child elements: `<subNode1>`, `<subNode2>`, and `<subNode3>`. The code then appends this element (pe) to the document element (documentElement) of the DOM object (pXMLDom).
9. Saves this dynamically created DOM object to the project's main directory, and prints the XML data in the application console.

**C/C++ Source File (dynamDOMsmart.cpp)**

```c
#include <stdio.h>
```
#import <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    IXMLDOMDocument3Ptr pXMLDom;
    HRESULT hr;

    CoInitialize(NULL);

    hr = pXMLDom.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XML DOM");
        return NULL;
    }
    pXMLDom->preserveWhiteSpace = VARIANT_TRUE;

    // Create a processing instruction targeted for xml.
    IXMLDOMProcessingInstructionPtr pi;
    pi = pXMLDom->createProcessingInstruction("xml", "version='1.0'"四周);
    if (pi != NULL) {
        pXMLDom->appendChild(pi);
        pi.Release();
    }

    // Create a processing instruction targeted for xml-stylesheet.
    pi = pXMLDom->createProcessingInstruction("xml-stylesheet", "type='text/xml' href='dom.xsl'");
    if (pi != NULL) {
        pXMLDom->appendChild(pi);
        pi.Release();
    }

    // Create a comment for the document.
    IXMLDOMCommentPtr pc;
    pc = pXMLDom->createComment("sample xml file created using XML DO");
    if (pc != NULL) {
        pXMLDom->appendChild(pc);
        pc.Release();
    }

    // Create the root element (i.e., the documentElement).
    IXMLDOMElementPtr pe;
    pe = pXMLDom->createElement("root");

    // Create a "created" attribute for the root element and
    // assign the "using dom" character data as the attribute value.
    IXMLDOMAttributePtr pa;
    pa = pXMLDom->createAttribute("created");
if (pa != NULL)
{
    pa->value = "using dom";
    pe->setAttributeNode(pa);
    pa.Release();
}

// Add the root element to the DOM instance.
pXMLDom->appendChild(pe);
pe.Release();

// Next, we will create and add more nodes to the root element we've just created.

// Create an element to hold text content.
pe = pXMLDom->createElement("node1");
if (pe != NULL)
{

    // Add newline + tab for indentation.
    pXMLDom->documentElement->appendChild(pXMLDom->createTextNode("    

    pe->text = "some character data";
    pXMLDom->documentElement->appendChild(pe);
    pe.Release();
}

// Create an element to hold a CDATA section.
pe = pXMLDom->createElement("node2");
if (pe != NULL)
{

    // Add newline + tab for indentation.
    pXMLDom->documentElement->appendChild(pXMLDom->createTextNode("    

    IXMLDOMCDATASectionPtr pcd;
    pcd = pXMLDom->createCDATASection("<some mark-up text>");
    if (pcd != NULL) {
        pe->appendChild(pcd);
        pcd.Release();
    }
    pXMLDom->documentElement->appendChild(pe);
    pe.Release();
}

// Create an element to hold three empty subelements.
pe = pXMLDom->createElement("node3");
if (pe != NULL)
{

    // Add newline + tab for indentation.
To add the dynamDOM source code to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file dynamDOM.cpp.
2. Copy the C/C++ source code above and paste it into the source file you just created.

Next, build and run the dynamDOM project. The result should be the output shown in the following topic.
Output

The **dynamDOMsmart** application outputs an XML data file, **dynamDom.xml**, in the project's main directory. In addition to this file, you should see the following output in a console window.

Dynamically created DOM:

```xml
<?xml version="1.0"?>
<?xml-stylesheet type='text/xml' href='dom.xsl'?>
<!--sample xml file created using XML DOM object.-->
<root created="using dom">
  <node1>some character data</node1>
  <node2><![CDATA[<some mark-up text>]]></node2>
  <node3>
    <subNode1/>
    <subNode2/>
    <subNode3/>
  </node3>
</root>
```

DOM saved to dynamDom.xml

The next topic demonstrates how to query nodes from DOM.
Query Nodes from DOM (Smart)

MSXML provides two DOM methods that allow you to query a node or a set of nodes in an XML DOM object: SelectNodes and SelectSingleNode. You specify the node or nodes of interest in an XPath expression, which is taken as the input to these methods.

In this C/C++ demonstration, we assume that you have some understanding of XPath expressions.

This project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source:</strong> queryNodeSmart.cpp</td>
<td>Selects both a single node and a node-set from a DOM object.</td>
</tr>
<tr>
<td><strong>Resource:</strong> stocks.xml</td>
<td>An XML data file.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>When you build and run the <strong>queryNodesSmart</strong> project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

Create the queryNodesSmart project in Visual C++

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project queryNodesSmartProj.

Next, we'll add the source code for the queryNodesSmart project.

See Also

XPath Developer's Guide | XPath Reference
This example application calls both the `selectSingleNode` method and the `selectNodes` method on an XML DOM object loaded from the `stocks.xml` file. The same XPath expression is input to both methods:

```
//stock[1]/`
```

This expression specifies all the child elements of the first `<stock>` element in the XML document. In MSXML, the `selectSingleNode` method returns the first element of the resultant node-set, and the `selectNodes` method returns all the elements in the node-set.

Programmatically, this source code performs the following steps:

1. Creates an instance of an XML DOM object (`pXMLDom`).
2. Calls the `load` method on `pXMLDom` to load XML data from a file (`stocks.xml`).
3. Calls the `selectSingleNode` method on `pXMLDom` to select a DOM node according to the input XPath expression (`//stock[1]/`). Then displays the resulting node in the console if the query is successful.
4. Calls the `selectNodes` method on `pXMLDom` to select a DOM node-set according to the input XPath expression (`"//stock[1]/"`). Then displays the resulting nodes in the console if the query is successful.

C/C++ Source File (`queryNodesSmart.cpp`)

```c++
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[])
{
    IXMLDOMDocument3Ptr pXMLDom;
    HRESULT hr;
    CoInitialize(NULL);

    hr = pXMLDom.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr))
```
pXMLDom->async = VARIANT_FALSE;
if ( pXMLDom->load("stocks.xml") != VARIANT_TRUE )
{
    printf("Failed load xml data from file.\n%s\n", (LPCSTR)pXMLDom->parseError->Getreason());
    return -1;
}

// Query a single node.
IXMLDOMNodePtr pNode = pXMLDom->selectSingleNode("//stock[1]/*");
if ( pNode == NULL )
{
    printf("Invalid node fetched.\n%s\n", (LPCSTR)pXMLDom->parseError->Getreason());
}
else
{
    printf("Result from selectSingleNode:\nNode, <%s>:\n%" , (LPCSTR)pNode->nodeName, (LPCSTR)pNode->xml);
}

// Query a node-set.
IXMLDOMNodeListPtr pnl = pXMLDom->selectNodes("//stock[1]/*");
printf("Results from selectNodes:\n");
for ( int i=0; i<pnl->length; i++ )
{
    pNode = pnl->item[i];
    printf("Node (%d), <%s>:\n%" , i, (LPCSTR)pNode->nodeName, (LPCSTR)pnl->item[i]->xml);
}

pXMLDom.Release();
pNode.Release();
pnl.Release();
CoUninitialize();
return 0;

To add the queryNodes source code to the project

1. Select FileView on the project browser, and highlight
   queryNodeSmartProj files. From the File menu, select New.
2. On the Files tab of the New dialog box, highlight C++ Source File. Then
type "queryNodes.cpp" in the **File name** text box, and click **OK**.

3. Copy the C/C++ source code above and paste it into the source file you just created.

Next, we'll add the [resource file](#) to the **queryNodesSmart** project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: stocks.xml

XML Data File (stocks.xml)

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="stock.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

To add the stocks.xml resource file to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file `queryNodesSmart`.
2. Copy `stocks.xml` from above and paste it into the source file you just created.

Next, build and run the `queryNodesSmart` project. The result should be the output shown in the following topic.
Output

After you build and run the application, you should see the following output displayed in the application's console.

Result from selectSingleNode:
Node, <name>:
   <name>new</name>

Results from selectNodes:
Node (0), <name>:
   <name>new</name>
Node (1), <symbol>:
   <symbol>zzzz</symbol>
Node (2), <price>:
   <price xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="number">20.313</price>

Notice that the output above confirms that the selectSingleNode method picks out only the first node from the node set as specified by the XPath expression (//stock[1]/*). The selectNodes method returns the entire set of nodes specified by this XPath expression.

The next topic demonstrates how to validate XML.
Validate an XML Document or Fragment (Smart)

In MSXML 5.0 and later, there are two ways to validate XML data against a set of XML Schema definitions. One uses the `validate` method on an XML DOM instance to validate the entire DOM object. The other uses the `validateNode(node)` method on a DOM instance to validate a DOM fragment. A DOM fragment is a sub-tree rooted at the specified node.

The example in this tutorial illustrates both of these methods. This project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: validateDOMsmart.cpp</td>
<td>Validates an XML file against an XSD schema.</td>
</tr>
</tbody>
</table>

When you build and run the `validateDOMsmart` project, you should get this output in a console window.

**Create the validateDOMsmart project in Visual C++**

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see [Set Up My Visual C++ Project](#). Name the project `validateDOMsmartProj`.

Next, we'll add the [source code](#) for the `validateDOMsmart` project.
The source code performs the following basic steps:

1. Creates a DOM instance (pXMLDoc) to hold the XML data.
2. Creates a DOM instance (pXSDDoc) to hold the XML Schema definition.
3. Creates an IXMLSchemaCollection or IXMLSchemaCollection2 object (pSCache). This object is also called a schema cache. The application then adds the XML Schema definition (pXSDDoc) to the pSCache.
4. Associates pSCache with the schemas property of the DOM object for the XML data (pXMLDoc).
5. Calls the following validation methods on the DOM object for XML data (pXMLDoc):
   - Calls the validate method on pXMLDoc to validate the data set as a whole, and/or
   - Calls the validateNode(pNode) method on pXMLDoc to validate a node object (pNode) selected from pXMLDoc.

Checks the error returned from validate method and/or the validateNode(pNode) method, to determine if the specified XML data set is valid against the given XML Schema definition.

C/C++ Source File (validateDOMsmart.cpp)

```c
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

void dump_com_error(_com_error &e);

int main(int argc, char* argv[])
{
    IXMLDOMDocument3Ptr pXMLDoc = NULL;
    IXMLDOMDocument3Ptr pXSDDoc = NULL;
    IXMLDOMParseErrorPtr pError = NULL;
    IXMLDOMSchemaCollectionPtr pSCache = NULL;
    IXMLDOMNodePtr pNode = NULL;
    IXMLDOMNodeListPtr pNodelist= NULL;
    HRESULT hr;
```
CoInitialize(NULL);
try{

    // Load books.xml into a DOM instance.
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Cannot create DOMDocument50 instance for XMLDoc\n")
        return 0;
    }
    pXMLDoc->async = VARIANT_FALSE;
pXMLDoc->validateOnParse = VARIANT_FALSE;
    if(pXMLDoc->load("books.xml") != VARIANT_TRUE)
    {
        printf("Cannot load books.xml to XMLDoc object\n");
        return 0;
    }

    // Load books.xsd into a DOM instance.
    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Cannot create DOMDocument50 instance for XSDDoc\n")
        return 0;
    }
    pXSDDoc->async = VARIANT_FALSE;
pXSDDoc->validateOnParse = VARIANT_FALSE;
    if(pXSDDoc->load("books.xsd") != VARIANT_TRUE)
    {
        printf("Cannot load books.xsd to XSDDoc object\n");
        return 0;
    }

    // Create a schema cache instance.
    hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
    if (FAILED(hr)) {
        printf("Cannot create XMLSchemaCache50 instance in pSCache\n")
        return 0;
    }

    // Add the just-loaded schema definition to the schema collecti
    hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
    if (FAILED(hr)) {
        printf("Cannot add schema\n");
        return 0;
    }

    // Associate the schema collection with the XMLDoc instance.
    pXMLDoc->schemas = pSCache.GetInterfacePtr();

    // Validate the entire DOM.
    printf("Validating DOM...\n");
pError = pXMLDoc->validate();
if (pError->errorCode != 0)
    printf("\tXMLDoc is not valid because:\n%\n", (LPCSTR)pError->Getreason());
else
    printf("\tXMLDoc is validated: \n%\n", (LPCSTR)pXMLDoc->xml);

// Validate all //book nodes, node-by-node.
pNextodelist = pXMLDoc->selectNodes("//book");
printf("Validating all //book nodes, one by one ...
");
for (int i = 0; i < pNextodelist->length; i++)
{
    pNode = pNextodelist->item[i];
    pError = pXMLDoc->validateNode(pNode);
    if (pError->errorCode != 0)
        printf("\t<%s> (%d) is not valid because\n%\n", (LPCSTR)pNode->nodeName, i,
            (LPCSTR)pError->Getreason());
    else
        printf("\t<%s> (%d) is a valid node\n", (LPCSTR)pNode->nodeName,
            (LPCSTR)pNode->nodeName, i);
}

// Validate all children of all book nodes, //book/*, node-by-
pNextodelist = pXMLDoc->selectNodes("//book/*");
printf("Validating all children of all book nodes, //book/*, on
for (i = 0; i < pNextodelist->length; i++)
{
    pNode = pNextodelist->item[i];
    pError = pXMLDoc->validateNode(pNode);
    if (pError->errorCode != 0)
        printf("\t<%s> (%d) is not valid because\n%\n", (LPCSTR)pNode->nodeName, i,
            (LPCSTR)pError->Getreason());
    else
        printf("\t<%s> (%d) is a valid node\n", (LPCSTR)pNode->nodeName,
            (LPCSTR)pNode->nodeName, i);
}
}

catch(_com_error &e)
{
    dump_com_error(e);
}
pXMLDoc.Release();
pXSDDoc.Release();
pError.Release();
pSCache.Release();
pNode.Release();
pNextodelist.Release();
CoUninitialize();
void dump_com_error(_com_error &e) {
    printf("Error(dump_com_error)\n");
    printf("\tCode = %08lx\n", e.Error());
    printf("\tCode meaning = %s", e.ErrorMessage());
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
    printf("\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

To add the validateDOMsmart source code to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file validateDOMsmart.cpp.
2. Copy the C/C++ source code above, and paste it into the source file you just created.

Next, we'll add the resource files to the validateDOMsmart project.
Resource books.xml and books.xsd

This example uses two resources files. One is an XML data file (books.xml) and the other is the XML schema definition file (books.xsd). According to books.xsd, the second <book> element in books.xml is missing the required <pub_date> child element. Therefore, when we attempt to validate the XML file against the given schema, we should get a validation error.

XML File (books.xml)

```xml
<?xml version="1.0"?>
<x:books xmlns:x="urn:books">
  <book id="bk001">
    <author>Writer</author>
    <title>The First Book</title>
    <genre>Fiction</genre>
    <price>44.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>An amazing story of nothing.</review>
  </book>
  
  <book id="bk002">
    <author>Poet</author>
    <title>The Poet's First Poem</title>
    <genre>Poem</genre>
    <price>24.95</price>
    <review>Least poetic poems.</review>
  </book>
</x:books>
```

XSD File (books.xsd)

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:books"
            xmlns:bks="urn:books">

  <xsd:element name="books" type="bks:BooksForm"/>

  <xsd:complexType name="BooksForm">
    <xsd:sequence>
      <xsd:element name="book" type="bks:BookForm" minOccurs="0"
```
To add books.xml and books.xsd to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the new file books.xml.
2. Copy books.xml and paste it into the source file you just created.
3. Create another new C++ source file.
4. Copy books.xsd and paste it into the source file you just created.

Note You can also copy the files into the project's main directory using Windows Explorer (or a command prompt).

Next, build and run the validateDOMsmart project. The result should be the output shown in the following topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Output

After you build and run the project, the application should produce the following output in a console window:

Validating DOM...
XMLDoc is not valid because
Element content is invalid according to the DTD/Schema.
Expecting: pub_date.

Validating all //book nodes, one by one ...
  <book> (0) is a valid node
  <book> (1) is not valid because
Element content is invalid according to the DTD/Schema.
Expecting: pub_date.

Validating all children of all book nodes, //book/*, one by one ...
  <author> (0) is a valid node
  <title> (1) is a valid node
  <genre> (2) is a valid node
  <price> (3) is a valid node
  <pub_date> (4) is a valid node
  <review> (5) is a valid node
  <author> (6) is a valid node
  <title> (7) is a valid node
  <genre> (8) is a valid node
  <price> (9) is a valid node
  <review> (10) is a valid node

The next topic demonstrates how to make XML requests over HTTP.
Make XML Requests Over HTTP
(Smart)

This tutorial demonstrates how to request an XML data file from a web server using HTTP. The C++ application presented here is a client making the request. A local Web server (localhost) is used for the demonstration. The XML data consists of a set of contact information (contacts.xml) that is stored as an XML file on the server. The server fulfils the request through an ASP page (contact.asp), which is accessible from the localhost/sxh virtual directory.

This project uses the following files and directories.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: XMLOverHTTP.cpp</td>
<td>Retrieves and displays XML data from a Web server.</td>
</tr>
<tr>
<td>Virtual Directory: sxh</td>
<td>A virtual directory, with a corresponding directory on your drive.</td>
</tr>
<tr>
<td>Resource: contacts.xml</td>
<td>An XML data file containing contact information</td>
</tr>
<tr>
<td>ASP Page: contact.asp</td>
<td>An ASP page that queries a DOM object loaded from the contacts.xml file.</td>
</tr>
<tr>
<td>Output</td>
<td>This is the output you should get when you build and run the XMLOverHTTP project.</td>
</tr>
</tbody>
</table>

Create the XMLOverHTTP Visual C++ Project

- Create a Win32 console application in Visual C++. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the project XMLOverHTTPProj.

Next, we'll add the source code for the XMLOverHTTPProj project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Source: XMLOverHTTP.cpp

This C/C++ code performs the following steps:

1. Creates an IXMLHTTPRequest object, pIXMLHTTPRequest.
2. Uses the open method and the send method to request the contact information for "John Doe".
3. Uses the responseXML property to retrieve the XML data returned from the server.
4. Displays the XML data to the console.

C/C++ Source File (XMLOverHTTP.cpp)

#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[])
{
    CoInitialize(NULL);

    IXMLHTTPRequestPtr pIXMLHTTPRequest = NULL;
    IXMLDOMDocumentPtr pXMLDoc = NULL;
    BSTR bstrString = NULL;
    HRESULT hr;

    try {
        hr=pIXMLHTTPRequest.CreateInstance(__uuidof(XMLHTTP50));
        SUCCEEDED(hr) ? 0 : throw hr;

        hr=pIXMLHTTPRequest->open("GET",
            "http://localhost/sxh/contact.asp?SearchID=John Doe",
            false);
        SUCCEEDED(hr) ? 0 : throw hr;

        hr=pIXMLHTTPRequest->send();
        SUCCEEDED(hr) ? 0 : throw hr;

        pXMLDoc=pIXMLHTTPRequest->responseXML;
        bstrString=pXMLDoc->xml;
        if(bstrString)
        {
            printf("%S\n", bstrString);
        }
    }
}
SysFreeString(bstrString);
    bstrString = NULL;
}

catch (...) {
    printf("error: %x\n", hr);
}

if (bstrString)
    SysFreeString(bstrString);
if (pIXMLHTTPRequest)
    pIXMLHTTPRequest.Release();
if (pXMLDoc)
    pXMLDoc.Release();

CoUninitialize();
return 0;
}

To add the XMLOverHTTP source code to the project

1. Create a new C++ source file. For detailed instructions on how to do this, see Set Up My Visual C++ Project. Name the file XMLOverHTTP.cpp.
2. Copy the C/C++ source code above, and paste it into the source file you just created.

Next, we'll set up a virtual directory. You must complete this step before you build and run the application.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Virtual Directory: sxh

For this tutorial, we use an Internet Information Services (IIS) Web server hosted on the local host. Make sure IIS is installed on your machine before you continue.

To set up a virtual directory

1. Create a directory, such as c:\XMLOverHTTP, to map to this virtual directory.
2. Open IIS.
3. Navigate to Web Sites -> Default Web Site, and right-click.
4. Select New and click Virtual Directories to open the Virtual Directory Creation Wizard.
5. Click Next.
6. Enter "sxh" under Alias. Then click Next.
7. Click Browse... and navigate to the directory you created in Step 1. Click Next.
8. Accept the default access permissions by clicking Next.
9. Click Finish.

Next, we'll add the resource file, contacts.xml, to the directory you created in Step 1.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: contacts.xml

The XML data file contains two contact entries, each with a name and phone number.

XML Data File (contacts.xml)

```xml
<?xml version="1.0"?>
<contacts>
  <contact>
    <name>John Doe</name>
    <phone>123-456-7890</phone>
  </contact>
  <contact>
    <name>Jane Doe</name>
    <phone>123-456-7891</phone>
  </contact>
</contacts>
```

To add contacts.xml to the project

1. Create an empty text file in the directory you created to correspond to your virtual directory (for example, c:\XMLOverHTTP, corresponding localhost/sxh). Name the text file contacts.xml.
2. Copy the resource listing above, and paste it into the XML file you just created.

Next, we'll create an ASP page to handle the request.
ASP Page: contact.asp

The ASP page makes queries against a DOM object loaded from the contacts.xml file. For example, if the request comes with the a query string "SearchID=John Doe", the ASP page returns the contact information for John Doe. Otherwise, it returns the contact information of all listed persons. If the specified person is not a listed entry, the ASP returns an error to the client.

ASP page (contact.asp)

<%@language="javascript"%>
<%
    var xpath;
    var sName=Request.QueryString("SearchID");
    if (sName)
        xpath = "//contact[name='" + sName + "]";
    else
        xpath = "contacts";

    try {
        var oDs = Server.CreateObject("MSXML2.DOMDocument.5.0");
        oDs.async = false;
        oDs.resolveExternals = false;
        oDs.validateOnParse = false;

        var path = Server.MapPath("contacts.xml");
        if (oDs.load(path) == true) {
            var oContact = oDs.selectSingleNode(xpath);
            Response.ContentType = "text/xml";
            Response.Write(oContact.xml);
        }
    } catch (e) {
        Response.ContentType = "text/xml";
        Response.Write("<error>failed to create Contacts:" +"<desc>"+e.description+"</desc>" +"</error>"));
    }
%

To add the contact.asp file to the virtual directory

1. Create an empty text file in the directory you created to correspond to your
virtual directory (for example, c:\XMLOverHTTP, corresponding localhost/sxh). Name the new file contact.asp
2. Copy the resource listing above, and paste it into the XML file you just created. Save the file.

Now, build and run the XMLOverHTTP project. The result should be the output shown in the following topic.
When you build and run the **XMLOverHTTP** project, you should get the following output in a console window:

```xml
<contact>
    <name>John Doe</name>
    <phone>123-456-7890</phone>
</contact>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Program with DOM in Visual Basic

This tutorial is intended for Visual Basic programmers interested in writing XML applications using the DOM APIs as implemented in Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office. It includes a series of simple examples written in Microsoft Visual Basic and tested using Microsoft Visual Basic 6.0. The objective is to provide a quick and practical orientation, so that you can start to write your own XML applications.

Each demonstration is a self-contained application that you can build and run independently. Each demonstration provides complete source code, the expected output, any required resource files, and instructions for building and running the sample application.

These demonstrations are task-oriented, and are simplified for clarity. Therefore, their implementation is not always optimized and might not represent the best coding practice.

This tutorial consists of the following demonstrations.

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set Up My Visual Basic Project</strong></td>
<td>Discusses the requirements for using MSXML, and describes how to install the MSXML components.</td>
</tr>
<tr>
<td><strong>Instantiate an XML DOM Object (Visual Basic)</strong></td>
<td>Demonstrates two ways to instantiate an XML DOM object in Visual Basic.</td>
</tr>
<tr>
<td><strong>Load an XML File into a DOM Object (Visual Basic)</strong></td>
<td>Demonstrates how to create an XML DOM instance and load its content from an external XML data file.</td>
</tr>
<tr>
<td><strong>Serialize an XML DOM Object to a File (Visual Basic)</strong></td>
<td>Demonstrates how to serialize an XML DOM object in a text file.</td>
</tr>
<tr>
<td><strong>Perform XSL Transformations (Visual Basic)</strong></td>
<td>Demonstrates how to perform XSL Transformations (XSLT).</td>
</tr>
<tr>
<td><strong>Create an XML DOM Object</strong></td>
<td>Demonstrates how to create an XML DOM object programmatically, including processing instructions,</td>
</tr>
</tbody>
</table>
Dynamically *(Visual Basic)* comments, elements, attributes, CDATA sections and text nodes.

Query XML DOM Nodes *(Visual Basic)* Demonstrates how to query a DOM node or node-set using XPath expressions.

Make XML Requests Over HTTP *(Visual Basic)* Demonstrates how to perform client requests of XML data from a web server.

Validate an XML Document or Fragment Using Visual Basic Demonstrates how to validate an XML document and/or fragment against an XML schema.

Use the onReadyStateChange Property *(Visual Basic)* Demonstrates how to implement the onreadystatechange property, which was intended primarily for use by scripting clients, in Visual Basic.
Set Up My Visual Basic Project

To call XML DOM APIs from a Visual Basic application, first you need to make sure that MSXML is installed on your machine. See Get Ready to Work with MSXML for instructions on how to install MSXML.

Next, you need to create a Visual Basic project with a reference to MSXML. The following instructions describe this process.

To create a Visual Basic project

1. Launch Microsoft Visual Basic 6.0. Open a New Project dialog box from the File menu, if necessary.
2. Select the New tab from the New Project dialog box, and double click the Standard EXE icon.
3. If you want to change the name of the project and/or forms, you can do so in the project's Properties window. Otherwise, use the default settings.
4. Add a reference to MSXML 5.0. To do this, select References… from the Project menu, then check the box for Microsoft XML, v50 and click OK.
5. Save the project setting before adding code.

This empty Visual Basic project will serve as the starting point for each of the following demonstrations. First we'll look at how to instantiate an XML DOM object.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Instantiate an XML DOM Object (Visual Basic)

There are two ways to instantiate an XML DOM object in Visual Basic. One way is to create the object at compile time, as follows:

```vbnet
Dim objXMLDoc As New DOMDocument50
```

You can also create the object at run time, as follows:

```vbnet
Dim objXMLDoc
Set objXMLDoc = New DOMDocument50
```

In this demonstration we illustrate both ways of creating a XML DOM instance.

This tutorial uses or creates the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: InstantiateDOM.frm</td>
<td>Visual Basic source code.</td>
</tr>
<tr>
<td>Output</td>
<td>Output from the application.</td>
</tr>
</tbody>
</table>

To create the InstantiateDOM Visual Basic Project

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see [Set Up My Visual Basic Project](#).
2. Name the project **InstantiateDOMProj**.

Next, we'll add the [source code](#) for the **InstantiateDOM** project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
In this application, we create two XML DOM objects, and load an XML string into each object.

**Visual Basic Source File (InstantiateDOM.frm)**

```vbnet
Private Sub Form_Load()
    ' Instantiate an object at run time.
    Dim dom1
    Set dom1 = New DOMDocument50
    dom1.async = False
    dom1.resolveExternals = False
    dom1.loadXML "<a>A</a>

    ' Instantiate an object at compile time.
    Dim dom2 As New DOMDocument50
    dom2.async = False
    dom2.resolveExternals = False
    dom2.loadXML "<b>B</b>

    ' Display the content of both objects.
    MsgBox "dom1: " + dom1.xml + vbNewLine + "dom2: " + dom2.xml
End Sub
```

**To add InstantiateDOM.frm to the project**

- Copy the code listing above. Paste it into the Visual Basic code editor as the `form_load` subroutine, replacing any code fragments that are already there.

Next, run the project. The result should be the output shown in the following topic.
Output

When you run the **InstantiateDOM** project, you should get the following output:

- **dom1**: `<a>A</a>`
- **dom2**: `<b>B</b>`

The next tutorial shows how to [load an XML file into an XML DOM object](#).
Load an XML File into a DOM Object (Visual Basic)

To process XML data serialized as an XML file on the file system, you use the load method on an XML DOM object, supplying the file path as the input.

This tutorial uses or creates the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: LoadXMLFile.frm</td>
<td>Visual Basic source code.</td>
</tr>
<tr>
<td>Resource: test.xml</td>
<td>External XML file to be loaded into DOM.</td>
</tr>
<tr>
<td>Output</td>
<td>Output from the application.</td>
</tr>
</tbody>
</table>

To create the LoadXMLFile Visual Basic Project

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see Set Up My Visual Basic Project.
2. Name the project LoadXMLFileProj.

Next, we'll add the source code for the LoadXMLFile project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Source: LoadXMLFile.frm

The following code snippet loads an XML file from the current directory into an XML DOM object. In Visual Basic, the current directory is represented by App.Path property.

Visual Basic Source File (LoadXMLFile.frm)

Private Sub Form_Load()
    Dim doc As New DOMDocument50
    doc.async = False
    doc.validateOnParse = False
    doc.resolveExternals = False

    Path = App.Path + "\test.xml"
    doc.Load Path
    MsgBox "doc: " + doc.xml
End Sub

To add LoadXMLFile.frm to the project

- Copy the code listing above. Paste it into the Visual Basic code editor as the form_load subroutine, replacing any code fragments that are already there.

Next, we'll add the resource file, test.xml.
Resource: test.xml

The following simple XML data file is well-formed.

**XML File (test.xml)**

```xml
<?xml version="1.0"?>
<root>
  <node>one</node>
  <nodes>
    <node>1</node>
    <node>2</node>
  </nodes>
  <node>two</node>
</root>
```

**To add test.xml to the LoadXMLFile Project**

- Copy the XML data above, and paste it into a text file. Save the file as `test.xml` to the same directory where you saved `LoadXMLFileProj`.

Next, run the project. The result should be the output shown in the following topic.
Output

When you run the **LoadXMLFile** project, you should get the following output:

doc:
`<?xml version="1.0"?>
<root>
    <node>one</node>
    <nodes>
        <node>1</node>
        <node>2</node>
    </nodes>
    <node>two</node>
</root>`

Next, we'll [serialize an XML DOM object to a file](https://example.com).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Serialize an XML DOM Object to a File (Visual Basic)

You might often need to persist a DOM object so that you can reuse it later, or to save the XML object that is output from an XSLT transformation. To do this, you can call the save method on the DOM object.

This tutorial uses or creates the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source: SaveDOMToFile.frm</strong></td>
<td>Visual Basic source code.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Output from the application.</td>
</tr>
</tbody>
</table>

**To create the SaveDOMToFile Visual Basic Project**

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see [Set Up My Visual Basic Project](#).
2. Name the project **SaveDOMToFileProj**.

Next, we'll add the source code for this project.
Source: SaveDOM.frm

In this demonstration, we first create a DOM object and fill in the content by calling the loadXML method on the object. Then, we save the object to a file named "saved.xml".

**Visual Basic Source File (SaveDOM.frm)**

Private Sub Form_Load()
    Dim doc As New DOMDocument50
    doc.async = False
    doc.validateOnParse = False
    doc.resolveExternals = False

    doc.loadXML _
    "<?xml version='1.0'?>
    "<doc title='test'>
    "   <page num='1'>
    "      <para title='Saved at last'>
    "      This XML data is finally saved.
    "      </para>
    "   </page>
    "   <page num='2'>
    "      <para>
    "      This page is intentionally left blank.
    "      </para>
    "   </page>
    "</doc>
    Path = App.Path + "\saved.xml"
    doc.save Path
End Sub

**To add SaveDOM.frm to the project**

- Copy the code listing above. Paste it into the Visual Basic code editor as the `form_load` subroutine, replacing any code fragments that are already there.

Next, run the project. The result should be the output shown in the following topic.
Output

This project outputs the following XML file, `saved.xml`, in the current working directory.

```xml
<?xml version="1.0"?>
<doc title="test">
  <page num="1">
    <para title="Saved at last">
      This XML data is finally saved.
    </para>
  </page>
  <page num="2">
    <para>
      This page is intentionally left blank.
    </para>
  </page>
</doc>
```

The next topic demonstrates how to [perform XSL Transformations using Visual Basic](#).
Perform XSL Transformations (Visual Basic)

To perform XSL Transformations (XSLT) using DOM, you need to create at least two XML DOM objects: one for an XML document, and the other for the XSLT style sheet. You can then call the `transformNode` method on the DOM object for the XML data file, using the DOM object for the XSLT style sheet as input. The transformation outputs a string.

Alternatively, you can call the `transformNodeToObject` method on the DOM object for the XML data with the style sheet as the input. This transformation outputs another XML DOM object. This exercise demonstrates both ways to perform XSLT.

This tutorial uses or creates the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: XSLT.frm</td>
<td>Visual Basic source code.</td>
</tr>
<tr>
<td>Output</td>
<td>Output from the application.</td>
</tr>
</tbody>
</table>

**To create the XSLT Visual Basic Project**

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see [Set Up My Visual Basic Project](#).
2. Name the project XSLTProj.

Next, we'll add the [source code](#) for this project.
Private Function CreateDOM()
    Dim dom
    Set dom = New DOMDocument50
    dom.async = False
    dom.validateOnParse = False
    dom.resolveExternals = False
    Set CreateDOM = dom
End Function

Private Sub Form_Load()
    Dim doc, xsl, out, str

    Set doc = CreateDOM
    doc.Load App.Path + "\test.xml"

    Set xsl = CreateDOM
    xsl.Load App.Path + "\test.xsl"

    str = doc.transformNode(xsl)
    MsgBox "doc.transformNode: " + vbCrLf + str

    Set out = CreateDOM
    doc.transformNodeToObject xsl, out
    MsgBox "doc.transformNodeToObject:" + vbCrLf + out.xml
End Sub

To add XSLT.frm to the project

- Copy the code listing above. Paste it into the Visual Basic code editor as the form_load subroutine, replacing any code fragments that are already there.

Next, we'll add the resource files to the project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: test.xml and test.xsl

This project uses resource files, an XML data file and an XSLT style sheet. For the XML file, use test.xml from the LoadXMLFile project. The XSLT style sheet, test.xsl, is listing in this topic. This style sheet renames specified elements and leaves others unchanged.

XML Data File (test.xml)

```xml
<?xml version="1.0"?><root>
    <node>one</node>
    <nodes>
        <node>1</node>
        <node>2</node>
    </nodes>
    <node>two</node>
</root>
```

XSLT Style Sheet (test.xsl)

```xml
<?xml version="1.0"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<!-- Map "root" element to "network" element. -->
<xsl:template match="root">
    <xsl:element name="network">
        <xsl:apply-templates/>
    </xsl:element>
</xsl:template>

<!-- Keep any other elements as-is. -->
<xsl:template match="/ | @* | node()">
    <xsl:copy>
        <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

To add test.xml and test.xsl to the XSLT Project
1. Copy `test.xml` from above, and paste it into a text file. Save the file as `test.xml` to the same directory where you saved `XSLTProj`.
2. Copy the XSLT file above, and paste it into a text file. Save the file as `test.xsl` to the same directory where you saved `XSLTProj` and `test.xml`.

Next, run the project. The result should be the output shown in the following topic.
Output

When you run the **XSLT** project, you should get the following output in two consecutive dialog boxes:

`Doc.transformNode:`

```xml
<?xml version="1.0"?><network><node>one</node><nodes><node>1</node><node>2</node></nodes><node>two</node></network>
```

`doc.transformNodeToObject:`

```xml
<?xml version="1.0" encoding="UTF-16"?>
<network><node>one</node><nodes><node>1</node><node>2</node></nodes><node>two</node></network>
```

Next, we'll create an XML document dynamically.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Create an XML Document Dynamically (Visual Basic)

You already know that you can create an XML document as an XML file using any text editor. However, it is often necessary to create an XML document or fragment dynamically in an application. For example, an application might need to output results in an XML format for downstream processing, or legacy data might need to be reformatted in XML.

In this project, we will show you how to create an XML document in memory. Specifically, we will demonstrate how to add processing instructions, comments, elements, and text nodes. We will also show how to use extra white space text nodes to provide a structured format when the DOM object is serialized.

This tutorial uses or creates the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: dynamDOM.frm</td>
<td>Visual Basic source code.</td>
</tr>
<tr>
<td>Output</td>
<td>Output from the application.</td>
</tr>
</tbody>
</table>

To create the DynamDOM Visual Basic Project

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see Set Up My Visual Basic Project.
2. Name the project DynamDOMProj.

Next, we’ll add the source code for this project.
Source: DynamDOM.frm

This sample application creates a simple, but complete, XML DOM object, with <root> as the document element. This element contains three child elements: <node1>, <node2>, and <node3>. The first child element contains character data. The second child element contains a CDATA section. The last child element contains three empty child elements: <subnode1>, <subnode2>, and <subnode3>.

Visual Basic Source File (DynamDOM.frm)

Private Function CreateDOM()
    Dim dom
    Set dom = New DOMDocument50
    dom.async = False
    dom.validateOnParse = False
    dom.resolveExternals = False
    dom.preserveWhiteSpace = True
    Set CreateDOM = dom
End Function

Private Sub Form_Load()
    Dim dom, node, attr

    On Error GoTo ErrorHandler

    Set dom = CreateDOM

    ' Create a processing instruction targeted for xml.
    Set node = dom.createProcessingInstruction("xml", "version='1.0'
    dom.appendChild node
    Set node = Nothing

    ' Create a processing instruction targeted for xml-stylesheet.
    Set node = dom.createProcessingInstruction("xml-stylesheet", _
    "type='text/xml' href='test.xsl'")
    dom.appendChild node
    Set node = Nothing

    ' Create a comment for the document.
    Set node = dom.createComment("sample xml file created using XML
    dom.appendChild node
    Set node = Nothing

    ' Create the root element.
Dim root
Set root = dom.createElement("root")

' Create a "created" attribute for the root element and
' assign the "using dom" character data as the attribute value.
Set attr = dom.createAttribute("created")
attr.Value = "using dom"
root.setAttributeNode attr
Set attr = Nothing

' Add the root element to the DOM instance.
dom.appendChild root
' Insert a newline + tab.
root.appendChild dom.createTextNode(vbNewLine + vbTab)
' Create and add more nodes to the root element just created.
' Create a text element.
Set node = dom.createElement("node1")
node.Text = "some character data"
' Add text node to the root element.
root.appendChild node
Set node = Nothing
' Add a newline plus tab.
root.appendChild dom.createTextNode(vbNewLine + vbTab)

' Create an element to hold a CDATA section.
Set node = dom.createElement("node2")
Set cd = dom.createCDATASection("<some mark-up text>")
node.appendChild cd
Set cd = Nothing
dom.documentElement.appendChild node
Set node = Nothing
' Add a newline plus tab.
root.appendChild dom.createTextNode(vbNewLine + vbTab)

' Create an element to hold three empty subelements.
Set node = dom.createElement("node3")

' Create a document fragment to be added to node3.
Set frag = dom.createDocumentFragment
' Add a newline + tab + tab.
frag.appendChild dom.createTextNode(vbNewLine + vbTab + vbTab)
frag.appendChild dom.createElement("subNode1")
' Add a newline + tab + tab.
frag.appendChild dom.createTextNode(vbNewLine + vbTab + vbTab)
frag.appendChild dom.createElement("subNode2")
' Add a newline + tab + tab.
frag.appendChild dom.createTextNode(vbNewLine + vbTab + vbTab)
frag.appendChild dom.createElement("subNode3")
' Add a newline + tab.
frag.appendChild dom.createTextNode(vbNewLine + vbTab)
node.appendChild frag
Set frag = Nothing

root.appendChild node
' Add a newline.
root.appendChild dom.createTextNode(vbNewLine)
Set node = Nothing

' Save the XML document to a file.
dom.save App.Path + "\dynamDom.xml"
Set root = Nothing
Set dom = Nothing
Exit Sub

ErrorHandler:
    MsgBox Err.Description
End Sub

To add dynamDOM.frm to the project

- Copy the code listing above. Paste it into the Visual Basic code editor as the form_load subroutine, replacing any code fragments that are already there.

Next, run the project. The result should be the output shown in the following topic.
Output

This project outputs the following XML file, `dynamDOM.xml`, in the current working directory.

Notice that white spaces, rendered as text nodes, are used for indentation.

```xml
<?xml version="1.0"?>
<?xml-stylesheet type='text/xml' href='test.xsl'?>
<!--sample xml file created using XML DOM object.-->
<root created="using dom">
    <node1>some character data</node1>
    <node2><![CDATA[<some mark-up text>]]></node2>
    <node3>
        <subNode1/>
        <subNode2/>
        <subNode3/>
    </node3>
</root>
```

Next, we'll demonstrate how to query XML DOM nodes.
Query XML DOM Nodes (Visual Basic)

MSXML provides two DOM methods that allow you to query a node or a set of nodes in an XML DOM object: SelectNodes and SelectSingleNode. You specify the node or nodes of interest in an XPath expression, which is taken as the input to these methods.

In this demonstration, we assume that you have some understanding of XPath expressions.

This project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: queryNodes.frm</td>
<td>This Visual Basic source file selects both a single node and a node-set from a DOM object.</td>
</tr>
<tr>
<td>Output</td>
<td>When you build and run the queryNodes project, you should get this output.</td>
</tr>
</tbody>
</table>

To create the DynamDOMProj Visual Basic Project

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see Set Up My Visual Basic Project.
2. Name the project DynamDOMProj.

Next, we’ll add the source code for this project.

See Also

XPath Developer's Guide | XPath Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
This example application calls both the `selectSingleNode` method and the `selectNodes` method on an XML DOM object loaded from the `stocks.xml` file. The same XPath expression is input to both methods:

```
//stock[1]/*
```

This expression specifies all the child elements of the first `<stock>` element in the XML document. In MSXML, the `selectSingleNode` method returns the first element of the resultant node-set, and the `selectNodes` method returns all the elements in the node-set.

Programmatically, this source code performs the following steps:

1. Creates an instance of an XML DOM object (`oXMLDom`).
2. Calls the `load` method on `oXMLDom` to load XML data from a file (`stocks.xml`).
3. Calls the `selectSingleNode` method on `oXMLDom` to select a DOM node according to the input XPath expression (`//stock[1]/*`). Then displays the resulting node in the console if the query is successful.
4. Calls the `selectNodes` method on `oXMLDom` to select a DOM node-set according to the input XPath expression (`"//stock[1]/*"`). Then displays the resulting nodes in the console if the query is successful.

**Visual Basic Source File (queryNodes.frm)**

```vb
Private Sub Form_Load()
    ' Output string:
    Dim strout As String
    strout = ""

    ' Load an xml document into a DOM instance.
    Dim oXMLDom As New DOMDocument50
    oXMLDom.async = False
    oXMLDom.validateOnParse = False
    oXMLDom.resolveExternals = False
    oXMLDom.preserveWhiteSpace = True
    If oXMLDom.Load(App.Path + "\stocks.xml") = False Then
        MsgBox "Failed to load xml data from file."
```
Exit Sub
End If

' Query a single node.
Dim oNode As IXMLDOMNode
Set oNode = oXMLDom.selectSingleNode("//stock[1]/*")
If oNode Is Nothing Then GoTo MoreNodes

strout = strout _
  + "Result from selectSingleNode" + vbNewLine _
  + "Node, <" + oNode.nodeName + ">: " + vbNewLine _
  + vbTab + oNode.xml + vbNewLine + vbNewLine

MoreNodes:
' Query a node-set.
Dim oNodes As IXMLDOMNodeList
Set oNodes = oXMLDom.selectNodes("//stock[1]/*")

strout = strout _
  + "Results from selectNodes:" + vbNewLine
For i = 0 To oNodes.length - 1
  Set oNode = oNodes.nextNode
  If Not (oNode Is Nothing) Then
    sName = oNode.nodeName
    sData = oNode.xml
    strout = strout _
      + "Node (" + CStr(i) + "), <" + sName + ">:" _
      + vbNewLine + vbTab + sData + vbNewLine
  End If
Next
MsgBox strout

End Sub

To add queryNodes.frm to the project

- Copy the code listing above. Paste it into the Visual Basic code editor as the form_load subroutine, replacing any code fragments that are already there.

Next, we'll add the resource files to the queryNodes project.
Resource: stocks.xml

XML File (stocks.xml)

```xml
<?xml version="1.0"?><!DOCTYPE portfolio SYSTEM "stocks.dtd"><portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nasdaq">
    <name>new</name>
    <symbol>zzzz</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

To add stocks.xml to the queryNodes project

1. Create a new text file in the project's main working directory using Notepad or another text editor. Name the file `stocks.xml`.
2. Copy `stocks.xml`, above, and paste it into the empty text file you just created.

Next, run the project. The result should be the output shown in the following topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Output

When you run the queryNodes project, you should get the following output:

Result from selectSingleNode:
Node, <name>:
   <name>new</name>

Results from selectNodes:
Node (0), <name>:
   <name>new</name>
Node (1), <symbol>:
   <symbol>zzzz</symbol>
Node (2), <price>:
   <price xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="number">20.313</price>

Notice that the output above confirms that the selectSingleNode method picks out only the first node from the node set as specified by the XPath expression (//stock[1]/*). The selectNodes method returns the entire set of nodes specified by this XPath expression.

Next, we'll demonstrate how to make XML requests over HTTP.
Make XML Requests Over HTTP (Visual Basic)

This tutorial demonstrates how to request an XML data file from a web server using HTTP. The Visual Basic application is a client making the request, and localhost is a local Web server. An XML data file containing contact information (contacts.xml) is stored on the server. The server fulfills the request through an ASP page (contact.asp), which is accessible from the localhost/sxh virtual directory. The request is made in an asynchronous call. We create a class module (as the handler to the onreadystatechange event) to process the response when it becomes available.

This project uses or creates the following files and directories.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source: XMLOverHTTP.cpp</td>
<td>This Visual Basic source file retrieves and displays XML data from a Web server.</td>
</tr>
<tr>
<td>Class Module: myHttpRequestHandlers</td>
<td>This class module enables asynchronous requests.</td>
</tr>
<tr>
<td>Virtual Directory: sxh</td>
<td>A virtual directory, with a corresponding directory on your drive.</td>
</tr>
<tr>
<td>Resource: contacts.xml</td>
<td>An XML data file containing contact information</td>
</tr>
<tr>
<td>ASP Page: contact.asp</td>
<td>An ASP page that queries a DOM object loaded from the contacts.xml file.</td>
</tr>
<tr>
<td>Output</td>
<td>You should this output get when you run the XMLOverHTTP project.</td>
</tr>
</tbody>
</table>

To create the XMLOverHTTP Visual Basic Project

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see Set Up My Visual Basic Project.
2. Name the project **XMLOverHTTPProj**.

Next, we’ll add the [source code](#) for this project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
This Visual Basic code example consists primarily of two subroutines: MakeRequest and ProcessResponse.

In MakeRequest, we construct a request URL based on user input, connect to the specified server, and issue the request. If the request is synchronous, we then call ProcessResponse. Otherwise, we exit the routine and the onreadystatechange event handler calls ProcessResponse when the response becomes available. In the ProcessResponse routine, we extract the requested information from the server and display it in a text box on the form.

Visual Basic Source File (XMLOverHTTP.frm)

```vbnet
Public oHttpReq As XMLHttpRequest
Public oXMLDoc As DOMDocument

Private Sub Command1_Click()
    MakeRequest (True)
End Sub

Private Sub MakeRequest(ByVal isAsync As Boolean)
    Set oHttpReq = New XMLHttpRequest
    Dim xhrHandler As myHttpRequestHandlers

    If isAsync = True Then
        Set xhrHandler = New myHttpRequestHandlers

        ' Set a readyStateChange handler.
        oHttpReq.onreadystatechange = xhrHandler
    End If

    ' Construct the URL from user input.
    url = Text1.Text
    If Text2.Text <> "" Then
        url = url + "?SearchID=" + Text2.Text
    End If
    ' Clear the display.
    Text3.Text = ""

    ' Open a connection and set up a request to the server.
    oHttpReq.open "GET", url, isAsync
```
' Send the request to the server.
oHttpReq.send
' In a synchronous call, we must call ProcessResponse. In an
' asynchronous call, the OnReadyStateChange handler calls
' ProcessResponse.
If isAsync = False Then
ProcessResponse
End If
End Sub
Public Sub ProcessResponse()
' Receive the response from the server.
Set oXMLDoc = oHttpReq.responseXML
' Display the server response to the user.
Set oNode = oXMLDoc.selectSingleNode("//phone")
If oNode Is Nothing Then
Text3.Text = "Requested information not found."
Else
Text3.Text = oNode.Text
End If
End Sub
Private Sub Form_Load()
Text3.Text = ""
Text1.Text = "http://localhost/sxh/contact.asp"
Text2.Text = "John Doe"
End Sub

To add XMLOverHTTP.frm to the project and set the controls
1. Add three TextBox controls to the form. The Text1 control will be used for
the URL of the contact look-up service. Text2 will allow a user to specify
the name of a contact whose telephone number will be queried. Text3 is
used to display the query result.
2. Add a label for each text box and change the captions to "Service:",
"Name:", and "Phone:", respectively.
3. Add a command control (Command1) that will start a request when a user
clicks on it.
4. Arrange the controls in the form as shown below:

5. Copy the code listing above. Paste it into the Visual Basic code editor as the
form_load subroutine, replacing any code fragments that are already there.


Next, we'll add a class module as the handler to the onreadystatechange event of the IXMLHTTPRequest object.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Class Module: myHttpRequestHandlers

To enable asynchronous requests, we add a class module as an event handler for the onreadystatechange events of an IXMLHTTPRequest object.

To add the class module

1. Select Add Class Module from the Project menu in the Visual Basic editor, and click Open to open a new class module.
2. Change the name of the new class module from "Class1" to "myHttpRequestHandlers".
3. Paste the following code into the class module:

   Option Explicit
   Sub OnReadyStateChange()
       If form1.oHttpReq.readyState = 4 Then
           form1.ProcessResponse
       End If
   End Sub

There are other ways to set up the event handler. For more information, see Use the OnReadyStateChange Property in Visual Basic.

Next, we'll set up a virtual directory. You must complete this step before you build and run the application.
Virtual Directory: sxh

For this tutorial, we use an Internet Information Services (IIS) Web server hosted on the local host. Make sure IIS is installed on your machine before you continue.

To set up a virtual directory

1. Create a directory, such as c:\XMLOverHTTP, to map to this virtual directory.
2. Open IIS.
3. Navigate to Web Sites -> Default Web Site, and right-click.
4. Select New and click Virtual Directory... to open the Virtual Directory Creation Wizard.
5. Click Next.
6. Enter "sxh" under Alias. Then click Next.
7. Click Browse... and navigate to the directory you created in Step 1. Click Next.
8. Accept the default access permissions by clicking Next.
9. Click Finish.

Next, we'll add the resource file, contacts.xml, to the directory you created in Step 1.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: contacts.xml

The following XML data file contains two contact entries, each with a name and phone number.

XML Data File (contacts.xml)

```xml
<?xml version="1.0"?>
<contacts>
  <contact>
    <name>John Doe</name>
    <phone>123-456-7890</phone>
  </contact>
  <contact>
    <name>Jane Doe</name>
    <phone>123-456-7891</phone>
  </contact>
</contacts>
```

To add contacts.xml to the XMLOverHTTP project

1. Create an empty text file in the directory you created to correspond to your virtual directory (for example, c:\XMLOverHTTP, corresponding localhost/sxh). Name the text file contacts.xml.
2. Copy the resource listing above, and paste it into the XML file you just created.

Next, we'll create an ASP page to handle the request.
ASP Page: contact.asp

The ASP page makes queries against a DOM object loaded from the contacts.xml file. For example, if the request comes with the a query string "SearchID=John Doe", the ASP page returns the contact information for John Doe. Otherwise, it returns the contact information of all listed persons. If the specified person is not a listed entry, the ASP returns an error to the client.

ASP page (contact.asp)

```<%@language="javascript"%>
<%
    var xpath;
    var sName=Request.QueryString("SearchID")();
    if (sName)
        xpath = "//contact[name='" + sName + "]";
    else
        xpath = "contacts";

    try {
        var oDs = Server.CreateObject("MSXML2.DOMDocument.5.0");
        oDs.async = false;
        oDs.resolveExternals = false;
        oDs.validateOnParse = false;

        var path = Server.MapPath("contacts.xml");
        if (oDs.load(path) == true) {
            var oContact = oDs.selectSingleNode(xpath);
            Response.ContentType = "text/xml";
            Response.Write(oContact.xml);
        }
    }
    catch (e) {
        Response.ContentType = "text/xml";
        Response.Write("<error>failed to create Contacts:");
        +"<desc">+e.description+"</desc>" +"</error>"};
%
```

To add contact.asp to the virtual directory

1. Create an empty text file in the directory you created to correspond to your
virtual directory (for example, `c:\XMLOverHTTP`, corresponding to `localhost/sxh`). Name the new file `contact.asp`.

2. Copy the resource listing above, and paste it into the XML file you just created. Save the file.

Next, run the sample application. Your output should be the same as that shown in the next topic.
Output

When you build and run the **XMLOverHTTP** project, you should get "123-456-7890" listed for the **Phone:** entry when the text in the **Name:** field is "John Doe". If you change "John Doe" to "Jane Doe" and then click the **Get** button, you should get "123-456-7891" as the result.
Validate an XML Document or Fragment (Visual Basic)

In MSXML 5.0 and later, there are two ways to validate XML data against a set of XML Schema definitions. One uses the `validate` method on an XML DOM instance to validate the entire DOM object. The other uses the `validateNode(node)` method on a DOM instance to validate a DOM fragment. A DOM fragment is a sub-tree rooted at the specified node.

The example in this tutorial illustrates both of these methods. This project uses the following files.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source: validateDOM.frm</strong></td>
<td>Validates an XML file against an XSD schema.</td>
</tr>
<tr>
<td><strong>Resource: books.xml and books.xsd</strong></td>
<td>An XML data file and an XSD schema file. When you build and run the <code>validateDOM</code> project, you should get this output in a console window.</td>
</tr>
</tbody>
</table>

Output

To create the validateDOM Visual Basic Project

1. Create a Visual Basic project and set a reference to MSXML 5.0. For detailed instructions on how to do this, see [Set Up My Visual Basic Project](link).
2. Name the project `validateDOMProj`.

Next, we'll add the source code for this project.
Source: validateDOM.frm

The source code performs the following basic steps:

1. Creates a DOM instance (oXMLDoc) to hold the XML data.
2. Creates a DOM instance (oXSDDoc) to hold the XML Schema definition.
3. Creates an IXMLSchemaCollection or IXMLSchemaCollection2 object (oSCache). This object is also called a schema cache. The application then adds the XML Schema definition (oXSDDoc) to the oSCache.
4. Associates oSCache with the schemas property of the DOM object for the XML data (oXMLDoc).
5. Calls the following validation methods on the DOM object for XML data (oXMLDoc):
   - Calls the validate method on oXMLDoc to validate the data set as a whole
   - Calls the validateNode(oNode) method on oXMLDoc to validate a node object (oNode) selected from oXMLDoc.
6. Checks the error returned from validate method and/or the validateNode(oNode) method, to determine if the specified XML data set is valid against the given XML Schema definition.

Visual Basic Source File (validateDOM.frm)

Private Sub Form_Load()
   ' Output string:
   Dim strout As String
   strout = ""

   ' Load an XML document into a DOM instance.
   Dim oXMLDoc As DOMDocument50
   Set oXMLDoc = DOMFromFile(App.path + "\books.xml")
   If oXMLDoc Is Nothing Then
      Exit Sub
   End If

   ' Load the schema for the xml document.
   Dim oXSDDoc As DOMDocument50
Set oXSDDoc = DOMFromFile(App.path + "\books.xsd")
If oXSDDoc Is Nothing Then
    Exit Sub
End If

' Create a schema cache instance.
Dim oSCache As New XMLSchemaCache50

' Add the just-loaded schema definition to the schema collection
oSCache.Add "urn:books", oXSDDoc

' Assign the schema to the XML document's schema collection.
Set oXMLDoc.schemas = oSCache

' Validate the entire DOM.
strout = strout _
        + "Validating DOM..." + vbNewLine
Dim oError As IXMLDMParseError
Set oError = oXMLDoc.Validate
If oError.errorCode <> 0 Then
    strout = strout + vbTab _
            + "XMLDoc is not valid because " _
            + vbNewLine + oError.reason + vbNewLine
Else
    strout = strout _
            + vbTab + "XMLDoc is validated:" + vbNewLine _
            + oXMLDoc.xml + vbNewLine
End If

Dim oNodes As IXMLDOMNodeList
' Validate all "/books" nodes, node by node.
strout = strout _
        + "Validating all book nodes, "/book\", "_" _
        + "one by one ..." + vbNewLine
Set oNodes = oXMLDoc.selectNodes("//book")
strout = strout + ValidateNodes(oXMLDoc, oNodes)

' Validate all children of //books nodes, node by node.
strout = strout _
        + "Validating all children of all book nodes, //book/*, " _
        + "one by one ..." + vbNewLine
Set oNodes = oXMLDoc.selectNodes("//book/*")
strout = strout + ValidateNodes(oXMLDoc, oNodes)

MsgBox strout
End Sub

Private Function DOMFromFile(ByVal path As String)
    If path = "" Then
        Set DOMFromFile = Nothing
    End If
End Function
Exit Function
End If

Dim dom As New DOMDocument50
dom.async = False
dom.validateOnParse = False
dom.resolveExternals = False
dom.preserveWhiteSpace = True
If dom.Load(path) = False Then
    MsgBox "Can't create DOM from " + path
    Set DOMFromFile = Nothing
    Exit Function
End If
Set DOMFromFile = dom
End Function

Private Function ValidateNodes(oXMLDoc As DOMDocument50, _
oNodes As IXMLDOMNodeList) As String
If oXMLDoc Is Nothing Then
    ValidateNodes = "Error in ValidateNodes(): Invalid oXMLDoc"
    Exit Function
End If

If oNodes Is Nothing Then
    ValidateNodes = "Error in ValidateNodes(): Invalid oNodes"
    Exit Function
End If

Dim oNode As IXMLDOMNode
Dim oError As IXMLDOMParseError
Dim strout As String

For i = 0 To oNodes.Length - 1
    Set oNode = oNodes.NextNode
    If Not (oNode Is Nothing) Then
        Set oError = oXMLDoc.ValidateNode(oNode)
        If oError.errorCode = 0 Then
            strout = strout + vbTab _
                + "<" + oNode.nodeName + ""> (" + CStr(i) + ") is a valid node " + vbNewLine
        Else
            strout = strout + vbTab _
                + "<" + oNode.nodeName + ""> (" + CStr(i) + ") " _
                + "is not valid because" + vbNewLine _
                + oError.reason + vbNewLine
        End If
    End If
Next
ValidateNodes = strout
To add validateDOM.frm to the project

- Copy the code listing above. Paste it into the Visual Basic code editor as the **form_load** subroutine, replacing any code fragments that are already there.

Next, we'll add the resource files.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Resource: books.xml and books.xsd

This example uses two resources files. One is an XML data file (books.xml) and the other is an XML Schema definition file (books.xsd). According to books.xsd, the second <book> element in books.xml is missing the required <pub_date> child element. Therefore, when we attempt to validate the XML file against the given schema, we get a validation error.

XML File (books.xml)

```xml
<?xml version="1.0"?>
<x:books xmlns:x="urn:books">
  <book id="bk001">
    <author>Writer</author>
    <title>The First Book</title>
    <genre>Fiction</genre>
    <price>44.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>An amazing story of nothing.</review>
  </book>
  <book id="bk002">
    <author>Poet</author>
    <title>The Poet's First Poem</title>
    <genre>Poem</genre>
    <price>24.95</price>
    <review>Least poetic poems.</review>
  </book>
</x:books>
```

XSD File (books.xsd)

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:books"
  xmlns:bks="urn:books">
  <xsd:element name="books" type="bks:BooksForm"/>
  <xsd:complexType name="BooksForm">
    <xsd:sequence>
      <xsd:element name="book" type="bks:BookForm" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="BookForm">
    <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="genre" type="xsd:string"/>
        <xsd:element name="price" type="xsd:float"/>
        <xsd:element name="pub_date" type="xsd:date"/>
        <xsd:element name="review" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>

To add books.xml and books.xsd to the project

1. Create a new text file in the project's main working directory using Notepad or another text editor. Name the new file books.xml.
2. Copy books.xml, above, and paste it into the empty text file you just created.
3. Create a new text file in the project's main working directory using Notepad or another text editor. Name the new file books.xsd.
4. Copy books.xsd, above, and paste it into the empty text file you just created.

Next, run the project. The result should be the output shown in the following topic.
Output

When you run the **validateDOM** project, you should get the following output:

Validating DOM...
  XMLDoc is not valid because
Element content is invalid according to the DTD/Schema.
Expecting: pub_date.

Validating all //book nodes, one by one ...
  <book> (0) is a valid node
  <book> (1) is not valid because
Element content is invalid according to the DTD/Schema.
Expecting: pub_date.

Validating all children of all book nodes, //book/*, one by one ...
  <author> (0) is a valid node
  <title> (1) is a valid node
  <genre> (2) is a valid node
  <price> (3) is a valid node
  <pub_date> (4) is a valid node
  <review> (5) is a valid node
  <author> (6) is a valid node
  <title> (7) is a valid node
  <genre> (8) is a valid node
  <price> (9) is a valid node
  <review> (10) is a valid node

The next topic describes how to use the **onReadyStateChange** property.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Use the `onReadyStateChanged` Property (Visual Basic)

This topic discusses the implementation details necessary to use `onreadystatechange` notification in Microsoft Visual Basic applications.
Background Information for onReadyStateChange events

The onreadystatechange callback function was not implemented as a COM automation event in the IXMLHTTPRequest and IServerXMLHTTPRequest components. This is because these components are heavily used in scripting environments, many of which do not support COM events. The onreadystatechange callback function was intended to be easy to use when working with scripting clients such as VBScript and JScript.

Because the onreadystatechange property was not implemented through COM-based automation events, Visual Basic (and C/C++) applications need to implement this callback functionality differently.
Using onReadyStateChange in Visual Basic applications

In Visual Basic, you can use any of the following approaches to design applications that support onreadystatechange events.

- Use a timer control to poll the readyState property. When the value of the readyState property indicates that the data is ready, turn the timer off.
- Use a DomDocument object to load the XML and handle the state using the WithEvents keyword.

  Note If you are using the IXMLHTTPRequest and IServerXMLHTTPRequest components to first post your XML data to a Web server, this option will not work for you.

- Create a wrapper class, and create a procedure to handle the event within the class module. Set the procedure to the default, and bind the class to the onreadystatechange event to either the IXMLHTTPRequest or IServerXMLHTTPRequest component, depending on which component you are using with your application.

The following sample application demonstrates each of these three approaches.

To use OnReadyStateChange in a Visual Basic application

1. Open Microsoft® Visual Basic® 6.0. In the New Project dialog box, double-click Standard EXE.
2. On the Project menu, click References.
3. In the Available References list, select Microsoft XML,v5.0, and then click OK.
4. Add four command buttons to Form1 and set the caption of each button as follows:

<table>
<thead>
<tr>
<th>Control</th>
<th>Caption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command1</td>
<td>Fail</td>
</tr>
<tr>
<td>Command2</td>
<td>Polling using Timer</td>
</tr>
<tr>
<td>Command3</td>
<td>Using Class Wrapper</td>
</tr>
</tbody>
</table>
5. Add a timer control to Form1.
6. Copy and paste the following code into Form1.

Option Explicit

Public XMLHttpRequest As MSXML2.XMLHTTP50
Public WithEvents XMLDom As MSXML2.DOMDocument50

Private Function FunctionReadyStateChange()
    Debug.Print XMLHttpRequest.readyState
End Function

Private Sub Command1_Click()
    FailedOnReadyState
End Sub

Private Sub Command2_Click()
    TimerResolution
End Sub

Private Sub Command3_Click()
    ClassResolution
End Sub

Private Sub Command4_Click()
    DOMResolution
End Sub

Private Sub FailedOnReadyState()
On Error GoTo FailedState
    If Not XMLHttpRequest Is Nothing Then Set XMLHttpRequest = 

        Set XMLHttpRequest = New MSXML2.XMLHTTP50

    ' Assign the wrapper class object to onreadystatechange.
    XMLHttpRequest.onreadystatechange = FunctionReadyStateChange

    ' Get some stuff asynchronously.
    XMLHttpRequest.open "GET", "http://localhost/test.xml", True
    XMLHttpRequest.send

    Exit Sub

FailedState:
    MsgBox Err.Number & ": " & Err.Description

End Sub
Private Sub TimerResolution()
    If Not XMLHttpRequest Is Nothing Then Set XMLHttpRequest = Nothing
    Timer1.Interval = 1
    ' Get some stuff asynchronously.
    XMLHttpRequest.open "GET", "http://localhost/test.xml", True
    XMLHttpRequest.send
End Sub

Private Sub ClassResolution()
    If Not XMLHttpRequest Is Nothing Then Set XMLHttpRequest = Nothing
    Dim MyOnReadyStateWrapper As MyReadyStateHandler
    Set XMLHttpRequest = New MSXML2.XMLHTTP50
    ' Create an instance of the wrapper class.
    Set MyOnReadyStateWrapper = New MyReadyStateHandler
    ' Assign the wrapper class object to onreadystatechange.
    XMLHttpRequest.OnReadyStateChange = MyOnReadyStateWrapper
    ' Get some stuff asynchronously.
    XMLHttpRequest.open "GET", "http://localhost/test.xml", True
    XMLHttpRequest.send
End Sub

Private Sub DOMResolution()
    If Not XMLHttpRequest Is Nothing Then Set XMLHttpRequest = Nothing
    If Not XMLDom Is Nothing Then Set XMLDom = Nothing
    ' Get some stuff asynchronously.
    Set XMLDom = New MSXML2.DOMDocument50
    XMLDom.async = True
    XMLDom.Load "http://localhost/test.xml"
End Sub

Private Sub Timer1_Timer()
    Debug.Print XMLHttpRequest.readyState
    If XMLHttpRequest.readyState = 4 Then
        MsgBox "Done"
        Timer1.Interval = 0
    End If
End Sub
Private Sub XMLDom_onreadystatechange()
    Debug.Print XMLDom.readyState
    If XMLDom.readyState = 4 Then
        MsgBox "Done"
    End If
End Sub

7. From the **Project** menu, click **Add Class Module**.
8. Change the name of the new class module from "Class1" to "MyReadyStateHandler"
9. Paste the following code into the class module:

```
Option Explicit

Sub OnReadyStateChange()
    Debug.Print Form1.XMLHttpRequest.readyState
    If Form1.XMLHttpRequest.readyState = 4 Then
        MsgBox "Done"
    End If
End Sub
```

10. In the sample code added in the previous step, highlight the procedure name "OnReadyStateChange" by selecting it in the Code window.
11. From the **Tools** menu, click **Procedure Attributes**.

    In the **Procedure Attributes** dialog, the **Name** combo box should show "OnReadyStateChange."

12. Click **Advanced**.
13. In **Procedure ID**, select "(Default)" from the available options.
14. Click **OK**.
15. Save the class module (**MyReadyStateHandler.cls**) to file.
16. Open Notepad and paste the following XML into it

```
<?xml version="1.0"?>
<Root>
    <Testing>This is to test the onreadystatechange event on the XMLHttpRequest or DOMDocument</Testing>
    <Testing>This is due to the event not being declared in the type library</Testing>
</Root>
```

17. Save the file as **test.xml** to your IIS localhost directory. For example, this folder might be C:\inetpub\wwwroot for a typical default installation of IIS with Windows 2000.
18. In Visual Basic, from the **Run** menu, click **Start** to run the application.
19. Try the following command options to observe the different approaches to
using the onreadystatechange event within Visual Basic.
1. To force a ready state failure, click \textbf{Fail}.
2. To view the polling resolution, click \textbf{Polling using Timer}.
3. To view the wrapper class solution, click \textbf{Using Class Wrapper}.
4. To view the \texttt{DOMDocument} approach, click \textbf{Using \texttt{DOMDocument}}.

20. For each of the code paths in the previous step, you can place brake-points at various places to step through the code.

\textbf{Remarks}

For step 19a, the following error message should appear indicating the failure:

\texttt{424: Object required}

For steps 19b, 19c and 19d, return values for \texttt{onreadystatechange} should appear in the Visual Basic debug window while the document loads. When the document completes loading, a message box saying "Done" should appear.
# Program with DOM in JScript

The following tutorials demonstrate common DOM tasks using Microsoft JScript.

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use DOMDocument</td>
<td>Demonstrates how to use the DOMDocument object to navigate, query, and modify the content and structure of an XML document.</td>
</tr>
<tr>
<td>Use IXMLDOMNode</td>
<td>Demonstrates how to represent XML document components as IXMLDOMNode objects (nodes).</td>
</tr>
<tr>
<td>Use IXMLDOMNodeList</td>
<td>Demonstrates how to navigate and gather information about a list of nodes.</td>
</tr>
<tr>
<td>Use IXMLDOMNodeNamedNodeMap</td>
<td>Demonstrates how to navigate, manipulate, and gather information about a named collection of nodes.</td>
</tr>
</tbody>
</table>
Use DOMDocument

The DOMDocument object is the foundation of the XML Document Object Model (DOM). It exposes properties and methods that allow you to navigate, query, and modify the content and structure of an XML document.

The following topics describe ways to use the DOMDocument object.

- Creating a DOMDocument Object
- Loading and Saving XML
- Getting and Setting Parse Flags
- Gathering Document Information
- Accessing the Document Tree
- Creating New Nodes
Creating a DOMDocument Object

Before you can parse or create XML documents using the Document Object Model (DOM), you must create a DOMDocument object. After you have created that object, you can use it to load, parse, navigate, and manipulate XML files.

**To create a new, empty DOMDocument object**

Use the following Microsoft® JScript® code.

```javascript
var mydoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
```

Use the following Microsoft Visual Basic® Scripting Edition (VBScript) code.

```vbnet
Dim xmldoc
Set xmldoc = CreateObject("Msxml2.DOMDocument.5.0")
```

Use the following Microsoft Visual Basic code.

```vbnet
Set xmldoc = New DOMDocument50
```

**See Also**

IXMLDOMDocument/DOMDocument
Loading and Saving XML

Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office provides two methods for loading XML documents into the Document Object Model (DOM) and one method and one property for serializing that object model back into an XML stream. The load and loadXML methods parse documents into a DOM. The save method and xml property convert the DOM back into an XML document. The load and save methods are used to retrieve information from outside a script, while the loadXML method and xml property are generally used inside scripts, parsing and returning XML strings found inside.

Use the load method to load an XML file by path name, URL, or from a Microsoft Internet Information Services (IIS) Request object.

The following example loads an XML file by URL.

XMLDoc.load("http://www.examples.microsoft.com/reports.xml")

The document object, XMLDoc, will load the reports.xml document, parse it, and, depending on the content of the document, create an object tree representing the document or report an error.

By default, the loading and parsing of an XML file occurs asynchronously. To load the XML file synchronously, set the async property to False. This forces the loading of the entire XML document before processing continues. To abort an asynchronous download, call the abort method on the document object.

You can also load an XML file as a string using the loadXML method.

XMLDoc.loadXML("<customer><first_name>Joe</first_name> <last_name>Smith</last_name></customer>")

To save a parsed XML document to a file or another object, use the save method. The save method takes either a file name as a string, an Active Server Pages (ASP) Response object, a DOMDocument object, or any custom object that supports persistence. The following Microsoft Visual Basic® Scripting Edition (VBScript) example is part of an ASP page that saves the XML document to a file.
<% dim xmldoc
    set xmldoc = Server.CreateObject("Msxml2.DOMDocument.5.0")
    xmldoc.async = false
    xmldoc.load(Request)
    xmldoc.save(Server.MapPath("sample.xml"))
%>

The following example is in Microsoft JScript®.

<% var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmldoc.async = false;
    xmldoc.load(Request);
    xmldoc.save(Server.MapPath("sample.xml"));
%>

You can use the xml property of DOMDocument to retrieve the XML document as a string. This allows you to retrieve a Unicode string representation of the XML document model. An ASP script that uses the XML parser to process information sent in the Request object is shown in VBScript.

<% dim xmldoc
    set xmldoc = Server.CreateObject("Msxml2.DOMDocument.5.0")
    xmldoc.async = false
    xmldoc.load(Request)
    xmlAsString=xmldoc.xml
%>

The following example is in JScript.

<% var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmldoc.async = false;
    xmldoc.load(Request);
    xmlAsString=xmldoc.xml;
%>

**See Also**

async Property | load Method | loadXML Method | save Method
Getting and Setting Parse Flags

Although Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office is a single library, there are times when you may want to parse an XML document in different ways. The XML recommendation allows parsers to validate or not validate documents, and allows nonvalidating parsers to skip the retrieval of external resources. In addition, you may want the parser to process the document while your code continues or remove extraneous white space from your documents.

The DOMDocument object exposes four properties that allow the user to change parsing behavior at run time.

- **async**
- **validateOnParse**
- **resolveExternals**
- **preserveWhiteSpace**

Each of these properties takes and returns a Boolean value. The default value for async, validateOnParse, and resolveExternals is True. The default value for preserveWhiteSpace, which specifies whether to honor the xml:space attribute in XML documents, is False.

The following Microsoft Visual Basic® Scripting Edition (VBScript) sample code loads and parses an XML file synchronously; however, the XML document is not validated as it is parsed.

```vbs
dim XMLDoc
set XMLDoc = Server.CreateObject("Msxml2.DOMDocument.5.0")
XMLDoc.async = false
XMLDoc.validateOnParse = false
XMLDoc.load("sample.xml")
```

The following example is in Microsoft JScript®.

```js
var XMLDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
XMLDoc.async = false;
XMLDoc.validateOnParse = false;
XMLDoc.load("sample.xml");
```
**Note**  Although not explicitly indicated in this code, by default, the parser will resolve externals and will not honor the value of the `xml:space` attribute.

Asynchronous parsing allows your application to do other things, such as display a progress bar, while the file is being parsed. After loading the XML document but before manipulating or reading it, be sure to check the `readyState` property to ensure that the document has been fully loaded and that the Document Object Model (DOM) is available.

**See Also**

[readyState Property (DOMDocument)](#)
Gathering Document Information

These properties can be used to get the following information about the document parsing process.

- **doctype**
  The document type definition (DTD) accompanying the document.

- **implementation**
  The implementation for the document.

- **parseError**
  The last error to occur while parsing.

- **readyState**
  Information about the state of the XML document.

- **url**
  The URL, if any, of the XML file being loaded and parsed.

For more information about error reporting, see [IXMLDOMParseError](IXMLDOMParseError).

The `readyState` property is critical for programs that use Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office asynchronously to improve performance. When loading XML documents asynchronously, your programs may need to check the state of the parsing process. MSXML provides four states, numbered one through four: loading, loaded, interactive (parsing), and completed. The following Microsoft JScript® and HTML example demonstrates the sequence of ready states when asynchronously loading a document.

```javascript
<script>
var xmlDoc;
function Load()
{
```
```
xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.onreadystatechange = CheckState;
xmlDoc.load(URL.value);
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
}

function CheckState()
{
  var state = xmlDoc.readyState;
  RESULTS.innerHTML += "readyState = " + state + "<BR>"
  if (state == 4)
  {
    var err = xmlDoc.parseError;
    if (err.errorCode != 0)
      RESULTS.innerHTML += err.reason + "<BR>"
    else
      RESULTS.innerHTML += "success" + "<BR>"
  }
}
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Accessing the Document Tree

You can access the tree for an XML document either by beginning at the root element and moving from node to node or by querying for a specific node or nodes. You can navigate to the root element of the document by using the documentElement property. This property returns the root element as an IXMLDOMNode.

The following books.xml file is the same as the file used in the document map.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<!--catalog last updated 2000-11-01-->
<catalog xmlns="http://www.example.com/catalog/">
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description><![CDATA[An in-depth look at creating applications]]></description>
  </book>
  <book id="bk109">
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>After an inadvertant trip through a Heisenberg Unc</description>
  </book>
</catalog>
```

You can walk the tree to the first level of elements contained by the root using Microsoft® Visual Basic® Scripting Edition (VBScript).

```vbscript
Dim root
Dim xmlDoc
Dim child

Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmlDoc.async = False
xmlDoc.load("c:\books.xml")
```
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error ", myErr.reason)
Else
    'Set root to the XML document's root element, COLLECTION:
    Set root = xmlDoc.documentElement
    'Walk from the root to each of its child nodes:
    For Each child In root.childNodes
        MsgBox child.text
    Next
End If

When run, this example displays the text of each of the root's child nodes (book elements) as a separate text message.

You can use the following methods to navigate to a specific node or nodes deep within the tree.

- **nodeFromID method**

  The nodeFromID method takes a unique ID, as identified by an XML schema or document type definition (DTD), and returns the node corresponding to that specific ID.

- **getElementsByTagName method (DOMDocument)**

  The getElementsByTagName method takes a string representing a specific tag name and returns all element nodes with this tag name.

You can use the following code to retrieve all element nodes named <AUTHOR> in books.xml.

Dim ElemList
Dim xmlDoc
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmlDoc.async = False
xmlDoc.load("c:\xml\books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error ", myErr.reason)
Else
    Set ElemList = xmlDoc.getElementsByTagName("author")
    For i=0 To (ElemList.length -1)
The `selectNodes` method of the `IXMLDOMNode` object offers a more flexible way of accessing nodes using XML Path Language (XPath).

**See Also**

[Document Map](#) | [documentElement Property](#) | [IXMLDOMNode](#) | [selectNodes Method](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Creating New Nodes

The DOMDocument object exposes the createNode method, which allows you to create a qualified node by supplying the node type, name, and namespaceURI of the new node.

DOMDocument also exposes the following eight methods, which allow you to create specific Document Object Model (DOM) nodes.

- createAttribute method
- createCDATASection method
- createComment method
- createDocumentFragment method
- createElement method
- createEntityReference method
- createProcessingInstruction method
- createTextNode method

These methods return objects and interfaces and are described in the XML Reference. These specific objects and interfaces expose properties and methods associated with that node. For example, the following sample code assigns an XMLDOMElement object with the name of "item" to the variable xmlElem.

```xml
xmlElem = xmlDoc.createElement("item")
```

You can then add an attribute to that element by calling the setAttribute method and passing that method a string containing the attribute name and a string containing the attribute value.

```xml
xmlElem.setAttribute("id","bar")
```

Using **books.xml** again as an example, the following code creates a new element, PAGES, and uses the appendChild method to add it to the second child node of the root, COLLECTION. It then sets the text property for the new element to "400".

```xml
Dim xmlDoc
Dim root
```
Dim newElem
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmlDoc.async = False
xmlDoc.load("c:\books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set newElem = xmlDoc.createElement("PAGES")
    root.childNodes.item(1).appendChild newElem
    root.childNodes.item(1).lastChild.text = "400"
    MsgBox root.childNodes.item(1).xml
End If

You can assign a new document to a document object by assigning the document a new root element using the documentElement property.

XMLDoc.documentElement = XMLNode

In order for this operation to work, XMLNode must be an element type node.

Note Assigning the document object a new document element does not change the other children of the document. For example, the doctype element does not change.

See Also

Accessing the Document Tree | appendChild Method | createNode Method | documentElement Property | setAttribute Method | XML Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Use IXMLDOMNode

IXMLDOMNode is the principal object within the Document Object Model (DOM). Elements, attributes, comments, processing instructions, and every other document component can be represented as an IXMLDOMNode. The DOMDocument object is itself an IXMLDOMNode object, as are the members of node lists and named node maps. The following topics discuss how to use IXMLDOMNode.

- Gathering Information About a Node
- Getting and Setting Data Within a Node
- Navigating the Tree
- Manipulating the Children of a Node
- Transforming a Node

See Also

IXMLDOMNode
Gathering Information About a Node

The following method and properties can be used to get the following information about a node.

- **hasChildNodes method**
  
  If the node has any children.

- **namespaceURI property**
  
  The namespace to which the node belongs.

- **nodeName property**
  
  The name of the node.

- **nodeType property**
  
  The type of the node.

- **nodeTypeString property**
  
  The type of the node.

- **parsed property**
  
  If the node has been parsed.

- **specified property**
  
  If the node is explicitly specified in the XML file or implied within the document type definition (DTD) or XML Schema.

- **xml property**
  
  The string representation of the node.
The following sample code tests the `nodeNameString` property to see if the node is of type "element". If the node is of type "element", the string representation of the node is loaded into a new document object through the `loadXML` method, creating a document fragment.

```js
if (xmlNode.nodeTypeString == "element")
    XMLDoc.loadXML(xmlNode.xml)
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
}
```

You can use the following code if you prefer to work with the numeric representation.

```vb
if (xmlNode.nodeType == 1)
    XMLDoc.loadXML(xmlNode.xml)
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
End IF
```

You can get the name of a node by checking the `nodeName` property.

```js
currentNodeName=xmlNode.nodeName
```

You can use the following code to find nodes without child nodes and retrieve their content as a string.

```js
if (xmlNode.hasChildren == false)
    currentNodeContent=xmlNode.xml
```

You can use the following code to limit the nodes processed to nodes from specific namespaces.

```js
if (xmlNode.namespaceURI == "http://www.examples.microsoft.com/catalog")
    'catalog-specific processing here
```
Getting and Setting Data Within a Node

Data marked up as text within elements, attributes, comments, and processing instructions is exposed in the Document Object Model (DOM) as node values.

The following code sets the value of an attribute to "hello world".

```javascript
newAttNode = myElementNode.createAttribute("newAtt")
newAttNode.nodeValue = "hello world"
```

There are three ways to access text within an element node.

- The `nodeValue` property provides direct access to values of certain types of nodes, including attributes, text nodes, comments, processing instructions, and CDATA sections. Navigate down to the element's children (the text nodes within) and call `nodeValue`.
- The `text` property returns the text within the element. The text within an element can also be changed using the `text` property, but this will overwrite any child elements.

The following sample code tests to see if the text within the element `elem1` is equal to "hello world". If so, the element is assigned the text "hi! world".

```javascript
if (elem1.text == "hello world")
    elem1.text = "hi! world"
```

- The `nodeTypedValue` property is part of the data type support built into Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office and returns the typed value of the node. This property is only for typed elements. The `dataType` property, which returns the data type of the element, can be used in conjunction with this property. The `nodeTypedValue` and `datatype` properties can both be set. However, the parser will return an error if the new `nodeTypedValue` is of the wrong data type or if the new data type does not match the `nodeTypedValue`. For more information about data type support, see [XDR Schema Data Types](#)
Reference.

See Also

dataType Property | nodeValue Property | nodeTypedValue Property | text Property
Navigating the Tree

The Document Object Model (DOM) treats the contents of documents as a collection of nodes of various types. Some nodes contain other nodes, or groups of nodes, building a structure commonly referred to as the document tree. Programs that access or manipulate the information stored in XML documents through the DOM must navigate this tree structure.

From an IXMLDOMNode object, you can navigate directly to its parent node (parentNode), children (childNodes, firstChild, lastChild), siblings (previousSibling, nextSibling), or the document object to which the node belongs (ownerDocument). If the node is of type element, attribute, or entityReference, you can call the definition property to navigate to the schema definition of the node. In addition, if the node is of type element, processingInstruction, documentType, entity, or notation, you can navigate to the attributes on the node using the attributes property.

The following code tests to see if the elem1 element has any element children. If so, it navigates from elem1 to that element's first child. The text of that first child is then changed to "I'm the first child of elem1".

```javascript
if (elem1.hasChildNodes == true && elem1.firstChild.nodeTypeString == "element")
    elem1.firstChild.text = "I'm the first child of elem1"
```

To navigate to nodes other than the first or last sibling, you can start from the first or last child and use the previousSibling or nextSibling methods to move forward or backward through the list.

```javascript
wantedNodeCount=2
currentNode=1
set wantedNode=elem1.firstChild
Do While currentNode<wantedNodeCount
    set wantedNode=wantedNode.nextSibling
    currentNode=currentNode+1
Loop
```

The following is an example in Microsoft® JScript®.

```javascript
wantedNodeCount=2;
currentNode=1;
```
wantedNode = elem1.firstChild;
while (currentNode < wantedNodeCount) {
    wantedNode = wantedNode.nextSibling;
    currentNode++;
}

In either case, the variable `wantedNode` contains the desired node, and can be used for other processing or navigation. With a few additional comparisons, you can use techniques like these to find nodes that meet specific type and content criteria. If you prefer to walk through the list of child nodes using numeric references, use the `IXMLDOMNodeList` object returned by the `childNodes` property.

It is also possible to navigate to other nodes in the tree using the `selectNodes` and `selectSingleNode` methods. These methods take an XML Path Language (XPath) expression as an argument and return the node or nodes that match that query. For more information about XPath, see XPath Syntax.

See Also

- attributes Property
- childNodes Property
- definition Property
- firstChild Property
- lastChild Property
- nextSibling Property
- ownerDocument Property
- parentNode Property
- previousSibling Property
- selectNodes Method
- selectSingleNode Method
Manipulating the Children of a Node

The IXMLDOMNode object exposes four methods that allow you to manipulate the children of a node. Each of these methods takes an IXMLDOMNode object as an argument.

- appendChild method
- replaceChild method
- removeChild method
- insertBefore method

The following code takes a new element that is created using createElement and adds that as the last child of elem1.

```
newChild = XMLDoc.createElement("last_child")
elem1.appendChild(newChild)
```

The appendChild method always adds the node to the end of the list of children. To insert the node elsewhere in the child node list, use the insertBefore method. This method takes two parameters: the node to be inserted and the node before which the new child is to be inserted.

The following Microsoft® Visual Basic® Scripting Edition (VBScript) example loads books.xml and creates a new element, pages. Then, using a reference to the first child of the first node contained by the document element as the reference point (refElem), it inserts the new element with the insertBefore method.

```
Dim xmlDoc
Dim root
Dim newElem
Dim refElem
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmlDoc.async = False
xmlDoc.load("c:\xmlinact\books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
```
`create new element
Set newElem = xmlDoc.createElement("pages")
'start from the root element
Set root = xmlDoc.documentElement
'find the first child of the root element (using collection syntax)
Set firstElem = root.childNodes.item(1)
'find the first child of the first child (using property syntax)
Set refElem = firstElem.firstChild
'insert our new pages element
root.childNodes.item(1).insertBefore(newElem, refElem)
'report the content of the first child of the root element
MsgBox firstElem.xml
End If

The following is an example in Microsoft JScript®.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("c:\books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    //create new element
    var newElem = xmlDoc.createElement("pages");
    //start from the root element
    var root = xmlDoc.documentElement;
    //find the first child of the root element (using collection syntax)
    var firstElem = root.childNodes.item(1);
    //find the first child of the first child (using property syntax)
    var refElem = firstElem.firstChild;
    //insert our new pages element
    root.childNodes.item(1).insertBefore(newElem, refElem);
    //report the content of the first child of the root element
    alert(firstElem.xml);
}

The following VBScript example uses the removeChild method to remove a child node. Most of the code is similar to the preceding example, but this script removes an element rather than creating and adding one.

Dim xmlDoc
Dim root
Dim oldChild
Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
xmlDoc.async = False
xmlDoc.load("c:\books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
Dim myErr
Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    'find the first child of the root element (using collection syntax)
    Set firstElem = root.childNodes.item(1)
    Set oldChild = firstElem.firstChild
    firstElem.removeChild oldChild
    MsgBox firstElem.xml
End If

The following is an example in JScript.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmDoc.async = false;
xmDoc.load("c:\books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    //start from the root element
    var root = xmlDoc.documentElement;
    //find the first child of the root element (using collection syntax)
    var firstElem = root.childNodes.item(1);
    //pick the element to remove
    var oldChild = firstElem.firstChild;
    //remove the element
    firstElem.removeChild(oldChild);
    //report the content of the first child of the root element
    alert(firstElem.xml);
}

See Also

Accessing the Document Tree
Transforming a Node

The transformNode and transformNodeToObject methods can be used to transform an XML node into a string or object using XSL Transformations (XSLT). For more information about transforming nodes using XSLT, see the XSLT Developer's Guide.

See Also

transformNode Method | transformNodeToObject Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Use IXMLDOMNodeList

The IXMLDOMNodeList object is returned by the childNodes collection and the selectNodes and getElementsByTagName methods.

The following topics discuss how to use the IXMLDOMNodeList object.

- Gathering Information About the Node List
- Navigating the Node List

See Also

IXMLDOMNodeList
Gathering Information About the Node List

You can retrieve the length of the node list from the `length` property. Among other things, the length can be used for iterating through the list of children. For example, the following Microsoft® JScript® code loops through the children of `elem1`, searching for a child element with the text value of "hello world". If such a child exists, the index of that child is assigned to the variable "helloWorldIndex".

```javascript
for (i = 0; i < elem1.childNodes.length; i++){
    if (elem1.childNodes.item(i).text == "hello world")
        helloWorldIndex = i;
}
```

In Microsoft Visual Basic® Scripting Edition (VBScript), the same code would look like the following.

```vbscript
For i = 0 to elem1.childNodes.length - 1
    if (elem1.childNodes.item(i).text = "hello world") Then
        helloWorldIndex = i
Next
```

You can also move through the node list using the `nextNode` method.

See Also

[length Property (IXMLDOMNodeList)](length_property_ixmldomnodelist)
Navigating the Node List

You can retrieve a specific member of the node list by using the `item` method. The `item` method takes a number corresponding to a node's position within the node list. To retrieve the first node in the node list, call `item(0)`. It is also possible to navigate through the node list using the `nextNode` method, which returns the next node in the node list.

The following Microsoft® JScript® code performs the same task as the code in the previous topic.

```javascript
currentNode = elem1.childNodes.nextNode;
while (currentNode != null) {
    if (currentNode.text == "hello world")
        helloWorldNode = currentNode;
    currentNode = elem1.childNodes.nextNode;
}
```

This is the same example in Microsoft Visual Basic® Scripting Edition (VBScript).

```vbnet
Set currentNode = elem1.childNodes.nextNode
Do While currentNode <> null
    If currentNode.text = "hello world" Then
        helloWorldNode = currentNode
    End If
    currentNode = elem1.childNodes.nextNode
Loop
```

The `currentNode` is modified using the node returned by `elem1.childNodes.nextNode`. The initial position of the node list is defined to be before the first node. Therefore, the first time `nextNode` is called on the node list, the first node is returned. The code tests for `currentNode != null`. If the current node is the last node in the list or the node list has no members, `nextNode` returns null.

**See Also**

Gathering Information About the Node List | item Method
(IXMLDOMNodeList) | nextNode Method (IXMLDOMNodeList)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Use **IXMLDOMNamedNodeMap**

An **IXMLDOMNamedNodeMap** object is returned by the `attributes` property. The **IXMLDOMNamedNodeMap** object differs from the node list in that it is a collection of nodes that can be accessed by name as well as number. Because the order of attributes is not important in XML, this can make attribute processing easier.

The following topics discuss how to use the **IXMLDOMNamedNodeMap** object.

- [Gathering Information About the Named Node Map](#)
- [Navigating the Named Node Map](#)
- [Manipulating a Named Node Map](#)

**See Also**

- [attributes Property](#)
- [IXMLDOMNamedNodeMap](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Gathering Information About the Named Node Map

Like the IXMLDOMNodeList object, the IXMLDOMNamedNodeMap object exposes the length property. This property returns the number of members in the named node map.

See Also

[link to length Property (IXMLDOMNamedNodeMap)]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Navigating the Named Node Map

As with a node list, you can access members of the named node map by index using the `item` method. You can also access the members of a named node map name by using the following methods.

- **getNamedItem method**

  The `getNamedItem` method takes the name of the desired node as a parameter.

  The following code retrieves the value of the ID attribute on the `<elem1>` element and assigns that value to the variable `idValue`.

  ```javascript
  idValue = elem1.attributes.getNamedItem("ID").nodeValue
  ```

- **getQualifiedItem method**

  The `getQualifiedItem` method takes the name and namespace URI of the desired node.

Each method returns an `IXMLDOMNode` object.

Like the `IXMLDOMNodeList`, `IXMLDOMNamedNodeMap` exposes the `nextNode` property.

**See Also**

`item Method (IXMLDOMNamedNodeMap)`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Devel
Manipulating a Named Node Map

You can use the following methods to manipulate named node maps.

- **setNamedItem method**

  The `setNamedItem` method takes an XML node object as a parameter, adding that node to the named node map. If an attribute already exists with the same name as that specified by the `nodeName` of the node added, the old attribute is replaced. The following code creates a new attribute node with the name `ID` and adds it to the attributes of `<elem1>`.

  ```javascript
  idAtt = XMLDoc.createAttribute("ID")
  elem1.attributes.setNamedItem(idAtt)
  ```

- **removeNamedItem method**

  The `removeNamedItem` method takes a node name as a parameter, removing the node with that name.

- **removeQualifiedItem method**

  The `removeQualifiedItem` method takes a node name and namespace URI as its parameters, removing the corresponding attribute.

Alternatively, element nodes also support the `setAttribute`, `setAttributeNode`, `removeAttribute`, and `removeAttributeNode` methods if you prefer to treat attributes as properties of elements rather than as members of a collection contained by an element.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
DOM Reference

This section contains the following topics.

- XML DOMEnumerated Constants
- XML DOM Objects/Interfaces
- XML DOM Properties
- XML DOM Methods
- XML DOM Events
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
XML DOM Enumerated Constants

IXMLDOMNodeType enumerations specify valid settings for the Document Object Model (DOM) node type. The node type determines valid values and whether the node can have child nodes.

Values

The IXMLDOMNodeType enumeration defines the following valid values.

NODE_ELEMENT (1)
   The node represents an element (its nodeTypeString property is "element"). An Element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection, and EntityReference. The Element node can be the child of the Document, DocumentFragment, EntityReference, and Element nodes.

NODE_ATTRIBUTE (2)
   The node represents an attribute of an element (its nodeTypeString property is "attribute"). An Attribute node can have the following child node types: Text and EntityReference. The Attribute node does not appear as the child node of any other node type; it is not considered a child node of an Element.

NODE_TEXT (3)
   The node represents the text content of a tag (its nodeTypeString property is "text"). A Text node cannot have any child nodes. The Text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.

NODE_CDATA_SECTION (4)
   The node represents a CDATA section in the XML source (its nodeTypeString property is "cdatasection"). CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATASection node cannot have any child nodes. The CDATASection node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.

NODE_ENTITY_REFERENCE (5)
   The node represents a reference to an entity in the XML document (its
nodeTypeString property is "entityreference"). This applies to all entities, including character entity references. An EntityReference node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The EntityReference node can appear as the child of the Attribute, DocumentFragment, Element, and EntityReference nodes.

NODE_ENTITY (6)
The node represents an expanded entity (its nodeTypeString property is "entity"). An Entity node can have child nodes that represent the expanded entity (for example, Text and EntityReference nodes). The Entity node can appear as the child of the DocumentType node.

NODE_PROCESSING_INSTRUCTION (7)
The node represents a processing instruction from the XML document (its nodeTypeString property is "processinginstruction"). A ProcessingInstruction node cannot have any child nodes. The ProcessingInstruction node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.

NODE_COMMENT (8)
The node represents a comment in the XML document (its nodeTypeString property is "comment"). A Comment node cannot have any child nodes. The Comment node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.

NODE_DOCUMENT (9)
The node represents a document object, that as the root of the document tree, provides access to the entire XML document (its nodeTypeString property is "document"). It is created using the progID "Microsoft.XMLDOM" or through a data island using <XML> or <SCRIPT LANGUAGE=XML>. A Document node can have the following child node types: Element (maximum of one), ProcessingInstruction, Comment, and DocumentType. The Document node cannot appear as the child of any node types.

NODE_DOCUMENT_TYPE (10)
The node represents the document type declaration, indicated by the <!DOCTYPE> tag (its nodeTypeString property is "documenttype"). A DocumentType node can have the following child node types: Notation and Entity. The DocumentType node can appear as the child of the Document node.

NODE_DOCUMENT_FRAGMENT (11)
The node represents a document fragment (its nodeTypeString property is
"documentfragment"). The DocumentFragment node associates a node or subtree with a document without actually being contained within the document. A DocumentFragment node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The DocumentFragment node cannot appear as the child of any node types.

**NODE_NOTATION (12)**

The node represents a notation in the document type declaration (its `nodeTypeString` property is "notation"). A Notation node cannot have any child nodes. The Notation node can appear as the child of the DocumentType node.

**Remarks**

Note that built-in and numeric entities (such as `&amp;`) are exposed as Text nodes rather than as Entity nodes.

**See Also**

[nodeTypeString Property] [IXMLDOMNode]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
**XML DOM Objects/Interfaces**

This part of the XML Document Object Model (DOM) reference describes the programming object syntax.

[Script, Visual Basic, C/C++]

The first table lists the core objects/interfaces. The second table lists the implemented objects/interfaces.

**XML DOM Core Objects/Interfaces**

The following objects/interfaces represent implementations of the XML DOM core objects/interfaces. The objects/interfaces include Microsoft extensions to support the following items and operations: namespaces, data types, XML schemas, Extensible Stylesheet Language (XSL), XSL Transformations (XSLT) operations, asynchronous loading, and the saving of documents.

<table>
<thead>
<tr>
<th>IXMLElement</th>
<th>Represents the top node of the XML DOM tree.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXMLElement2</td>
<td>An extension of DOMDocument. This extension supports schema caching and runtime validation, and provides a way to switch on XML Path Language (XPath) support.</td>
</tr>
<tr>
<td>IXMLElementMap</td>
<td>Enables iteration and access, by name, to the collection of attributes. IXMLElementMap includes support for namespaces.</td>
</tr>
<tr>
<td>IXMLElement</td>
<td>Represents a single node in the document tree. IXMLElement is the base interface for accessing data in the XML object model. This interface includes support for data types, namespaces, document type definitions (DTDs), and XML schemas.</td>
</tr>
</tbody>
</table>
### XML DOM Implemented Objects/Interfaces

The following table represents implementations of other XML DOM interfaces.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IXMLDOMNodeList</strong></td>
<td>Enables iteration and indexed access operations on the live collection of IXMLDOMNode.</td>
</tr>
<tr>
<td><strong>IXMLDOMParseError</strong></td>
<td>Returns detailed information about the last error, including the error number, the line number, the character position, and a text description.</td>
</tr>
<tr>
<td><strong>IXMLHTTPRequest</strong></td>
<td>Provides client-side protocol support for communication with HTTP servers.</td>
</tr>
<tr>
<td><strong>IServerXMLHTTPRequest/ServerXMLHTTP</strong></td>
<td>Provides methods and properties that enable you to establish an HTTP connection between files or objects on different Web servers. This interface is derived from IXMLHTTPRequest.</td>
</tr>
<tr>
<td><strong>IXMLDOMAttribute</strong></td>
<td>Represents an attribute.</td>
</tr>
<tr>
<td><strong>IXMLDOMCDATASection</strong></td>
<td>Quotes or escapes blocks of text so that text is not interpreted as markup language.</td>
</tr>
<tr>
<td><strong>IXMLDOMCharacterData</strong></td>
<td>Provides text manipulation methods that are used by several objects.</td>
</tr>
<tr>
<td><strong>IXMLDOMComment</strong></td>
<td>Represents the content of an XML comment.</td>
</tr>
<tr>
<td><strong>IXMLDOMDocumentFragment</strong></td>
<td>Represents a lightweight object that is useful for tree insertion operations.</td>
</tr>
<tr>
<td><strong>IXMLDOMDocumentType</strong></td>
<td>Contains information associated with the</td>
</tr>
</tbody>
</table>
**IXMLDOMElement**
Represents the element.

**IXMLDOMEntity**
Represents a parsed or unparsed entity in the XML document.

**IXMLDOMEntityReference**
Represents an entity reference node.

**IXMLDOMImplementation**
Provides methods that are independent of any particular instance of the document object model.

**IXMLDOMNotation**
Contains a notation declared in the DTD or schema.

**IXMLDOMProcessingInstruction**
Represents a processing instruction. XML defines this instruction to keep processor-specific information in the text of the document.

**XMLSchemaCache**
Represents a set of namespace URIs.

**IXMLDOMSchemaCollection/XMLSchemaCache**
Represents a SchemaCache object.

**IXMLDOMSchemaCollection2/XMLSchemaCache**
An extension of IXMLDOMSchemaCollection. Represents a list of nodes that match an XPath expression.

**IXMLDOMSelection**
Represents the text content of an element or attribute.

**IXMLDOMText**

**IXSLProcessor**
Used for performing transformations with cached XSL templates.

**IXSLTemplate**
Represents a cached XSL style sheet.
For a list of all valid node types, see XML DOM Enumerated Constants.

[C/C++]

Implementing header files

If you have Microsoft® Internet Explorer 5.0 or Microsoft® Internet Explorer 5.5 installed, you can download the interface header files from MSDN®.

To download the header files

1. Under Web & Internet Samples TOC, click Headers and Libraries.
2. Click Internet Explorer 5.01 Headers & Libraries or Internet Explorer 5.5 Headers & Libraries. Then follow the on-screen instructions to download these headers and libraries. Save them to C:\Workshop.
5. Click the Directories tab, and then add C:\Workshop\Include and C:\Workshop\Include\NT5 to the top of the Directories list.
6. In the Show Directories for box, select Library files.
7. Add C:\Workshop\Lib and C:\Workshop\Lib\NT5 to the top of the Directories list.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☑ in the upper-left corner of the page.

Other Resources MS

Web & Internet Samples
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refe
IXMLDOMAttribute

Represents an attribute of the IXMLDOMElement. Valid and default values for the attribute are defined in a document type definition (DTD) or schema.

[Script]

Example

The following script example creates an IXMLDOMAttribute object from the first attribute of the root and displays the object's value and xml properties.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var objDOMatt;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    objDOMatt = root.firstChild.attributes.item(0);
    alert(objDOMatt.value);
    alert(objDOMatt.xml);
}
```

[Visual Basic]

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMAttribute object from the first attribute of the root and displays the object's value and xml properties.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim objDOMatt As IXMLDOMAttribute
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
Dim myErr
Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  Set objDOMatt = root.firstChild.Attributes.Item(0)
  MsgBox (objDOMatt.Value)
  MsgBox (objDOMatt.xml)
End If

[C/C++]

Example

#import "msxml5.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface)  \
  if(pInterface)  \
  {
    pInterface->Release();\n    pInterface = NULL;\n  }

#define RELEASE(pInterface)  \
  {
    pInterface->Release();\n    pInterface = NULL;\n  }

int main(int argc, char* argv[])
{
  _variant_t varValue;
  BSTR bstrAttributeName = ::SysAllocString(_T("dateCreated"));
  IXMLDOMAttribute *pIXMLDOMAttribute = NULL;
  IXMLDOMElement *pIXMLDOMElement = NULL;
  IXMLDOMDocument *pIXMLDOMDocument = NULL;
  HRESULT hr;

  try
  {
    // Initialize pIXMLDOMDocument and load an
    // XML file into it.
    // Get the document root.
    hr = pIXMLDOMDocument->get_documentElement( &pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;
    if(pIXMLDOMElement)
    {  

varValue = T("Year 2000");

// Set the "dateCreated" attribute to "year 2000"
hr = pIXMLDOMElement->setAttribute(bstrAttributeName, varValue)
SUCCEEDED(hr) ? 0 : throw hr;

// Retrieve the changed attribute.
hr = pIXMLDOMElement->getAttributeNode(bstrAttributeName,
&pIXMLDOMAttribute);
SUCCEEDED(hr) ? 0 : throw hr;

pIXMLDOMAttribute->get_nodeValue(&varValue);
if(varValue.vt != VT_NULL)
{
    // Display the retrieved attribute in a message box.
    ::MessageBox(NULL, _bstr_t(varValue), bstrAttributeName, MB

RELEASE(pIXMLDOMElement);
RELEASE(pIXMLDOMAttribute);
}
::SysFreeString(bstrAttributeName);
bstrAttributeName=NULL;
}
catch(...)
{
if(bstrAttributeName)
{
    ::SysFreeString(bstrAttributeName);
bstrAttributeName = NULL;
}
CHECK_AND_RELEASE(pIXMLDOMElement);
// DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.
return 0;
}

Output (in a message box)

Year 2000

Remarks

Attribute nodes cannot be the immediate children of IXMLDOMDocumentFragment. However, they can be associated with element nodes that are children of IXMLDOMDocumentFragment node.
The relationship between the value and specified members can be summarized as follows: If the attribute has an assigned value in the document and the specified property is True, the value is the assigned value. If the attribute has no assigned value in the document and has a default value in the DTD, the specified property is False and the value is the default value in the DTD. If the attribute has no assigned value in the document and has a value of #IMPLIED in the DTD, the attribute does not appear in the structure model of the document.

In XML, the child nodes of the attribute node represent the value of an attribute because the value can contain entity references. Thus attributes that contain entity references will have a child list containing both text nodes and entity reference nodes. In addition, because the attribute type might be unknown, there are no tokenized attribute values.

[C/C++]

IXMLDOMAttribute inherits IXMLDOMNode but are not actually child nodes of the element and are not considered part of the document tree. Attributes are considered members of their associated elements rather than independent and separate. Thus IXMLDOMAttribute parentNode, previousSibling, and nextSibling members have the value Null.

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Inherits From: IXMLDOMNode

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

Versioning

MSXML 2.0 and later
See Also

IXMLDOMAttribute Members | IXMLDOMElement |
IXMLDOMDocumentFragment | value Property | specified Property |
IXMLDOMNode | parentNode Property | previousSibling Property | nextSibling Property
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMAttribute Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>dataType</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>name</td>
<td>Contains the attribute name. Read-only.</td>
</tr>
<tr>
<td>namespaceURI</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of this node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
<tr>
<td>nodeType</td>
<td>Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node</td>
</tr>
</tbody>
</table>
can have child nodes. Read-only.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeTypedValue*</td>
<td>Contains the node value expressed in its defined data type. Read/write.</td>
</tr>
<tr>
<td>nodeTypeString*</td>
<td>Returns the node type in string form. Read-only.</td>
</tr>
<tr>
<td>nodeValue</td>
<td>Contains the text associated with the node. Read/write.</td>
</tr>
<tr>
<td>ownerDocument</td>
<td>Returns the root of the document that contains the node. Read-only.</td>
</tr>
<tr>
<td>parentNode</td>
<td>Contains the parent node. Read-only.</td>
</tr>
<tr>
<td>parsed*</td>
<td>Indicates the parsed status of the node and child nodes. Read-only.</td>
</tr>
<tr>
<td>prefix*</td>
<td>Returns the namespace prefix. Read-only.</td>
</tr>
<tr>
<td>previousSibling</td>
<td>Contains the previous sibling of this node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>specified</td>
<td>Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>text</td>
<td>Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.</td>
</tr>
<tr>
<td>value</td>
<td>Contains the attribute value. Read/write.</td>
</tr>
<tr>
<td>xml</td>
<td>Contains the XML representation of the node and all its descendants. Read-only.</td>
</tr>
</tbody>
</table>

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendChild</td>
<td>Appends new child node as the last child of this node.</td>
</tr>
<tr>
<td>cloneNode</td>
<td>Clones a new node.</td>
</tr>
<tr>
<td>hasChildNodes</td>
<td>Provides a fast way to determine whether a node has children.</td>
</tr>
<tr>
<td>insertBefore</td>
<td>Inserts a child node to the left of the specified node or at the end of the list.</td>
</tr>
<tr>
<td>removeChild</td>
<td>Removes the specified child node from the list of children and returns it.</td>
</tr>
<tr>
<td>replaceChild</td>
<td>Replaces the specified old child node with the supplied</td>
</tr>
</tbody>
</table>
new child node.

selectNodes
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

transformNode
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNodeToObject
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation in the supplied object.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**IXMLDOMCDATASection**

Used to quote or escape blocks of text to keep that text from being interpreted as markup language.

[C/C++]

Although the *IXMLDOMCDATASection* inherits *IXMLDOMText*, unlike text nodes, the `normalize` method of *IXMLDOMElement* does not merge CDATASection nodes.

[Script]

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeCDATA;
nodeCDATA = xmlDoc.createCDATASection("Hello");
alert(nodeCDATA.xml);
```

[Visual Basic]

**Example**

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeCDATA As IXMLDOMCDATASection
Set nodeCDATA = xmlDoc.createCDATASection("Hello")
MsgBox nodeCDATA.xml
```

[C/C++]

**Example**

The following C/C++ example creates and appends a new CDATASection node to the root document element.

```c
#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
```
{ if FAILED(_hr) throw(_hr); }

void XMLDOMCDATASectionSample()
{
    try {
        IXMLDOMDocumentPtr docPtr;
        IXMLDOMNodePtr DOMNodePtr;

        // init
        TESTHR(CoInitialize(NULL));
        TESTHR(docPtr.CreateInstance("msxml2.domdocument"));

        // load a document
        _variant_t varXml("C:\book.xml");
        _variant_t varOut((bool)TRUE);
        varOut = docPtr->load(varXml);
        if ((bool)varOut == FALSE)
            throw(0);
        MessageBox(NULL, _bstr_t(docPtr->xml), _T("Original Document"));
        DOMNodePtr = docPtr->createCDATASection("<fragment>XML tags inside a CDATA section</fragment>");
        docPtr->documentElement->appendChild(DOMNodePtr);
        MessageBox(NULL, _bstr_t(docPtr->xml), _T("New Document"), MB_OK);
    } catch(...) {
        MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK)
    }
    CoUninitialize();
}

Remarks

CDATA sections let you include material such as XML fragments within XML documents without needing to escape all the delimiters. The only delimiter recognized in a CDATA section is the "]]>" string that ends the CDATA section.

CDATA sections cannot be nested.

The text contained by the CDATA section is stored in a text node. This text can contain characters that need to be escaped outside of CDATA sections.

IXMLDOMCDATASection has no unique members of its own, but exposes the same
members as the IXMLDOMText object.

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**Versioning**

MSXML 2.0 and later

**See Also**

normalize Method | IXMLDOMCDATASection Members | IXMLDOMElement | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMCDATASection Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMText.

Properties

- **attributes**: Contains the list of attributes for this node. Read-only.
- **baseName**: Returns the base name for the name qualified with the namespace. Read-only.
- **childNodes**: Contains a node list containing the children nodes. Read-only.
- **data**: Stores the node data depending on the node type. Read/write.
- **dataType**: Specifies the data type for this node. Read/write.
- **definition**: Returns the definition of the node in the document type definition (DTD) or schema. Read-only.
- **firstChild**: Contains the first child of the node. Read-only.
- **lastChild**: Returns the last child node. Read-only.
- **length**: Specifies the length, in characters, of the data. Read-only.
- **namespaceURI**: Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.
- **nextSibling**: Contains the next sibling of the node in the parent's child list. Read-only.
- **nodeName**: Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.
- **specifies the XML Document Object Model (DOM) node
**nodeType**

Type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**

Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**

Returns the node type in string form. Read-only.

**nodeValue**

Contains the text associated with the node. Read/write.

**ownerDocument**

Returns the root of the document that contains this node. Read-only.

**parentNode**

Contains the parent node. Read-only.

**parsed**

Indicates the parsed status of the node and child nodes. Read-only.

**prefix**

Returns the namespace prefix. Read-only.

**previousSibling**

Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**

Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema. Read-only.

**text**

Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**

Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

- **appendChild**
  Appends a new child node as the last child of the node.

- **appendData**
  Appends the supplied string to the existing string data.

- **cloneNode**
  Clones a new node.

- **deleteData**
  Deletes specified data.

- **hasChildNodes**
  Provides a fast way to determine whether a node has children.

- **insertBefore**
  Inserts a child node to the left of the specified node or at the end of the list.

- **insertData**
  Inserts a string at the specified offset.
**removeChild**  
Removes the specified child node from the list of children and returns it.

**replaceChild**  
Replaces the specified old child node with the supplied new child node.

**replaceData**  
Replaces the specified number of characters with the supplied string.

**selectNodes***  
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode***  
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**splitText**  
Splits this text node into two text nodes at the specified offset and inserts the new text node into the tree as a sibling that immediately follows this node.

**substringData**  
Retrieves a substring of the full string from the specified range.

**transformNode***  
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject**  
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMCDATASection | IXMLDOMText
IXMLDOMCharacterData

Provides text manipulation methods that are used by several objects.

[Script]

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeDescription;
xmlDoc.async = false;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error "+ myErr.reason);
} else {
    nodeDescription = xmlDoc.selectSingleNode("//description");
    alert(nodeDescription.text);
}
```

[Visual Basic]

Example

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeDescription As IXMLDOMElement
xmlDoc.async = False
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error "& myErr.reason)
Else
    Set nodeDescription = xmlDoc.selectSingleNode("//description")
    MsgBox nodeDescription.Text
End If
```

Remarks
IXMLDOMCharacterData does not directly correspond to any node type. Methods are reused by IXMLDOMCDATASection, IXMLDOMComment, and IXMLDOMText. These methods handle large amounts of text, including sizes larger than can be manipulated natively using string constructs.

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

Versioning

MSXML 2.0 and later

See Also

IXMLDOMCDATASection | IXMLDOMCharacterData Members | IXMLDOMComment | IXMLDOMElememt | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMCharacterData Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C+]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>data</td>
<td>Stores the node data depending on the node type. Read/write.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>length</td>
<td>Specifies the length, in characters, of the data. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of the node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
<tr>
<td></td>
<td>Specifies the XML Document Object Model (DOM) node</td>
</tr>
</tbody>
</table>
**nodeType**

Type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**

*Contains the node value expressed in its defined data type. Read/write.*

**nodeTypeString**

*Returs the node type in string form. Read-only.*

**nodeValue**

Contains the text associated with the node. Read/write.

**ownerDocument**

Returns the root of the document that contains this node. Read-only.

**parentNode**

Contains the parent node. Read-only.

**parsed**

*Indicates the parsed status of the node and child nodes.*

**prefix**

*Returns the namespace prefix. Read-only.*

**previousSibling**

Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**

*Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema. Read-only.*

**text**

Contains the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**

Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**

Appends a new child node as the last child of the node.

**appendData**

Appends the supplied string to the existing string data.

**cloneNode**

Clones a new node.

**deleteData**

Deletes the specified data.

**hasChildNodes**

Provides a fast way to determine whether a node has children.

**insertBefore**

Inserts a child node to the left of the specified node or at the end of the list.
insertData
Inserts a string at the specified offset.

removeChild
Removes the specified child node from the list of children and returns it.

replaceChild
Replaces the specified old child node with the supplied new child node.

replaceData
Replaces the specified number of characters with the supplied string.

selectNodes*
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode*
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

substringData
Retrieves a substring of the full string from the specified range.

transformNode*
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNodeToObject*
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

Events
None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
IXMLDOMCharacterData | IXMLDOMElement | IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**IXMLDOMComment**

Represents the content of an XML comment.

[Script]

**Example**

The following script example creates a new IXMLDOMComment (comment) and appends it as the last child node of DOMDocument.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var comment;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    comment = xmlDoc.createComment("Hello World!");
    root.appendChild(comment);
    alert(root.xml);
}
```

[Visual Basic]

**Example**

The following Microsoft® Visual Basic® example creates a new IXMLDOMComment (comment) and appends it as the last child node of DOMDocument.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim comment As IXMLDOMComment
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
```
Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set comment = xmlDoc.createComment("Hello World!")
    root.appendChild comment
    MsgBox (root.xml)
End If

[C/C++]

The following sample creates and appends a new XMLDOMComment node to the root document element.

#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
    { if FAILED(_hr) throw(_hr); }

void XMLDOMCommentSample()
{
    try {
        IXMLDOMDocumentPtr docPtr;
        IXMLDOMNodePtr DOMNodePtr;

        //init
        TESTHR(CoInitialize(NULL));
        TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));

        // load a document
        _variant_t varXml("C:\book.xml");
        _variant_t varOut((bool)TRUE);
        varOut = docPtr->load(varXml);
        if ((bool)varOut == FALSE)
            throw(0);
        MessageBox(NULL, _bstr_t(docPtr->xml), _T("Original Document"))
        DOMNodePtr = docPtr->createComment("This is a comment node");
        docPtr->documentElement->appendChild(DOMNodePtr);
        MessageBox(NULL, _bstr_t(docPtr->xml), _T("New Document"), MB_}
    } catch(...) {
        MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK)
    }
    CoUninitialize();
}
Remarks

The content refers to all characters between the start <!-- and end --> tags.

IXMLDOMComment has no unique members and exposes the same members IXMLDOMCharacterData.

Versioning

MSXML 2.0 and later

Requirements

**Implementation:** msxml4.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☑ in the upper-left corner of the page.

See Also

IXMLDOMCharacterData | IXMLDOMComment Members | IXMLDOMElement
The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMCharacterData.

### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>data</td>
<td>Stores the node data depending on the node type. Read/write.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>length</td>
<td>Specifies the length, in characters, of the data. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of this node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
<tr>
<td></td>
<td>Specifies the XML Document Object Model (DOM) node</td>
</tr>
</tbody>
</table>
**nodeType**  
Type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**  
Contains this node's value expressed in its defined data type. Read/write.

**nodeTypeString**  
Returns the node type in string form. Read-only.

**nodeValue**  
Contains the text associated with the node. Read/write.

**ownerDocument**  
Returns the root of the document that contains the node. Read-only.

**parentNode**  
Contains the parent node. Read-only.

**parsed**  
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**  
Returns the namespace prefix. Read-only.

**previousSibling**  
Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**  
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**text**  
Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**  
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**  
Appends a new child node as the last child of the node.

**appendData**  
Appends the supplied string to the existing string data.

**cloneNode**  
Clones a new node.

**deleteData**  
Deletes specified data.

**hasChildNodes**  
Provides a fast way to determine whether a node has children.

**insertBefore**  
Inserts a child node to the left of the specified node or at the end of the list.
**insertData**
Inserts the supplied string at the specified offset.

**removeChild**
Removes the specified child node from the list of children and returns it.

**replaceChild**
Replaces the specified old child node with the supplied new child node.

**replaceData**
Replaces the specified number of characters with the supplied string.

**selectNodes***
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode***
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**substringData**
Retrieves a substring of the full string from the specified range.

**transformNode***
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject***
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

**IXMLDOMComment** | **IXMLDOMCharacterData**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMDocument/DOMDocument

Represents the top level of the XML source. Includes members for retrieving and creating all other XML objects.

[Script]

Example

The following script examples demonstrate how to create the two types of objects.

```javascript
var objDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objFTDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
```

[Visual Basic]

Example

The following Microsoft® Visual Basic® examples demonstrate how to create the two types of objects.

```vbnet
Dim objDoc As New Msxml2.DOMDocument50
Dim objFTDoc As New Msxml2.FreeThreadedDOMDocument50
```

[C/C++]

Example

The following C/C++ example creates DOMDocument and queries for the other interfaces.

```c
HRESULT hr;
IXMLDOMDocument2 * pXMLDoc;
IXMLDOMNode * pXDN;
//...
hr = CoInitialize(NULL);
// Check the return value, hr...
hr = CoCreateInstance(CLSID_DOMDocument50, NULL, CLSCTX_INPROC_SERVER,
```

```c
```
In addition to the DOM interfaces, DOMDocument implements a number of standard COM interfaces. You can call the QueryInterface method on DOMDocument to get the following interfaces.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUnknown</td>
<td>DOMDocument is a wrapper object and each query for DOMDocument returns a new wrapper. You should only compare IUnknown interface pointers.</td>
</tr>
<tr>
<td>IConnectionPointContainer</td>
<td>Supports outgoing events ondataavailable and onreadystatechange through IPropertyNotifySink::OnChanged and IDispatch::Invoke.</td>
</tr>
<tr>
<td>IDispatch</td>
<td>Interface used by Visual Basic.</td>
</tr>
<tr>
<td>IDispatchEx</td>
<td>Interface used by dynamic late-bound scripting languages such as Microsoft Visual Basic Scripting Edition (VBScript) and Microsoft JScript®. This is not fully implemented. The following methods always return E_NOTIMPL: DeleteMemberByName or DeleteMemberByDispID, GetMemberProperties, GetMemberName, GetNextDispID, and GetNameSpaceParent. Can be used to get a FreeThreadedMarshaler for a free-threaded DOM document. This allows the free-threaded DOM document to be used in Active Server Pages (ASP) shared Session and Application states for sharing XML documents across multiple clients in memory.</td>
</tr>
<tr>
<td>IMarshal</td>
<td></td>
</tr>
</tbody>
</table>

**Note** In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

When the SetInterfaceSafetyOptions method is called with nonzero safety options, Microsoft®
**Remarks**

**Note**  When the object-creation methods (such as createElement) are used on the document, nodes are created in the context of the document (the
The ownerDocument property of the node points to the document, but the node is not part of the document tree. The node is only part of the document tree when it is explicitly added to the tree by calling insertBefore, replaceChild, or appendChild (or for attributes, setAttributeNode).

DOMDocument represents the top node in the tree. It implements all of the base Document Object Model (DOM) document methods and provides additional members that support Extensible Stylesheet Language (XSL) and XML transformations.

Only one object can be created: the document. All other objects are accessed or created from the document.

The document can be created using either a free-threaded or a rental-threaded model. The behavior of the two models is identical; rental-threaded documents exhibit better performance because the parser does not need to manage concurrent access among threads. You cannot combine nodes or documents that are created using differing threading models. The document threading model is determined by the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Rental-threaded model</th>
<th>Free-threaded model</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgID</td>
<td>Msxml2.DOMDocument.5.0</td>
<td>Msxml2.FreeThreadedDOMDocument.5.0</td>
</tr>
<tr>
<td>ClassID</td>
<td>88D969C0-F192-11D4-A65F-0040963251E5</td>
<td>88D969C1-F192-11D4-A65F-0040963251E5</td>
</tr>
<tr>
<td>VB Class Name</td>
<td>DOMDocument50</td>
<td>FreeThreadedDOMDocument50</td>
</tr>
</tbody>
</table>

**Note** When developing applications, you might typically consider an interface separately from any CoClasses that implement it. For example, the DOMDocument CoClass implements the IXMLDOMDocument interface. Normally, interfaces and CoClasses might be documented separately and used independently of one another. The IXMLDOMDocument interface, however, was not designed to be implemented separately from one of its provided CoClasses/implementations. It should always be implemented with either DOMDocument or DOMDocument2.

**Versioning**
MSXML 2.0 and later

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**IXMLDOMDocument/DOMDocument Members**

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>async</code>*</td>
<td>Specifies if asynchronous download is permitted. Read/write.</td>
</tr>
<tr>
<td><code>attributes</code></td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td><code>baseName</code>*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td><code>childNodes</code></td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td><code>dataType</code>*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td><code>definition</code>*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td><code>doctype</code></td>
<td>Contains the document type node that specifies the DTD for this document. Read-only.</td>
</tr>
<tr>
<td><code>documentElement</code></td>
<td>Contains the root element of the document. Read/write.</td>
</tr>
<tr>
<td><code>firstChild</code></td>
<td>Contains the first child of this node. Read-only.</td>
</tr>
<tr>
<td><code>implementation</code></td>
<td>Contains the IXMLDOMImplementation object for the document. Read-only.</td>
</tr>
</tbody>
</table>
lastChild
Returns the last child node. Read-only.

namespaceURI*
Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.

nextSibling
Contains the next sibling of the node in the parent's child list. Read-only.

nodeName
Returns the qualified name for attribute, document type, element, entity, or notation nodes. Returns a fixed string for all other node types. Read-only.

nodeType
Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.

nodeTypedValue*
Contains this node's value expressed in its defined data type. Read/write.

nodeTypeString*
Returns the node type in string form. Read-only.

nodeValue
Contains the text associated with the node. Read/write.

ondataavailable*
Specifies the event handler for the ondataavailable event. Write-only.

onreadystatechange*
Specifies the event handler to be called when the readyState property changes. Write-only.

ontransformnode*
Specifies the event handler for the ontransformnode event. Write-only.

ownerDocument
Returns the root of the document that contains this node. Read-only.

parentNode
Contains the parent node. Read-only.

parsed*
Indicates the parsed status of the node and child nodes. Read-only.

parseError*
Returns an IXMLDOMParseError object that contains information about the last parsing error. Read-only.

prefix*
Returns the namespace prefix. Read-only.

preserveWhiteSpace*
Specifies the default white space handling. Read/write.

previousSibling
Contains the previous sibling of the node in the parent's child list. Read-only.
readyState

Indicates the current state of the XML document. Read-only.

resolveExternals

Indicates whether external definitions (resolvable namespaces, DTD external subsets, and external entity references) are to be resolved at parse time, independent of validation. Read/write.

specified

Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

text

Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

url

Returns the URL for the last loaded XML document. Read-only.

validateOnParse

Indicates whether the parser should validate this document. Read/write.

xml

Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

Methods

abort

Aborts an asynchronous download in progress.

appendChild

Appends a new child as the last child of this node.

cloneNode

Clones a new node.

createAttribute

Creates a new attribute with the specified name.

createCDATASection

Creates a CDATA section node that contains the supplied data.

createComment

Creates a comment node that contains the supplied data.

createDocumentFragment

Creates an empty IXMLDOMDocumentFragment object.

createElement

Creates an element node using the specified name.

createEntityReference

Creates a new EntityReference object.
**createNode**

Creates a node using the supplied type, name, and namespace.

**createProcessingInstruction**

Creates a processing instruction node that contains the supplied target and data.

**createTextNode**

Creates a text node that contains the supplied data.

**getElementsByTagName**

Returns a collection of elements that have the specified name.

**hasChildNodes**

Provides a fast way to determine whether a node has children.

**insertBefore**

Inserts a child node to the left of the specified node or at the end of the list.

**load**

Loads an XML document from the specified location.

**loadXML**

Loads an XML document using the supplied string.

**nodeFromID**

Returns the node that matches the ID attribute.

**removeChild**

Removes the specified child node from the list of children and returns it.

**replaceChild**

Replaces the specified old child node with the supplied new child node.

**save**

Saves an XML document to the specified location.

**selectNodes**

Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode**

Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**transformNode**

Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation.

**transformNodeToObject**

Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.
Events

- **ondataavailable*** Denotes an extension to the W3C DOM.
  Fires when new data is available.

- **onreadystatechange*** Fires when the `readyState` property changes.

- **ontransformnode*** Fires before the style sheet is applied to a node.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- [IXMLDOMImplementation](#)
- [IXMLDOMNode](#)
- [IXMLDOMParseError](#)
- [ondataavailable Event](#)
- [ontransformnode Event](#)
- [readyState Property](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMDocument2

An extension of DOMDocument that supports schema caching and validation features, and switching on XML Path Language (XPath) support.

[Script]

Example

```javascript
var objDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objFTDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0")
```

[Visual Basic]

Example

```vbnet
Dim objDoc As New Msxml2.DOMDocument50
Dim objFTDoc As New Msxml2.FreeThreadedDOMDocument50
```

[C/C++]

Example

The following C/C++ example creates DOMDocument and queries for the IXMLDOMDocument2 interface.

```c
HRESULT hr;
IXMLDOMDocument2 * pXMLDoc = NULL;

//...
hr = CoInitialize(NULL); // Check the return value, hr...
hr = CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_INPROC_SERVER,
IID_IXMLDOMDocument2, (void**)&pXMLDoc); // Check the return
```

Remarks

The following CLSIDs support IID_IXMLDOMDocument2 in CoCreateInstance. The following proglIDs return an object that can be cast into an
IXMLDOMDocument2 interface in Microsoft® Visual Basic®. IXMLDOMDocument2 methods and properties are transparently available in late-bound scripting languages that go through IDispatch.

The same is true for DOMDocument objects exposed through the responseXML (IXMLHTTPRequest) property, and the DOMDocument and XSLDocument properties in the XML Multipurpose Internet Mail Extension (MIME) Viewer.

<table>
<thead>
<tr>
<th>CLSID</th>
<th>ProgID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLSID_DOMDocument</td>
<td>Msxml2.DOMDocument</td>
</tr>
<tr>
<td>CLSID_DOMFreeThreadedDocument</td>
<td>Msxml2.FreeThreadedDOMDocument</td>
</tr>
</tbody>
</table>

**Versioning**

MSXML 2.6 and later

**Requirements**

**Implementation:** msxml4.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Version-Dependent ProgID:** Msxml2.DOMDocument.5.0, Msxml2.FreeThreadedDOMDocument.5.0

**Version-Dependent CLSID:** 88d969c0-f192-11d4-a65f-0040963251e5

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMDocument2 Members | DOMDocument
# IXMLDOMDocument2 Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from DOMDocument.

The methods and properties in this table are an extension to the World Wide Web Consortium (W3C) DOM.

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaces</td>
<td>Returns the list of namespaces in the document as XMLSchemaCache.</td>
</tr>
<tr>
<td>schemas</td>
<td>Finds schema documents during load.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProperty</td>
<td>Returns the current setting (or default) for second-level properties.</td>
</tr>
<tr>
<td>setProperty</td>
<td>Used for setting second-level properties.</td>
</tr>
<tr>
<td>validate</td>
<td>Performs run-time validation on the currently loaded document using the currently loaded document type definition (DTD), schema, or schema collection.</td>
</tr>
</tbody>
</table>

## Events

None.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMDocument2 | DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
**IXMLODMDocument3**

As an extension of the IXMLDOMDocument2 interface, the IXMLDOMDocument3 interface supports two additional methods, `importNode` and `validateNode`. The first one can be used to clone a node from an XML DOM object of different threading model. The second method performs run-time validation of a specified DOM fragment against the currently loaded document type definition (DTD), schema, or schema collection.

**Properties**

None.

**Methods**

- `importNode`
  Clones a node from a different DOM, so that it can be added later into the document using the `appendChild` method.

- `validateNode`*
  Validates a DOM fragment against the currently loaded DTD, schema, or schema collection.

  * Extension to the W3C DOM Specification.

**Events**

None.

**Remarks**

With an pointer to this interface, you can access all the methods and properties of the IXMLDOMDocument2 interface.

**Requirements**
Implementation: msxml5.dll

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Version-Dependent ProgID: Msxml2.DOMDocument.5.0, Msxml2.FreeThreadedDOMDocument.5.0

Version-Dependent CLSID: 88d969e5-f192-11d4-a65f-0040963251e5

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMDocument/DOMDocument | IXMLDOMDocument2
IXMLDOMDocumentFragment

A lightweight object that is useful for tree insert operations.

[Script]

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var docFragment;
xmlDoc.async = false;
xmlDoc.loadXML("<root/>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    docFragment = xmlDoc.createDocumentFragment();
    docFragment.appendChild(xmlDoc.createElement("node1"));
    docFragment.appendChild(xmlDoc.createElement("node2"));
    docFragment.appendChild(xmlDoc.createElement("node3"));
    alert(docFragment.xml);
    xmlDoc.documentElement.appendChild(docFragment);
    alert(xmlDoc.xml);
}
```

[Visual Basic]

Example

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
Dim docFragment As IXMLDOMDocumentFragment
xmlDoc.async = False
xmlDoc.loadXML "<root/>
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set docFragment = xmlDoc.createDocumentFragment()
    docFragment.appendChild xmlDoc.createElement("node1")
    docFragment.appendChild xmlDoc.createElement("node2")
    docFragment.appendChild xmlDoc.createElement("node3")
    MsgBox docFragment.xml
```
Example

The following sample creates and appends a new IXMLDOMDocumentFragment to the root document element. It uses the following XML document.

```
```

```c++
#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
{ if FAILED(_hr) throw(_hr); }

void XMLDOMDocumentFragmentSample()
{
  //HRESULT hr;
  try
  {
    IXMLDOMDocumentPtr docPtr;
    //init
    TESTHR(CoInitialize(NULL));
  }
  //end try
  catch ( HRESULT hr )
  {
    //catch
    //log
    TESTHR(CoUninitialize());
  }
  //end catch
}
```
TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));

// load a document
_variant_t varXml(_T("c:\\book.xml"));
_variant_t varOut((bool)TRUE);
varOut = docPtr->load(varXml);
if ((bool)varOut == FALSE)
{// show error description - IXMLDOMParseError sample
    IXMLDOMParseErrorPtr errPtr = docPtr->GetparseError();
    _bstr_t bstrErr(errPtr->reason);
    _tprintf(_T("Error:
"));
    _tprintf(_T("Code	=	0x%x
"), errPtr->errorCode);
    _tprintf(_T("Source	=	Line	:	%ld;	Char	:	%ld
"),
            errPtr->line,
            errPtr->linepos);
    _tprintf(_T("Error	Description	=	%s
"), (char*)bstrErr);
}
else
{// dom fragment sample

    // create a new node and add to the doc
    _variant_t varTyp((short)NODE_ELEMENT);
    _bstr_t varName(_T("BOOK"));
    IXMLDOMNodePtr nodePtr = docPtr->createNode(varTyp,
    // create a doc fragment and associate the new
    IXMLDOMDocumentFragmentPtr fragPtr = docPtr->createDocumentFragment();
    fragPtr->appendChild(nodePtr);
    // add some elements to the new node
    varName = _T("TITLE");
    IXMLDOMNodePtr nodePtr1 = docPtr->createNode(var
    nodePtr1->text = _T("Creepy Crawlies");
    varName = "AUTHOR";
    IXMLDOMNodePtr nodePtr2 = docPtr->createNode(var
    nodePtr2->text = _T("Stefan Knorr");
    nodePtr->appendChild(nodePtr1);
    nodePtr->appendChild(nodePtr2);
    // display the fragment contents
    MessageBox(NULL, fragPtr->xml, ", MB_OK);
    // add the fragment to the original doc
    docPtr->documentElement->appendChild(fragPtr);
    // display the modified doc contents
    MessageBox(NULL, docPtr->xml, ", MB_OK);
}
}
catch (_com_error &e)
{
    _tprintf(_T("Error:\n"));
    _tprintf(_T("Code = %08lx\n"), e.Error());
    _tprintf(_T("Code meaning = %s\n"), (char*) e.ErrorMes
Remarks

The DocumentFragment object can represent a fragment of a document or portion of a document's tree. This makes it useful when implementing end user commands that allow users to rearrange a document, such as cutting and pasting.

The DocumentFragment node has special, defined behavior for IXMLDOMNode insert operations that makes it especially convenient for developers. When an IXMLDOMDocumentFragment is inserted into a DOMDocument node (or other node that can take children), the children of the DocumentFragment are inserted into the node rather than the DocumentFragment itself. This makes the DocumentFragment useful when the user wants to create nodes that are siblings; the DocumentFragment acts as the parent of these nodes so that the user can employ the standard methods from the IXMLDOMNode interface, such as insertBefore and appendChild.

The children of a DocumentFragment node make up zero or more nodes representing the tops of any subtrees defining the structure of the document. DocumentFragment nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a DocumentFragment might have only one child, and that child node could be a Text node. Such a structure model represents neither an HTML document nor a well-formed XML document.

IXMLDOMDocumentFragment has no unique members of its own, but exposes the same members as IXMLDOMNode.

Versioning
MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

appendChild Method | insertBefore Method | IXMLDOMDocumentFragment Members | IXMLDOMNode | DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**IXMLDOMDocumentFragment Members**

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>basename*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of the node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
</tbody>
</table>

Specifies the XML Document Object Model (DOM) node
**nodeType**
Type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**
Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**
Returns the node type in string form. Read-only.

**nodeValue**
Contains the text associated with the node. Read/write.

**ownerDocument**
Returns the root of the document that contains the node. Read-only.

**parentNode**
Contains the parent node. Read-only.

**parsed**
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**
Returns the namespace prefix. Read-only.

**previousSibling**
Contains the previous sibling of the node in the parent’s child list. Read-only.

**specified**
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**text**
Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**
Appends a new child as the last child of the node.

**cloneNode**
Clones a new node.

**hasChildNodes**
Provides a fast way to determine whether a node has children.

**insertBefore**
Inserts a child node to the left of the specified node or at the end of the list.

**removeChild**
Removes the specified child node from the list of children and returns it.
Replaces the specified old child node with the
**replaceChild**  
supplied new child node.

**selectNodes**  
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode**  
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**transformNode**  
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject**  
Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button 🗼 in the upper-left corner of the page.

**See Also**

IXMLDOMDocumentFragment | IXMLDOMNode
IXMLDOMDocumentType

Contains information associated with the document type declaration.

[C/C++]

Example

The following C/C++ example displays the doctype name of the document.

The following document was used.

```xml
<?xml version='1.0'?>
<!DOCTYPE COLLECTION [ 
<!ELEMENT COLLECTION (DATE?, BOOK+) >
<!ATTLIST COLLECTION
xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">
<!ELEMENT BOOK (TITLE, AUTHOR, PUBLISHER) >
<!ELEMENT DATE (#PCDATA) >
<!ELEMENT TITLE (#PCDATA) >
<!ELEMENT AUTHOR (#PCDATA) >
<!ELEMENT PUBLISHER (#PCDATA) > ]>
.COLLECTION
xmlns:dt="urn:schemas-microsoft-com:datatypes">
<BOOK>
<TITLE>Lover Birds</TITLE>
<AUTHOR>Cynthia Randall</AUTHOR>
<PUBLISHER>Lucerne Publishing</PUBLISHER>
</BOOK>
<BOOK>
<TITLE>The Sundered Grail</TITLE>
<AUTHOR>Eva Corets</AUTHOR>
<PUBLISHER>Lucerne Publishing</PUBLISHER>
</BOOK>
<BOOK>
<TITLE>Splish Splash</TITLE>
<AUTHOR>Paula Thurman</AUTHOR>
<PUBLISHER>Scootney</PUBLISHER>
</BOOK>
</COLLECTION>

#import "msxml5.dll"
```
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
{ if FAILED(_hr) throw(_hr); }

This is the C/C++ sample:

void XMLDOMDocumentTypeSample()
{
  try
  {
    IXMLDOMDocumentPtr docPtr;

    //init
    TESTHR(CoInitialize(NULL));
    TESTHR(docPtr.CreateInstance(_T("Msxml2.DOMDocument.5.0")));

    // Load a document.
    _variant_t varXml(_T("D:\xmlSample\xmlTest\book1.xml"));
    _variant_t varOut((bool)TRUE);
    varOut = docPtr->load(varXml);
    if ((bool)varOut == FALSE)
    {// Show error description - IXMLDOMParseError sample.
      IXMLDOMParseErrorPtr errPtr = docPtr->GetparseError();
      // Print error details.
    }
    else
    {
      IXMLDOMDocumentTypePtr docTypPtr = docPtr->doctype;
      if (docTypPtr)
      {
        _bstr_t bstrTyp(docTypPtr->name);
    _tprintf(_T("Document type name = %s\n"), (TCHAR*)bstrTyp);
      }
    }
  } catch (_com_error &e)
  { _tprintf(_T("Error:\n"));
    _tprintf(_T("Code = %08lx\n"), e.Error());
    _tprintf(_T("Code meaning = %s\n"), (TCHAR*) e.ErrorMessage());
    _tprintf(_T("Source = %s\n"), (TCHAR*) e.Source());
    _tprintf(_T("Error Description = %s\n"), (TCHAR*) e.Description());
  }
  catch(....)
  { _tprintf(_T("Unknown error!"));
  }
  CoUninitialize();
Remarks

Each DOMDocument includes a doctype that identifies the document's IXMLDOMDocumentType. The IXMLDOMDocumentType provides access to the list of entities and notations defined for the document.

Versioning

MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

doctype Property | DOMDocument | IXMLDOMDocumentType Members
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMDocumentType Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

Properties

- **attributes**: Contains the list of attributes for this node. Read-only.
- **nodeName**: Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.
- **baseName**: Returns the base name for the name qualified with the namespace. Read-only.
- **childNodes**: Contains a node list containing the children nodes. Read-only.
- **dataType**: Specifies the data type for this node. Read/write.
- **definition**: Returns the definition of the node in the document type definition (DTD) or schema. Read-only.
- **entities**: Contains a list of the entities declared in the DOCTYPE declaration. Read-only.
- **firstChild**: Contains the first child of the node. Read-only.
- **lastChild**: Returns the last child node. Read-only.
- **namespaceURI**: Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.
- **nextSibling**: Contains the next sibling of this node in the parent's child list. Read-only.
**nodeType**
Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**
Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**
Returns the node type in string form. Read-only.

**nodeValue**
Contains the text associated with the node. Read/write.

**notations**
Contains a list of the `IXMLDOMNode` objects present in the document type declaration. Read-only.

**ownerDocument**
Returns the root of the document that contains the node. Read-only.

**parentNode**
Contains the parent node. Read-only.

**parsed**
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**
Returns the namespace prefix. Read-only.

**previousSibling**
Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema. Read-only.

**text**
Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**
Appends a new child as the last child of the node.

**cloneNode**
Clones a new node.

**hasChildNodes**
Provides a fast way to determine whether a node has children.

**insertBefore**
Inserts a child node to the left of the specified node or at the end of the list.
removeChild

Removes the specified child node from the list of children and returns it.

replaceChild

Replaces the specified old child node with the supplied new child node.

selectNodes*

Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode*

Applies the specified pattern-matching operation to this node's context and returns the first matching node.

transformNode*

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNodeToObject*

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet, and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMDocumentType | IXMLDOMNode | IXMLDOMNotation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMElement

 Represents the element object.

[Script]

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var newRoot;
newRoot = xmlDoc.createElement("root");
xmlDoc.documentElement = newRoot;
alert(xmlDoc.xml);

[Visual Basic]

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim newRoot As IXMLDOMElement
Set newRoot = xmlDoc.createElement("root")
Set xmlDoc.documentElement = newRoot
MsgBox xmlDoc.xml

[C/C++]

Example

The following C/C++ example displays the entire XML document from its root element.

#include "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
    { if FAILED(_hr) throw(_hr); }

void XMLDOMElementSample()
{
    try {

IXMLDOMDocumentPtr docPtr;
IXMLDOMElementPtr ElementPtr;

//init
TESTHR(CoInitialize(NULL));
TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));

// Load a document.
_variant_t varXml("C:\book.xml");
_variant_t varOut((bool)TRUE);
varOut = docPtr->load(varXml);
if ((bool)varOut == FALSE)
    throw(0);
ElementPtr = docPtr->documentElement;
MessageBox(NULL, ElementPtr->xml, _T("Document from its root"))
} catch(...) {
    MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK)
} CoUninitialize();

Remarks

Element nodes are among the most common objects in the XML document tree. Element nodes can have attributes associated with them. By definition, attributes are not defined as child nodes of an element and are not considered to be part of the document tree. Accordingly, the IXMLDOMElement object provides methods to make it easier to manage attributes, including methods to associate an attribute with an element and to retrieve an attribute object and the attribute value by name.

To retrieve the set of all attributes associated with an element, you can also access the attributes property, which returns an IXMLDOMNamedNodeMap collection object that contains all the element's attributes.

Versioning

MSXML 2.0 and later

Requirements
**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Version-Dependent ProgID:** Msxml2.DOMDocument.5.0

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

getAttribute Method | IXMLDOMNodeNamedNodeMap | IXMLDOMElementMembers | IXMLDOMText | IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMElement Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

Properties

attributes
Contains the list of attributes for this node. Read-only.

baseName*
Returns the base name for the name qualified with the namespace. Read-only.

childNodes
Contains a node list containing the children nodes. Read-only.

dataType*
Specifies the data type for this node. Read/write.

definition*
Returns the definition of the node in the document type definition (DTD) or schema.

firstChild
Contains the first child of the node. Read-only.

lastChild
Returns the last child node. Read-only.

namespaceURI*
Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.

nextSibling
Contains the next sibling of this node in the parent's child list. Read-only.

nodeName
Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.

nodeType
Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.
**nodeTypeString**
Returns the node type in string form. Read-only.

**nodeValue**
Contains the text associated with the node. Read/write.

**ownerDocument**
Returns the root of the document that contains the node. Read-only.

**parentNode**
Contains the parent node. Read-only.

**parsed**
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**
Returns the namespace prefix. Read-only.

**previousSibling**
Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema. Read-only.

**tagName**
Contains the element name. Read-only.

**text**
Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**
Appends a new child as the last child of this node.

**cloneNode**
Clones a new node.

**getAttribute**
Gets the value of the attribute.

**getAttributeNode**
Gets the attribute node.

**getElementsByTagName**
Returns a list of all descendant elements that match the supplied name.

**hasChildNodes**
Provides a fast way to determine whether a node has children.

**insertBefore**
Inserts a child node to the left of the specified node or at the end of the list.

**normalize**
Normalizes all descendant elements by combining two or more adjacent text nodes into one unified text
removeAttribute

Removes or replaces the named attribute.

removeAttributeNode

Removes the specified attribute from this element.

removeChild

Removes the specified child node from the list of children and returns it.

replaceChild

Replaces the specified old child node with the supplied new child node.

selectNodes*

Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode*

Applies the specified pattern-matching operation to this node's context and returns the first matching node.

setAttribute

Sets the value of the named attribute.

setAttributeNode

Sets or updates the supplied attribute node on this element.

transformNode*

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNodeToObject*

Processes this node and its children using the supplied (XSLT) style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMElement | IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMEntity

Represents a parsed or unparsed entity in the XML document.

[C/C++]

Example

BOOL DOMEntityNotationName()
{
  BOOL bResult = FALSE;
  IXMLDOMEntity *pIXMLDOMEntity = NULL;
  BSTR bstrName ;
  HRESULT hr;

  try
  {
    pIXMLDOMEntity = GetFirstEntity();

    if(pIXMLDOMEntity)
    {
      hr =  pIXMLDOMEntity->get_notationName(&bstrName);
      ::MessageBox(NULL, bstrName, _T("Notation name"), MB_OK);
      bResult = TRUE;
      ::SysFreeString(bstrName);
      pIXMLDOMEntity->Release();
    }
  }
  catch(...)
  {
    if(pIXMLDOMEntity)
    {pIXMLDOMEntity->Release();
       DisplayErrorToUser();
    }
    return bResult;
  }

  IXMLDOMEntity* GetFirstEntity()
  {
    IXMLDOMNodeMap *pIXMLDOMNamedNodeMapEntities = NULL;
    IXMLDOMEntity *pIXMLDOMEntity = NULL;
    IXMLDOMNode *pIXMLDOMNode = NULL;
    IXMLDOMDocument *pIXMLDOMDocument = NULL;
    HRESULT hr;
  }
}
try {
    // Create an instance of DOMDocument and initialize
    // pIXMLDOMDocument.
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_doctype(&pIXMLDOMDocumentType);
    SUCCEEDED(hr) ? 0 : throw hr;

    if(pIXMLDOMDocumentType)
    {
        hr = pIXMLDOMDocumentType->get_entities
            (&pIXMLDOMNamedNodeMapEntities);
        if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMapEntities)
        {
            hr = pIXMLDOMNamedNodeMapEntities->nextNode(&pIXMLDOMNode);
            if(SUCCEEDED(hr) && pIXMLDOMNode)
            {
                hr = pIXMLDOMNode->QueryInterface(IID_IXMLDOMEntity,
                    (void**)&pIXMLDOMEntity);
                SUCCEEDED(m_hr) ? 0 : throw m_hr;
                pIXMLDOMNode->Release();
            }
            pIXMLDOMNamedNodeMapEntities->Release();
        }
        pIXMLDOMDocumentType->Release();
    }
}
catch(...){
    if(pIXMLDOMNode)
        pIXMLDOMNode->Release();
    if(m_pIXMLDOMNamedNodeMapEntities)
        m_pIXMLDOMNamedNodeMapEntities->Release();
    if(m_pIXMLDOMDocumentType)
        m_pIXMLDOMDocumentType->Release();
    DisplayErrorToUser();
}

// Release pIXMLDOMEntity when finished using it.
// Release pIXMLDOMDocument when finished using it.
return pIXMLDOMEntity;
}

Remarks

IXMLDOMEntity represents the entity itself rather than the entity declaration. The World Wide Web Consortium (W3C) Document Object Model (DOM) does not currently define an object that models the entity declaration.
According to the W3C DOM specification, an XML parser can completely expand entity references into entities before the structure model is passed to the DOM. When these entity references are expanded, the document tree does not contain any entity references.

When Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office validates the XML document, it expands external entities (except binary entities). The nodes representing the expanded entity are available as read-only children of the entity reference. The Microsoft implementation does not expand these entities when it is not validating.

The `nodeName` property contains the name of the entity.

The structure of the entity child list is exactly the same as the structure of the child list for the `IXMLDOMEntityReference` object with the same `nodeName` value.

Level 1 of the W3C DOM application programming interface (API) does not define a way to change entity nodes. (All `IXMLDOMEntity` object properties are read-only.)

**Versioning**

MSXML 2.0 and later

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

nodeName Property | IXMLDOMEntityReference | IXMLDOMEntity Members | DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMEntity Members

[Visual Basic, Script]
The following tables show the properties, methods, and events.

[C/C++]
The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

Properties

- **attributes**
  - Contains the list of attributes for this node. Read-only.

- **baseName**
  - Returns the base name for the name qualified with the namespace. Read-only.

- **childNodes**
  - Contains a node list containing the children nodes. Read-only.

- **dataType**
  - Specifies the data type for this node. Read-only.

- **definition**
  - Returns the definition of the node in the document type definition (DTD) or schema. Read-only.

- **firstChild**
  - Contains the first child of the node. Read-only.

- **lastChild**
  - Returns the last child node. Read-only.

- **namespaceURI**
  - Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.

- **nextSibling**
  - Contains the next sibling of this node in the parent's child list. Read-only.

- **nodeName**
  - Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.

- **nodeType**
  - Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.
nodeTypedValue* Contains this node's value, expressed in its defined data type. Read-only.
nodeTypeString* Returns the node type in string form. Read-only.
nodeValue Contains the text associated with the node. Read-only.
notationName Contains the notation name. Read-only.
ownerDocument Returns the root of the document that contains the node. Read-only.
parentNode Contains the parent node. Read-only.
parsed* Indicates the parsed status of the node and child nodes. Read-only.
prefix* Returns the namespace prefix. Read-only.
previousSibling Contains the previous sibling of the node in the parent's child list. Read-only.
publicId Contains the public identifier associated with the entity. Read-only.
specified* Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.
systemId Contains the system identifier associated with the entity. Read-only.
text* Contains the text content of the node or the concatenated text representing the node and its descendants. Read-only.
xml* Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

Methods

appendChild Appends a new child node as the last child of the node.
cloneNode Clones a new node.
hasChildNodes Provides a fast way to determine whether a node has children.
Inserts a child node to the left of the specified node or
**insertBefore** at the end of the list.

**removeChild** Removes the specified child node from the list of children and returns it.

**replaceChild** Replaces the specified old child node with the supplied new child node.

**selectNodes**
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode**
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**transformNode**
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject**
Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMEntity | IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMEntityReference

Represents an entity reference node.

[C/C++]

IXMLDOMEntityReference has no unique properties or methods of its own, but exposes the same objects and properties as the IXMLDOMNode object.

[Script]

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root, nodeEntityReference;
xmlDoc.async = false;
xmlDoc.loadXML("<root/>");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  nodeEntityReference = xmlDoc.createEntityReference("nbsp");
  root = xmlDoc.documentElement;
  root.appendChild(nodeEntityReference);
  alert(xmlDoc.xml);
}
```

[Visual Basic]

Example

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim nodeEntityReference As IXMLDOMEntityReference
xmlDoc.async = False
xmlDoc.loadXML "<root/>"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set nodeEntityReference = xmlDoc.createEntityReference("nbsp")
```
Set root = xmlDoc.documentElement
root.appendChild nodeEntityReference
MsgBox xmlDoc.xml
End If

**Remarks**

If the XML parser expands entity references while building the structure model, no **EntityReference** objects appear in the tree.

XML does not require non-validating processors to handle external entity declarations (those made in the external subset or declared in external parameter entities). This means that parsed entities declared in the external subset need not be expanded, and therefore the replacement value of the entity might not be available. If available, the replacement value of the referenced entity appears in the child list of **EntityReference**.

The resolution of the children of the **EntityReference** (the replacement value of the referenced entity) can be evaluated. Actions such as calling the **childNodes** property are assumed to trigger the evaluation. Character entities are expanded by the XML parser and do not appear as entity references, but instead appear within the character text in Unicode.

**Versioning**

MSXML 2.0 and later

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**
childNodes Property  |  IXMLDOMNode  |  IXMLDOMEntityReference Members
|  IXMLDOMEntity  |  DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMEntityReference
Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode interface.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of this node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
</tbody>
</table>

Specifies the XML Document Object Model (DOM) node
**nodeType**

Type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**

Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**

Returns the node type in string form. Read-only.

**nodeValue**

Contains the text associated with the node. Read/write.

**ownerDocument**

Returns the root of the document that contains the node. Read-only.

**parentNode**

Contains the parent node. Read-only.

**parsed**

Indicates the parsed status of the node and child nodes. Read-only.

**prefix**

Returns the namespace prefix. Read-only.

**previousSibling**

Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**

Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**text**

Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**

Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**

Appends a new child as the last child of this node.

**cloneNode**

Clones a new node.

**hasChildNodes**

Provides a fast way to determine whether a node has children.

**insertBefore**

Inserts a child node to the left of the specified node or at the end of the list.

**removeChild**

Removes the specified child node from the list of children and returns it.

Replaces the specified old child node with the
replaceChild

supplied new child node.

selectNodes*

Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode*

Applies the specified pattern-matching operation to this node's context and returns the first matching node.

transformNode*

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNodeToObject*

Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMEntityReference | IXMLDOMNode
IXMLDOMImplementation

Provides methods that are independent of any particular instance of the Document Object Model (DOM).

[Script]

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oImplementation;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    oImplementation = xmlDoc.implementation;
    alert(oImplementation.hasFeature("DOM", "1.0"));
}

[Visual Basic]

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim oImplementation As IXMLDOMImplementation
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set oImplementation = xmlDoc.implementation
    MsgBox oImplementation.hasFeature("DOM", "1.0")
End If

[C/C++]

Example
IXMLDOMImplementation *pIXMLDOMImplementation = NULL;
VARIANT_BOOL varbFlag;
BSTR bstrOutput = NULL;
BSTR bstrFeature = ::SysAllocString(_T("MS-DOM"));
HRESULT hr;
IXMLDOMDocument *pIXMLDOMDocument = NULL;

try {
    // Create an instance of DOMDocument and initialize pIXMLDOMDocument.
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_implementation(&pIXMLDOMImplementation);
    if(SUCCEEDED(hr) && pIXMLDOMImplementation) {
        hr = pIXMLDOMImplementation->hasFeature(bstrFeature, _T("1.0"));
        if(varbFlag == VARIANT_TRUE )
            bstrOutput = ::SysAllocString(_T("Feature Supported"));
        else
            bstrOutput = ::SysAllocString(_T("Feature not Supported"));

        ::MessageBox(NULL, bstrOutput, bstrFeature, MB_OK);
        ::SysFreeString(bstrOutput);
        bstrOutput = NULL;
        ::SysFreeString(bstrFeature);
        bstrFeature = NULL;
        pIXMLDOMImplementation->Release();
    }
}
catch(...) {
    if(bstrOutput)
        ::SysFreeString(bstrOutput);
    if(bstrFeature)
        ::SysFreeString(bstrFeature);
    if(pIXMLDOMImplementation)
        pIXMLDOMImplementation->Release();
    DisplayErrorToUser();
}

// Release pIXMLDOMDocument when finished using it.

Versioning

MSXML 2.0 and later

Requirements
Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMImplementation Members | DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMImplementation

Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IDispatch.

Properties

None.

Methods

hasFeature Indicates support for the specified feature.

Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMImplementation
IXMLDOMNodeMap

Adds support for namespaces and iteration through the collection of attribute nodes.

[Script]

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, namedNodeMap;
xmlDoc.async = false;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    namedNodeMap = nodeBook.attributes;
    alert(namedNodeMap.length);
}
```

[Visual Basic]

Example

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMNode
Dim namedNodeMap As IXMLDOMNodeMap
xmlDoc.async = False
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    Set namedNodeMap = nodeBook.Attributes
    MsgBox namedNodeMap.length
End If
```
Example

The following C/C++ example creates and appends a new attribute to the root document element.

```c++
#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
{
    if FAILED(_hr) throw(_hr);
}

void XMLDOMNamedNodeMap()
{
    try {
        IXMLDOMDocumentPtr docPtr;
        IXMLDOMNodePtr DOMNodePtr;
        IXMLDOMNamedNodeMapPtr DOMNamedNodeMapPtr;

        //init
        TESTHR(CoInitialize(NULL));
        TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));

        VARIANT vtTemp;
        vtTemp.vt=VT_I2;
        vtTemp.iVal = MSXML2::NODE_ATTRIBUTE;

        // load a document
        _variant_t varXml("C:\book.xml");
        _variant_t varOut((bool)TRUE);
        varOut = docPtr->load(varXml);
        if ((bool)varOut == FALSE)
            throw(0);

        DOMNodePtr = docPtr->createNode(vtTemp, "Sci-Fi", "");
        DOMNamedNodeMapPtr = docPtr->documentElement->attributes;
        DOMNamedNodeMapPtr->setNamedItem(DOMNodePtr);

        MessageBox(NULL, _bstr_t(docPtr->xml), _T("New Document"), MB_OK);
    } catch(...) {
        MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK)
    }
    CoUninitialize();
}
Remarks

IXMLDOMNamedNodeMap is a node collection that allows access by name as well as by index. This collection is typically used for attributes.

The World Wide Web Consortium (W3C) Document Object Model (DOM) definition does not require NamedNodeMap objects to be maintained in any particular order. Objects contained in a NamedNodeMap can also be accessed by an ordinal index, but this is simply to allow convenient enumeration and does not imply that the DOM specifies an order for these nodes. DOM implementations are not required to preserve the node order.

The Microsoft® implementation preserves the attributes in the order in which they appear in the source. Additional attributes are added to the end of the list. A namespace attribute, xmlns, is inserted as the first in the list if the tag name requires the namespace, or immediately before the attribute that uses an undeclared namespace.

Like the node list, however, a named node map collection is live; that is, the addition and removal of nodes, and changes within nodes, are immediately reflected in the collection. This means that two successive requests for items using the same index can return two different items, depending on changes to the collection. This also means that changes to the node objects are immediately available in the nodes obtained from the list.

When a default value is specified in the schema for an attribute and that attribute is removed, it is automatically regenerated with the default value, as specified in the W3C standard. The attribute reappears immediately because the collection is live, with a corresponding change to the count of items in the collection. When removing an attribute with such a default setting, you must again retrieve the number of items in the collection to obtain an accurate count. You should also reset any index into the collection.

Versioning

MSXML 2.0 and later
Requirements

**Implementation:**  msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:**  msxml2.h, msxml2.idl

**Version-Dependent ProgID:**  Msxml2.DOMDocument.5.0, Msxml2.FreeThreadedDOMDocument.5.0

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNamedNodeMap Members  |  IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMNamedNodeMap

Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IDispatch.

Properties

length  Indicates the number of items in the collection. Read-only.

Methods

getNamedItem  Retrieves the attribute with the specified name.

getQualifiedItem*  Returns the attribute with the specified namespace and attribute name.

item  Allows random access to individual nodes within the collection. Read-only.

nextNode*  Returns the next node in the collection.

removeNamedItem  Removes an attribute from the collection.

removeQualifiedItem  Removes the attribute with the specified namespace and attribute name.

reset  Resets the iterator.

setNamedItem  Adds the supplied node to the collection.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.
Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNamedNodeMap
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**IXMLDOMNode**

Extends the core node with support for data types, namespaces, document type definitions (DTDs), and schemas.

[Script]

**Example**

The following script example creates an IXMLDOMNode, `currNode`, and displays its XML.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(1);
    alert(currNode.xml);
}
```

[Visual Basic]

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMNode, `currNode`, and displays its XML.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(1)
```
The following C/C++ example creates and appends a new node to the document root.

```c++
#include <tchar.h>
#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
{ if FAILED(_hr) throw(_hr); }

void XMLDOMNodeSample()
{
    try {
        IXMLDOMDocumentPtr docPtr;
        IXMLDOMNodePtr DOMNodePtr;

        //init
        TESTHR(CoInitialize(NULL));
        TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));

        VARIANT vtTemp;
        vtTemp.vt=VT_I2;
        vtTemp.iVal = 1; //NODE_ELEMENT
        //load a document
        _variant_t varXml("c:\Temp\books.xml");
        _variant_t varOut((bool)TRUE);
        varOut = docPtr->load(varXml);
        if ((bool)varOut == FALSE)
            throw(0);
        MessageBox(NULL, _bstr_t(docPtr->xml), _T("Original Document"), MB_OK);
        DOMNodePtr = docPtr->createNode(vtTemp, "VIDEOS", "");
        docPtr->documentElement->appendChild(DOMNodePtr);
        MessageBox(NULL, _bstr_t(docPtr->xml), _T("New Document"), MB_OK);
    }
    catch(...) {
        MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK); // error message
    }
    CoUninitialize();
```
int main()
{
    XMLDOMNodeSample();
    return 0;
}

Versioning

MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode Members | IXMLDOMNodeList
IXMLDOMNode Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IDispatch.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of this node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of this node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Returns the qualified name for attribute, document type, element, entity, or notation nodes. Returns a fixed string for all other node types.</td>
</tr>
<tr>
<td>nodeType</td>
<td>Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.</td>
</tr>
</tbody>
</table>
**nodeTypedValue**
Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**
Returns the node type in string form. Read-only.

**nodeValue**
Contains the text associated with the node.
Read/write.

**ownerDocument**
Returns the root of the document that contains the node. Read-only.

**parentNode**
Contains the parent node. Read-only.

**parsed**
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**
Returns the namespace prefix. Read-only.

**previousSibling**
Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**text**
Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**
Appends a new child as the last child of the node.

**cloneNode**
Clones a new node.

**hasChildNodes**
Provides a fast way to determine whether a node has children.

**insertBefore**
Inserts a child node to the left of the specified node or at the end of the list.

**removeChild**
Removes the specified child node from the list of children and returns it.

**replaceChild**
Replaces the specified old child node with the
supplied new child node.

**selectNodes***
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode***
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**transformNode***
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject***
Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
IXMLDOMNodeList

Supports iteration through the live collection, in addition to indexed access.

[Script]

Example

The following script example creates an IXMLDOMNodeList object by using the document's getElementsByTagName method.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    objNodeList = xmlDoc.getElementsByTagName("author");
    alert(objNodeList.length);
    for (var objNode in objNodeList)
        alert(objNode.xml);
}
```

[Visual Basic]

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNodeList object by using the document's getElementsByTagName method, and examines the content of each node.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLDOMNodeList
Dim objNode As IXMLDOMNode
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
```
Set objNodeList = xmlDoc.getElementsByTagName("author")
MsgBox objNodeList.length
For Each objNode In objNodeList
    MsgBox objNode.xml
Next
End If

[C/C++]

Example

The following C/C++ example gets all elements with node name AUTHOR and displays the number of nodes found with the following sample.

```xml
<?xml version='1.0'?><COLLECTION xmlns:dt="urn:schemas-microsoft-com:datatypes"><DATE dt:dt="datetime">1998-10-13T15:56:00</DATE><BOOK>
    <TITLE>Lover Birds</TITLE>
    <AUTHOR>Cynthia Randall</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
</BOOK><BOOK>
    <TITLE>The Sundered Grail</TITLE>
    <AUTHOR>Eva Corets</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
</BOOK><BOOK>
    <TITLE>Splish Splash</TITLE>
    <AUTHOR>Paula Thurman</AUTHOR>
    <PUBLISHER>Scootney</PUBLISHER>
</BOOK></COLLECTION>
```

#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
{ if FAILED(_hr) throw(_hr); }

void XMLDOMNodeListSample()
{
    try {
        IXMLDOMDocumentPtr docPtr;
        IXMLDOMNodeListPtr NodeListPtr;
        IXMLDOMNodePtr DOMNodePtr;
```
//init
TESTHR(CoInitialize(NULL));
TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));

// load a document
_variant_t varXml("C:\book.xml");
_variant_t varOut((bool)TRUE);
varOut = docPtr->load(varXml);
if ((bool)varOut == FALSE)
    throw(0);
NodeListPtr = docPtr->getElementsByTagName("AUTHOR");
MessageBox(NULL, _bstr_t(NodeListPtr->length), _T("Node List length"), MB_OK);
}
catch(...)
{
    MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK);
}
CoUninitialize();

Remarks

A NodeList collection is live; that is, the addition and removal of nodes, and changes within nodes, are immediately reflected in the collection. This means that two successive requests for items using the same index can return two different items, depending on changes to the collection. This also means that changes to the node objects are immediately available in the nodes obtained from the list.

Versioning

MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNodeList Members | IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMNodeList Members

[Visual Basic, Script]
The following tables show the properties, methods, and events.

[C/C++]
The following tables show the properties, methods, and events.

This interface inherits from IDispatch.

**Properties**

- **length**
  Indicates the number of items in the collection. Read-only.

**Methods**

- **item**
  Allows random access to individual nodes within the collection.

- **nextNode***
  Returns the next node in the collection.

- **reset***
  Resets the iterator.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**
IXMLDOMNodeList
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMNotation

Contains a notation declared in the document type definition (DTD) or schema.

[C/C++]

The following C/C++ example displays the names of the notation nodes in the DTD declaration of the document.

The following document was used.

```xml
<?xml version='1.0'?>
<!DOCTYPE COLLECTION [ 
<!NOTATION XLS PUBLIC "http://www.microsoft.com/office/excel/">
<!ELEMENT COLLECTION (DATE?, BOOK+)>
<!ATTLIST COLLECTION
 xmlns:dt CDATA #FIXED "urn:schemas-microsoft-com:datatypes">
<!ELEMENT BOOK(TITLE, AUTHOR, PUBLISHER)>
<!ELEMENT DATE(#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
]>
<COLLECTION
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <BOOK>
  <TITLE>Lover Birds</TITLE>
  <AUTHOR>Cynthia Randall</AUTHOR>
  <PUBLISHER>Lucerne Publishing</PUBLISHER>
 </BOOK>
 <BOOK>
  <TITLE>The Sundered Grail</TITLE>
  <AUTHOR>Eva Corets</AUTHOR>
  <PUBLISHER>Lucerne Publishing</PUBLISHER>
 </BOOK>
 <BOOK>
  <TITLE>Splish Splash</TITLE>
  <AUTHOR>Paula Thurman</AUTHOR>
  <PUBLISHER>Scootney</PUBLISHER>
 </BOOK>
</COLLECTION>
```

# import "msxml5.dll"
using namespace MSXML2;
inline void TESTHR( HRESULT _hr )
{
    if FAILED(_hr) throw(_hr);
}

void XMLDOMNotationSample()
{
    try
    {
        IXMLDOMDocumentPtr docPtr;

        //init
        TESTHR(CoInitialize(NULL));
        TESTHR(docPtr.CreateInstance(_T("Msxml2.DOMDocument.5.0")));

        // load a document
        _variant_t varXml("D:\xmlSample\xmlTest\book1.xml");
        _variant_t varOut((bool)TRUE);
        varOut = docPtr->load(varXml);
        if ((bool)varOut == FALSE)
        { // show error description - IXMLDOMParseError sample
            IXMLDOMParseErrorPtr errPtr = docPtr->GetparseError();

            // display error information
        }
        else
        {
            IXMLDOMDocumentTypePtr docTypPtr = docPtr->doctype;
            if (docTypPtr)
            {
                int nCounter;
                IXMLDOMNodeNamedNodeMapPtr XMLDOMNodeNamedNodeMapPtr =
                docTypPtr->Getnotations();

                for(nCounter=0;nCounter < XMLDOMNodeNamedNodeMapPtr->length;nCounter++)
                {
                    IXMLDOMNodePtr XMLDOMNodePtr =
                    XMLDOMNodeNamedNodeMapPtr->Getitem(nCounter);
                    _tprintf(_T("%s"), (TCHAR*)XMLDOMNodePtr->nodeName);
                }
            }
        }
    }
    catch (_com_error &e)
    {
        _tprintf(_T("Error:\n"));
        _tprintf(_T("Code = %08lx\n"), e.Error());
        _tprintf(_T("Code meaning = %s\n"), (TCHAR*)e.ErrorMessage());
        _tprintf(_T("Source = %s\n"), (TCHAR*)e.Source());
    }
Remarks

Notations identify by name the format of unparsed entities, the format of elements that bear a notation attribute, or the application to which a processing instruction is addressed.

The nodeName property contains the notation name.

Versioning

MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

dnodeName Property | IXMLDOMNode | IXMLDOMNotation Members
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMNodeNotation Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of this node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
<tr>
<td>nodeType</td>
<td>Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.</td>
</tr>
</tbody>
</table>
**nodeTypedValue**  
Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**  
Returns the node type in string form. Read-only.

**nodeValue**  
Contains the text associated with the node. Read/write.

**ownerDocument**  
Returns the root of the document that contains the node. Read-only.

**parentNode**  
Contains the parent node. Read-only.

**parsed**  
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**  
Returns the namespace prefix. Read-only.

**previousSibling**  
Contains the previous sibling of the node in the parent's child list. Read-only.

**publicId**  
Contains the public identifier for the notation. Read-only.

**specified**  
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**systemId**  
Contains the system identifier for the notation. Read-only.

**text**  
Represents the text content of the node or the concatenated text representing the node and its descendants.

**xml**  
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**  
Appends a new child as the last child of this node.

**cloneNode**  
Clones a new node.

**hasChildNodes**  
Provides a fast way to determine whether a node has children.

**insertBefore**  
Inserts a child node to the left of the specified node or at the end of the list.

**removeChild**  
Removes the specified child node from the list of children and returns it.

Replaces the specified old child node with the
**replaceChild**

supplied new child node.

**selectNodes**

Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode**

Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**transformNode**

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject**

Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMNode | IXMLDOMNotation
**IXMLDOMParseError**

Returns detailed information about the last parse error, including the error number, line number, character position, and a text description.

**[Script]**

**Example**

The following script example attempts to load an XML document. It then tests the errorCode property of the IXMLDOMParseError object to see if an error has occurred.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert("A parse error occurred.");
} else {
    alert(xmlDoc.documentElement.xml);
}
```

**[Visual Basic]**

**Example**

The following Microsoft® Visual Basic® example attempts to load an XML document. It then tests the errorCode property of the IXMLDOMParseError object to see if an error has occurred.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
    MsgBox "A parse error occurred."
Else
    MsgBox xmlDoc.documentElement.xml
End If
```

**[C/C++]**
Example

The following C/C++ example reads a document with a missing end "TITLE" tag (in the second book) and displays the error.

```c
#import "msxml5.dll"
using namespace MSXML2;

inline void TESTHR( HRESULT _hr )
{ if FAILED(_hr) _com_issue_error(_hr); }

void XMLDOMParserSample()
{
  try
  {
    IXMLDOMDocumentPtr docPtr;
    //init
    TESTHR(CoInitialize(NULL));
    TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0"));
    // load a document
    _variant_t varXml("D:\xmlSample\xmlTest\book.xml");
    _variant_t varOut((bool)TRUE);
    varOut = docPtr->load(varXml);
    if ((bool)varOut == FALSE)
    {
      IXMLDOMParseErrorPtr errPtr = docPtr->GetparseError();
      _bstr_t bstrErr(errPtr->reason);
      printf("Error:\n");
      printf("Code = 0x%lx\n", errPtr->errorCode);
      printf("Source = Line : %ld; Char : %ld\n", errPtr->line, errPtr->linepos);
      printf("Error Description = %s\", (char*)bstrErr);
    }
    else
    {
      printf("Load successful");
    }
  }
  catch (_com_error &e)
  {
    printf("Error:\n");
    printf("Code = %08lx\n", e.Error());
    printf("Code meaning = %s\", (char*) e.ErrorMessage());
    printf("Source = %s\", (char*) e.Source());
  }
```
printf("Error Description = %s\n", (char*) e.Description());
}   
catch(...)
{
  printf("Unknown error!\n");
}
CoUninitialize();

Document

<?xml version='1.0'?>
<COLLECTION 
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <DATE dt:dt="datetime">1998-10-13T15:56:00</DATE>
  <BOOK>
    <TITLE>Lover Birds</TITLE>
    <AUTHOR>Cynthia Randall</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
  </BOOK>
  <BOOK>
    <TITLE>The Sundered Grail</TITLE>
    <AUTHOR>Eva Corets</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
  </BOOK>
  <BOOK>
    <TITLE>Splish Splash</TITLE>
    <AUTHOR>Paula Thurman</AUTHOR>
    <PUBLISHER>Scootney</PUBLISHER>
  </BOOK>
</COLLECTION>

Output

Error:
Code = 0xc00ce56d
Source = Line : 14; Char : 5
Error Description = End tag 'BOOK' does not match the start tag 'TITLE'

Versioning

MSXML 2.0 and later

Requirements
**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**

[IXMLDOMParseError Members]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMParseError Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IDispatch.

**Properties**

- **errorCode**
  Contains the error code of the last parse error. Read-only.
- **filepos**
  Contains the absolute file position where the error occurred. Read-only.
- **line**
  Specifies the line number that contains the error. Read-only.
- **linepos**
  Contains the character position within the line where the error occurred. Read-only.
- **reason**
  Describes the reason for the error. Read-only.
- **srcText**
  Returns the full text of the line containing the error. Read-only.
- **url**
  Contains the URL of the XML document containing the last error. Read-only.

**Methods**

None.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMParseError
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
The **IXMLDOMParseError2** interface extends the **IXMLDOMParseError** interface and provides enhanced error handling functionality. With **IXMLDOMParseError2**, you can:

- Retrieve and inspect all errors and warnings found during validation.
- Retrieve and examine the parameters used to construct an error message.
- Determine the location of the DOM node where an error occurred.

Use the `parseError` property of **DOMDocument** to access this interface from a DOM object.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>allErrors</code></td>
<td>Returns a collection of all errors and warnings found during validation.</td>
</tr>
<tr>
<td></td>
<td>Read-only.</td>
</tr>
<tr>
<td><code>errorParametersCount</code></td>
<td>Returns the number of error parameters returned in <code>errorParameters</code>. Read-only.</td>
</tr>
<tr>
<td><code>errorXPath</code></td>
<td>Returns an XPath expression, which locates the node where the error occurred. Read-only</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errorParameters</code></td>
<td>For a given index, this method returns the corresponding parameter used to construct the error message.</td>
</tr>
</tbody>
</table>

**Events**
None.

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Inheritance:** Inherits from the [IXMLDOMParseError](#) interface.

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[IXMLDOMDocument::parseError Property](#) | [IXMLDOMParseError Interface](#) | [IXMLDOMParseErrorCollection Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMParseErrorCollection

Returns a collection of parsing errors.

Use the allErrors property of the IXMLDOMParseError2 interface to access the IXMLDOMParseErrorCollection interface. Errors in this collection are in the same order in which they were found. The parseError object reported on the DOMDocument is also the first error object in this collection.

To use this interface, you must set the MultipleErrorMessages property on the DOM as shown in the following JScript code fragment:

`objDom.setProperty("MultipleErrorMessages", true);`

**Properties**

- `_newEnum` Returns the enumerator object for the error collection. This allows items in the collection to be enumerated in Visual Basic For Each... In... Next constructions. Read-only.
- `length` Returns the number of items in the error collection. Read-only.
- `item` Returns an item from the error collection with the specified index. The index starts at zero. Read-only
- `next` Returns the next item of the error collection. Read-only.

**Methods**

- `reset` Resets the internal position to the start, so that the next method will return the first error in the collection.

**Events**

None.

**Requirements**
**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Inheritance:** Inherits from the IDispatch interface.

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
**IXMLDOMProcessingInstruction**

Represents a processing instruction, which XML defines to keep processorspecific information in the text of the document.

[Script]

**Example**

The following script example creates a new IXMLDOMProcessingInstruction object and displays its XML representation.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var pi;
pi = xmlDoc.createProcessingInstruction("xml", "version="1.0"");
alert(pi.xml);
```

**Output (in a message box)**

`<?xml version="1.0"?>`

[Visual Basic]

**Example**

The following Microsoft® Visual Basic® example creates a new IXMLDOMProcessingInstruction object and displays its XML representation.

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
Dim pi As IXMLDOMProcessingInstruction
Set pi = xmlDoc.createProcessingInstruction("xml", "version="1.0""
MsgBox pi.xml
```

**Output (in a message box)**

`<?xml version="1.0"?>`

[C/C++]
Example

#import "msxml5.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface) \
    if(pInterface) \
    { \
        pInterface->Release();\ 
        pInterface = NULL;\ 
    }\

#define RELEASE(pInterface) \
    { \
        pInterface->Release();\ 
        pInterface = NULL;\ 
    }\

BOOL BuildDynamicXMLwithProcessingInstruction()
{
    BOOL bResult = FALSE;
    IXMLDOMDocument *pIXMLDOMDocument=NULL;
    IXMLDOMElement *pIXMLDOMElement=NULL;
    IXMLDOMProcessingInstruction *pIXMLDOMProcessingInstruction=NULL;
    IXMLDOMNode *pIXMLDOMNode = NULL;
    HRESULT hr;
    BSTR bstrValue;

    try
    {
        hr=CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_SERVER,
IID_IXMLDOMDocument2, (LPVOID*)(&pIXMLDOMDocument));
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXMLDOMDocument)
        {
            hr=pIXMLDOMDocument->createElement(CComBSTR(L"Node1"),
            &pIXMLDOMElement);
            if(SUCCEEDED(hr) && pIXMLDOMElement)
            {
                hr=pIXMLDOMElement->put_text(_T("test"));
                if(SUCCEEDED(hr))
                {
                    hr=pIXMLDOMDocument->createProcessingInstruction(_T("xml"),
                    _T("version='1.0'"),
                    &pIXMLDOMProcessingInstruction);
                    if(SUCCEEDED(hr) && pIXMLDOMProcessingInstruction)
                    {
                        pIXMLDOMDocument->appendChild(
                        pIXMLDOMProcessingInstruction, &pIXM
                        pIXMLDOMDocument->putref_documentElement(pIXMIXMLDOMElem}
hr=pIXMLDOMDocument->get_xml(&bstrValue);
if(SUCCEEDED(hr))
{
    ::MessageBox(NULL,bstrValue,L"Loaded Doc",MB_OK);
    bResult=TRUE;
    CHECK_AND_RELEASE(pIXMLDOMNode);
    RELEASE(pIXMLDOMProcessingInstruction);
}
RELEASE(pIXMLDOMElemnet);
RELEASE(pIXMLDOMDocument);
}
catch(...)
{
    CHECK_AND_RELEASE(pIXMLDOMElemnet);
    CHECK_AND_Release(pIXMLDOMDocument);
    CHECK_AND_Release(pIXMLDOMNode);
    CHECK_AND_Release(pIXMLDOMProcessingInstruction);
    DisplayErrorToUser();
    return bResult;
}

Output (in a message box)

<?xml version="1.0"?>
<Node1>test</Node1>

Remarks

The content of the ProcessingInstruction node is the entire content between the delimiters of the processing instruction.

The content of this node is usually subdivided into the target (the application to which this processing instruction is directed) and the content of the processing instruction. The target consists of the first token following the start of the tag, while the content of the processing instruction refers to the text that extends from the first non-white space character after the target through the character immediately preceding the ?>, which signifies the end of the tag.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

**Versioning**

MSXML 2.0 and later

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**See Also**

[DOMDocument] [IXMLDOMProcessingInstruction Members]
IXMLDOMProcessingInstruction

Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNode.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td>baseName*</td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td>childNodes</td>
<td>Contains a node list containing the children nodes. Read-only.</td>
</tr>
<tr>
<td>data</td>
<td>Retrieves and sets the content of the processing instruction, excluding the target. Read/write.</td>
</tr>
<tr>
<td>dataType*</td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>definition*</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>lastChild</td>
<td>Returns the last child node. Read-only.</td>
</tr>
<tr>
<td>namespaceURI*</td>
<td>Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Contains the next sibling of the node in the parent's child list. Read-only.</td>
</tr>
<tr>
<td>nodeName</td>
<td>Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.</td>
</tr>
</tbody>
</table>
Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**
Contains the node value expressed in its defined data type. Read/write.

**nodeValue**
Contains the text associated with the node. Read/write.

**ownerDocument**
Returns the root of the document that contains the node. Read-only.

**parentNode**
Contains the parent node. Read-only.

**parsed**
Indicates the parsed status of the node and child nodes. Read-only.

**prefix**
Returns the namespace prefix. Read-only.

**previousSibling**
Contains the previous sibling of the node in the parent's child list. Read-only.

**specified**
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**target**
Specifies the target for the processing instruction. Read-only.

**text**
Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**
Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

- **appendChild**
  Appends a new child as the last child of this node.

- **cloneNode**
  Clones a new node.

- **hasChildNodes**
  Provides a fast way to determine whether a node has children.

- **insertBefore**
  Inserts a child node to the left of the specified node or at the end of the list.
**removeChild**
Removes the specified child node from the list of children and returns it.

**replaceChild**
Replaces the specified old child node with the supplied new child node.

**selectNodes***
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

**selectSingleNode***
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

**transformNode***
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

**transformNodeToObject***
Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[IXMLDOMNode](#) | [IXMLDOMProcessingInstruction](#)
**IXMLDOMSchemaCollection/XMLSchemaCache**

Used by the schemas and namespaces properties of the IXMLDOMDocument2 interface.

[Script]

The following code shows how to create XMLSchemaCache objects in Microsoft® JScript®.

```javascript
var cache = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");
```

The following code shows how to create XMLSchemaCache objects in Microsoft® Visual Basic® Scripting Edition (VBScript).

```vbs
Dim cache
Set cache = CreateObject("Msxml2.XMLSchemaCache.5.0");
```

**Example**

The following example shows how to build a reusable schema cache.

1. Load a document that references schemas through x-schema.

   ```javascript
   var xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
   xmldoc.async = false;
   xmldoc.load("http://myserver/mydata.xml");
   if (xmlDoc.parseError.errorCode <> 0) {
       var myErr = xmlDoc.parseError;
       alert("You have error " + myErr.reason);
   }
   ```

2. Build a cache from the schemas loaded by your document.

   ```javascript
   var cache = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");
   cache.addCollection(xmldoc.namespaces);
   ```

3. Now you can use this cache in the load method of another document.

   ```javascript
   var xmldoc2 = new ActiveXObject("Msxml2.DOMDocument.5.0");
   xmldoc2.async = false;
   ```
xmlDoc2.schemas = cache;
xmlDoc2.load("http://myserver/newdata.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
}

The Xmldoc2 object will load much faster because the schemas are cached.

[C/C++]

The IXMLDOMSchemaCollection implementation in MSXML 2.6 and 3.0 also supports the following interface through QueryInterface.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>IID_EnumVARIANT</td>
<td>Returns an IEnumVARIANT implementation, so that the collection can be used in Microsoft® Visual Basic® For Each statements. Enumerates the namespace Uniform Resource Identifiers (URIs).</td>
</tr>
</tbody>
</table>

**Return Values**

**E_FAIL**
The attempt to modify a read-only object failed for one of the following reasons: argument is not a valid schema; invalid namespace URI; or document is not ready.

**E_INVALIDARG**
The parameter is incorrect.

**E_OUTOFMEMORY**
Out of memory.

**E_POINTER**
Invalid pointer.

**Example**

```cpp
import "msxml5.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface)  
if(pInterface)  
{  
```
pInterface->Release();
pInterface = NULL;
}
#define RELEASE(pInterface) 
{
pInterface->Release();
pInterface = NULL;
}

BOOL DOMDocument2namespaces()
{
    BOOL bResult = FALSE;
    short sResult = FALSE;
    IXMLDOMElement *pIXMLDOMElement=NULL;
    IXMLDOMSchemaCollection *pIXMLDOMSchemaCollection=NULL;
    IXMLDOMNode2 *pIXMLDOMDocument2=NULL;
    HRESULT hr;
    BSTR bstrValue;
    try
    {
        hr=CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_SERVER,
                IID_IXMLDOMNode2, (LPVOID*)(&pIXMLDOMDocument2));
        if(SUCCEEDED(hr))
        {
            hr=pIXMLDOMDocument2->put_async(VARIANT_FALSE);
            if(SUCCEEDED(hr))
            {
                hr=pIXMLDOMDocument2->load(_variant_t(
                    _T("d:\inetpub\wwwroot\samplexmltdt.xml")), &sResult);
                if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
                {
                    hr=pIXMLDOMDocument2->get_namespaces(
                        &pIXMLDOMSchemaCollection);
                    if(SUCCEEDED(hr))
                    {
                        LONG uLength;

                        bResult=TRUE;
                        hr=pIXMLDOMSchemaCollection->get_length(&uLength);
                        if(SUCCEEDED(hr))
                        {
                            for(int iIndex=0; iIndex < uLength; iIndex++)
                            {
                                hr=pIXMLDOMSchemaCollection->get_namespaceURI( iIndex, &bstrValue);
                            }
if(SUCCEEDED(hr))
    ::MessageBox(NULL, bstrValue, _T("Namespace
MB_OK));

RELEASE(pIXMLDOMDocument2);
}
}
}

catch(...)
{
    CHECK_AND_RELEASE(pIXMLDOMDocument2);
    DisplayErrorToUser();
}
return bResult;
}

d:\inetpub\wwwroot\samplexmlldtd.xml

<?xml version='1.0'?>
<COLLECTION xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <DATE dt:dt="datetime">1998-10-13T15:56:00</DATE>
    <BOOK>
        <TITLE>Lover Birds</TITLE>
        <AUTHOR>Cynthia Randall</AUTHOR>
        <PUBLISHER>Lucerne Publishing</PUBLISHER>
    </BOOK>
</COLLECTION>

Output (in a message box)

urn:schemas-microsoft-com:datatypes

Remarks

The IXMLDOMSchemaCollection/XMLSchemaCache object is free-threaded and can be used in multiple documents at the same time. The XML Schema document that is sent to the schema collection is cloned, and the XML Schema document remains writable. Any changes that occur in the XML Schema file after it is stored in the schema cache are not reflected in its cloned image. A single schema cache can be added to multiple schema collections, due to the creation of the cloned image.
Note In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

Versioning

MSXML 2.6 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Version-Dependent ProgID: Msxml2.XMLSchemaCache.5.0

Version-Dependent CLSID: 88d969c2-f192-11d4-a65f-0040963251e5

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMSchemaCollection/XMLSchemaCache_Members | IXMLDOMDocument2
IXMLDOMSchemaCollection/XMLSchemaCache Members

The property and methods in the following tables are an extension to the World Wide Web Consortium (W3C) DOM.

Properties

- **length**: Returns the number of namespaces currently in a collection. Read-only.
- **namespaceURI**: Returns the namespace at the specified index.
- **validateOnLoad**: Indicates whether schemas will be compiled and validated when loaded into the schema cache.

Methods

- **add**: Adds a new schema to the schema collection, and associates the given namespace URI with the specified schema.
- **addCollection**: Adds schemas from another collection into the current collection, and replaces any schemas that collide on the same namespace URI.
- **get**: Returns a read-only Document Object Model (DOM) node that contains the <Schema> element.
- **getDeclaration**: Returns an ISchemaItem object for a specified DOM node. The return value will include declaration information. This information typically references
the schema to be applied when validating the contents of the specified DOM node.

Returns an ISchema object. The schema contains the namespace URI specified in the namespaceURI parameter that is passed to this method. The ISchema interface can be used to further obtain information about the schema object that is returned.

**getSchema**

Removes the specified namespace from a collection.

**remove**

Performs run-time validation on the documents in the schema cache that have not been compiled and validated.

**validate**

**Events**

None.

**See Also**

IXMLDOMSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMSchemaCollection2/XMLDOMSchemaCollection

Used to expand the IXMLDOMSchemaCollection interface. IXMLDOMSchemaCollection2 inherits from IXMLDOMSchemaCollection. The new methods and properties make validation timing more flexible, and enable you to retrieve schema and declaration objects. The ISchema interface can be used to obtain further information from the schema object, which is obtained from the getSchema method call. The ISchemaItem interface can be used to obtain further information from the declaration object, which is obtained from the getDeclaration method.

Remarks

None

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.lib

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMSchemaCollection2/XMLDOMSchemaCollection Members |
**IXMLDOMSchemaCollection2/XMLDOMSchemaCollection Members**

[Visual Basic, Script]

The following table shows the methods of this interface.

[C/C++]

The following table show the methods of this interface.

This interface inherits from XMLDOMSchemaCollection.

**Methods**

- **getSchema**
  Returns an ISchema object that uses the namespace URI sent to this method.

- **getDeclaration**
  Returns a DOM node for an element or attribute of the ISchema object sent to this method.

- **validate**
  Returns a Boolean value indicating whether the validation of the schema document succeeded or failed.

**Properties**

- **validateOnLoad**
  Returns a Boolean value indicating whether the schema is to be validated when it is loaded into the schema cache, or if validation will occur when the validate method is called.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

IXMLDOMSchemaCollection2/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMSelection

Represents the list of nodes that match a given XML Path Language (XPath) expression.

[Script]

Example

In the following Microsoft® JScript® example, you can simply call the IXMLDOMSelection methods on the object returned from selectNodes.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument5.0");
xmlDoc.setProperty("SelectionLanguage", "XPath");
var selection = xmlDoc.selectNodes("//book");
var expr = selection.expr; //This returns "///book"
alert(expr);
```

[Visual Basic]

Example

In the following Microsoft Visual Basic® example, the selectNodes method returns IXMLDOMNodeList, so you have to cast the result of selectNodes into an IXMLDOMSelection.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim Selection As IXMLDOMSelection
Dim sExpr As String
xmlDoc.async = False
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set Selection = xmlDoc.selectNodes("//book")
    sExpr = Selection.expr ; This returns "///book"
    MsgBox (sExpr)
End If
```
Example

In the following C/C++ example, IXMLDOMSelection inherits the threading model of the document that created it. IXMLDOMSelection is created through the selectNodes method on IXMLDOMDocument2.

```cpp
#define CHECK_AND_RELEASE(pInterface)  
    if(pInterface)  
    {  
    pInterface->Release();  
    pInterface = NULL;  
    }  

#define RELEASE(pInterface)  
    {  
    pInterface->Release();  
    pInterface = NULL;  
    }

BOOL DOMSelectionDemo()
{
    BOOL bResult = FALSE;
    short sResult = FALSE;
    IXMLDOMSelection *pIXMLDOMSelection=NULL;
    IXMLDOMNodeList *pIXMLDOMNodeList=NULL;
    IXMLDOMNode *pIXMLDOMNode=NULL;
    IXMLDOMDocument2 *pIXMLDOMDocument2=NULL;
    BSTR bstrValue;
    HRESULT hr;

    try
    {
        hr=CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_SERVER,
                             IID_IXMLDOMDocument2, (LPVOID*)(&pIXMLDOMDocument2));
        if(SUCCEEDED(hr))
        {
            hr=pIXMLDOMDocument2->put_async(VARIANT_FALSE);
            if(SUCCEEDED(hr))
            {
                hr=pIXMLDOMDocument2->load(_variant_t(}
T("d:\inetpub\wwwroot\samplexmltdt.xml"), &sResult);
    if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
    {
        hr=pIXMLDOMDocument2->selectNodes(
            _T("*/BOOK[TITLE='Lover Birds']"), &pIXMLDOMNodeList)
        if(SUCCEEDED(hr))
        {
            hr=pIXMLDOMNodeList->QueryInterface(IID_IXMLDOMSelection
                ,(void**)pIXMLDOMSelection);
            if(SUCCEEDED(hr) && pIXMLDOMSelection)
            {
                LONG uLength;
                bResult=TRUE;
                hr=pIXMLDOMSelection->get_length(&uLength);
                if(SUCCEEDED(hr))
                {
                    for(int iIndex=0; iIndex < uLength; iIndex++)
                    {
                        // remove all the nodes from the list-dis
                        // them as they are removed.
                        hr=pIXMLDOMSelection->removeNext(
                            &pIXMLDOMNode);
                        if(SUCCEEDED(hr) && pIXMLDOMNode)
                        {
                            hr=pIXMLDOMNode->get_text(&bstrValue);
                            if(SUCCEEDED(hr))
                                ::MessageBox(NULL, bstrValue, _T("N
Text"), MB_OK);
                            RELEASE(pIXMLDOMNode);
                        }
                    }
                    RELEASE(pIXMLDOMSelection);
                }
                RELEASE(pIXMLDOMNodeList);
            }
        }
    }
    RELEASE(pIXMLDOMDocument2);
}
}
try
{
    CHECK_AND_RELEASE(pIXMLDOMNode);
    CHECK_AND_RELEASE(pIXMLDOMDocument2);
    CHECK_AND_RELEASE(pIXMLDOMNodeList);
    CHECK_AND_RELEASE(pIXMLDOMSelection);
    DisplayErrorToUser();
function bResult
{
  return bResult;
}

d:\inetpub\wwwroot\samplexmlldtd.xml

<?xml version='1.0'?>
<COLLECTION xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <DATE dt:dt="datetime">1998-10-13T15:56:00</DATE>
  <BOOK>
    <TITLE>Lover Birds</TITLE>
    <AUTHOR>Cynthia Randall</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
  </BOOK>
  <BOOK>
    <TITLE>The Sundered Grail</TITLE>
    <AUTHOR>Eva Corets</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
  </BOOK>
  <BOOK>
    <TITLE>Splish Splash</TITLE>
    <AUTHOR>Paula Thurman</AUTHOR>
    <PUBLISHER>Scootney</PUBLISHER>
  </BOOK>
</COLLECTION>

Output (in a message box)
Lover Birds Cynthia Randall Lucerne Publishing

Remarks

IXMLDOMSelection is an extension of the World Wide Web Consortium (W3C) DOM.

Versioning

MSXML 2.6 and later

Requirements

Implementation: msxml5.dll, msxml2.lib
[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Version-Dependent ProgID:** Msxml2.DOMDocument.5.0, Msxml2.FreeThreadedDOMDocument.5.0

**Version-Dependent CLSID:** 88d969c0-f192-11d4-a65f-0040963251e5

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMSelection Members
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
IXMLDOMSelection Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMNodeList.

The methods and properties in this table are an extension to the World Wide Web Consortium (W3C) DOM.

**Properties**

- **context**
  Gets the node (subtree) that is applied to the selection.

- **expr**
  Gets or sets the XML Path Language (XPath) expression string. Read/write.

- **length**
  Indicates the number of items in a collection. Read-only.

**Methods**

- **clone**
  Clones a copy of the current IXMLDOMSelection, with the same position and context.

- **getProperty**
  Returns a property.

- **item**
  Allows random access to individual nodes within a collection. This method has no effect on the current position.

- **matches**
  Checks if the node that is passed is contained in the current collection.
**nextNode**  
Returns the next node in the collection.

**peekNode**  
Gets the next node that `nextNode` will return without advancing the line position.

**removeAll**  
Removes all the nodes from the collection described by the `IXMLDOMSelection`.

**removeNext**  
Removes the next node.

**reset**  
Resets the iterator.

**Events**

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`IXMLDOMNodeList` | `IXMLDOMSelection`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
**IXMLDOMText**

Represents the text content of an element or attribute.

[Script]

**Example**

The following script example creates an IXMLDOMText object (objText), and inserts it before the first child of the root. The `insertBefore` method returns another IXMLDOMText object, objText1.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var objText;
var objText1;
var objNode;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  root = xmlDoc.documentElement;
  objText = xmlDoc.createTextNode("Hello World");
  objText1 = root.insertBefore(objText, root.firstChild);
  alert(root.xml);
}
```

[Visual Basic]

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMText object (objText), and inserts it before the first child of the root. The `insertBefore` method returns another IXMLDOMText object, objText1.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim objText As IXMLDOMText
Dim objNode As IXMLDOMNode
xmlDoc.Load ("books.xml")
```
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set objText = xmlDoc.createTextNode("Hello World")
    Set objText1 = root.insertBefore(objText, root.firstChild)
    MsgBox root.xml
End If

[C/C++]

Example

The following C/C++ example creates and appends a new text node to the root document element.

#import "msxml5.dll"
using namespace MSXML2;
inline void TESTHR( HRESULT _hr )
{ if FAILED(_hr) throw(_hr); }

void XMLDOMText() {
try {
    IXMLDOMDocumentPtr docPtr;
    IXMLDOMNodePtr DOMNodePtr;
    IXMLDOMNodeMapPtr DOMNamedNodeMapPtr;

    // init
    TESTHR(CoInitialize(NULL));
    TESTHR(docPtr.CreateInstance("Msxml2.DOMDocument.5.0");)

    // load a document
    _variant_t varXml("C:\book.xml");
    _variant_t varOut((bool)TRUE);
    varOut = docPtr->load(varXml);
    if ((bool)varOut == FALSE)
        throw(0);

    DOMNodePtr = docPtr->createTextNode("Hello World");
    docPtr->documentElement->appendChild(DOMNodePtr);
Remarks

XML refers to this text content as character data and distinguishes it from markup, the tags that modify that character data. If there is no markup inside an element, that element's text is contained in a single IXMLDOMText object that is the child of the element. If there is markup inside an element, it is parsed into child elements that are siblings of the IXMLDOMText object(s). (The content of the markup elements also appears within text nodes, which are the children of the specific markup element.)

When a document is first made available to the XML Document Object Model (DOM), all text is normalized: there is only one text node for each block of text. Users can create adjacent text nodes that represent the contents of a given element without any intervening markup but should be aware that there is no way to represent the separations between these nodes, so they will not persist between XML DOM sessions.

The normalize method on the IXMLDOMElement object merges adjacent text nodes into a single node. You should normalize before starting any operations that depend on a particular document structure to ensure that subsequent sessions will operate on the same structure.

Versioning

MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]
**Header and IDL files:** mxml2.h, mxml2.idl

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMElement | IXMLDOMText Members | normalize Method
IXMLDOMText Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLDOMCharacterData.

Properties

- **attributes**
  - Contains the list of attributes for this node. Read-only.
- **baseName**
  - Returns the base name for the name qualified with the namespace. Read-only.
- **childNodes**
  - Contains a node list containing the children nodes. Read-only.
- **data**
  - Stores the node data depending on the node type. Read/write.
- **dataType**
  - Specifies the data type for this node. Read/write.
- **definition**
  - Returns the definition of the node in the document type definition (DTD) or schema. Read-only.
- **firstChild**
  - Contains the first child of the node. Read-only.
- **lastChild**
  - Returns the last child node. Read-only.
- **length**
  - Specifies the length, in characters, of the data. Read-only.
- **namespaceURI**
  - Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.
- **nextSibling**
  - Contains the next sibling of the node in the parent's child list. Read-only.
- **nodeName**
  - Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types. Read-only.
  - Specifies the XML Document Object Model (DOM) node
**nodeType**

Type, which determines valid values and whether the node can have child nodes. Read-only.

**nodeTypedValue**

Contains the node value expressed in its defined data type. Read/write.

**nodeTypeString**

Returns the node type in string form. Read-only.

**nodeValue**

Contains the text associated with the node. Read/write.

**ownerDocument**

Returns the root of the document that contains the node. Read-only.

**parentNode**

Contains the parent node. Read-only.

**parsed**

Indicates the parsed status of the node and child nodes. Read-only.

**prefix**

Returns the namespace prefix. Read-only.

**previousSibling**

Contains the previous sibling of the node in the parent’s child list. Read-only.

**specified**

Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema. Read-only.

**text**

Represents the text content of the node or the concatenated text representing the node and its descendants. Read/write.

**xml**

Contains the XML representation of the node and all its descendants. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

**appendChild**

Appends a new child as the last child of the node.

**appendData**

Appends the supplied string to the existing string data.

**cloneNode**

Clones a new node.

**deleteData**

Deletes specified data.

**hasChildNodes**

Provides a fast way to determine whether a node has children.

**insertBefore**

Inserts a child node to the left of the specified node or at the end of the list.
insertData
Inserts a string at the specified offset.

removeChild
Removes the specified child node from the list of children and returns it.

replaceChild
Replaces the specified old child node with the supplied new child node.

selectNodes
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

specified
Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema.

splitText
Splits this text node into two text nodes at the specified offset and inserts the new text node into the tree as a sibling that immediately follows this node.

substringData
Retrieves a substring of the full string from the specified range.

transformNode
Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNodeToObject
Processes this node and its children using the supplied XSLT style sheet and returns the resulting transformation in the supplied object.

* Denotes an extension to the W3C DOM.

Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
IXMLDOMCharacterData
## XML DOM Properties

This section describes DOM property syntax.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>allErrors</strong></td>
<td>Returns a parseErrorCollection object containing all errors and warnings found during validation. Read-only.</td>
</tr>
<tr>
<td><strong>async</strong></td>
<td>Specifies whether asynchronous download is permitted. Read/write.</td>
</tr>
<tr>
<td><strong>attributes</strong></td>
<td>Contains the list of attributes for this node. Read-only.</td>
</tr>
<tr>
<td><strong>baseName</strong></td>
<td>Returns the base name for the name qualified with the namespace. Read-only.</td>
</tr>
<tr>
<td><strong>childNodes</strong></td>
<td>Contains a node list containing the child nodes. Read-only.</td>
</tr>
<tr>
<td><strong>context</strong></td>
<td>Gets the node (subtree) that is applied to the selection. Read/write.</td>
</tr>
<tr>
<td><strong>data (IXMLDOMCharacterData)</strong></td>
<td>Stores the node data depending on the node type. Read/write.</td>
</tr>
<tr>
<td><strong>data (IXMLDOMProcessingInstruction)</strong></td>
<td>Retrieves and sets the content of the processing instruction, excluding the target. Read/write.</td>
</tr>
<tr>
<td><strong>dataType</strong></td>
<td>Specifies the data type for this node. Read/write.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>definition</td>
<td>Returns the definition of the node in the document type definition (DTD) or schema. Read-only.</td>
</tr>
<tr>
<td>doctype</td>
<td>Contains the document type node that specifies the document type definition (DTD) for this document. Read-only.</td>
</tr>
<tr>
<td>documentElement</td>
<td>Contains the root element of the document. Read/write.</td>
</tr>
<tr>
<td>entities</td>
<td>Contains a list of the entities declared in the &lt;!DOCTYPE&gt; declaration. Read-only.</td>
</tr>
<tr>
<td>errorCode</td>
<td>Contains the error code of the last parse error. Read-only.</td>
</tr>
<tr>
<td>errorParametersCount</td>
<td>Returns the number of error parameters used in an error message. Read-only.</td>
</tr>
<tr>
<td>errorXPath</td>
<td>Returns an XPath expression that locates the node where an error occurred. Read-only.</td>
</tr>
<tr>
<td>expr</td>
<td>Gets or sets the XML Path Language (XPath) expression. Read/write.</td>
</tr>
<tr>
<td>filepos</td>
<td>Contains the absolute file position where the error occurred. Read-only.</td>
</tr>
<tr>
<td>firstChild</td>
<td>Contains the first child of the node. Read-only.</td>
</tr>
<tr>
<td>implementation</td>
<td>Contains the IXMLDOMImplementation object for the document.</td>
</tr>
</tbody>
</table>
input

Specifies which XML input tree to transform. Read/write.

item

Returns the item from the collection with the specified index.

lastChild

Returns the last child node. Read-only.

length (IXMLDOMCharacterData)

Specifies the length, in characters, of the data. Read-only.

length (IXMLDOMNodeList)

Specifies the number of items in the collection. Read-only.

length (IXMLElement)

Returns the number of items in the error collection. Read-only.

length (XMLSchemaCache/IXMLSchemaCollection)

Contains the number of namespaces currently in the collection. Read-only.

line

Specifies the line number that contains the error. Read-only.

linepos

Specifies the character position within the line where the error occurred. Read-only.

name (IXMLDOMAttribute)

Contains the attribute name. Read-only.

name (IXMLDOMDocumentType)

Contains the name of the document type. Read-only.

Read-only.

Returns the list of
namespaces

namespaceURI (IXMLDOMNode)

namespaceURI (XMLSchemaCache/IXMLDOMSchemaCollection)

next

nextSibling

dnodeName

dnodeType

dnodeTypedValue

dnodeTypeString

dnodeValue

namespaces used in the document as XMLSchemaCache. Read-only.

Returns the Uniform Resource Identifier (URI) for the namespace. Read-only.

Returns the namespace at the specified index. Read-only.

Returns the next item in the error collection object.

Contains the next sibling of the node in the parent’s child list. Read-only.

Returns the qualified name for attribute, document type, element, entity, or notation nodes. Returns a fixed string for all other node types. Read-only.

Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes. Read-only.

Contains the node value expressed in its defined data type. Read/write

Returns the node type in string form. Read-only.

Contains the text associated with the node. Read/write.
**notationName**
Contains the notation name. Read-only.

**notations**
Contains a list of the IXMLDOMNotation members that are present in the document type declaration. Read-only.

**ondataavailable**
Specifies the event handler for the ondataavailable event. Write-only.

**onreadystatechange (DOMDocument)**
Specifies the event handler to be called when the readyState property changes. Write-only.

**onreadystatechange (IXMLHTTPRequest)**
Specifies the event handler to be called when the readyState property changes. Write-only.

**onreadystatechange (ServerXMLHTTP/IServerXMLHTTPRequest)**
Specifies the event handler to be invoked when the readyState property changes. Write-only.

**ontransformnode**
Specifies the event handler for the ontransformnode event. Write-only.

**output**
Gets a custom output to write the result of the transformation. Read/write.

**ownerDocument**
Returns the root of the document that contains the node. Read-only.

**ownerTemplate**
Returns the style sheet template that was used to create IXSLProcessor. Read-only.

Contains the parent node.
**parentNode**

Indicates the parsed status of the node and child nodes. Read-only.

**parsed**

Returns a `IXMLDOMParseError` object that contains information about the last parsing error. Read-only.

**parseError**

Returns the namespace prefix. Read-only.

**prefix**

Specifies the default white space handling. Read/write.

**preserveWhiteSpace**

Contains the previous sibling of the node in the parent's child list. Read-only.

**previousSibling**

Contains the public identifier associated with the entity. Read-only.

**publicId (IXMLDOMEntity)**

Contains the public identifier for the notation. Read-only.

**publicId (IXMLDOMNotation)**

Indicates the current state of the XML document. Read-only.

**readyState (DOMDocument)**

Represents the state of the request. Read-only.

**readyState (IXMLHTTPRequest)**

Returns the current state of the processor. Read-only.

**readyState (IXSLProcessor)**

Represents the state of the request. Read-only.

**readyState (ServerXMLHTTP/IServerHTTPRequest)**

Describes the reason for the error. Read-only.

**reason**
Indicates whether external definitions, resolvable namespaces, document type definition (DTD) external subsets, and external entity references, are to be resolved at parse time, independent of validation. Read/write.

Represents only one of several forms in which the HTTP response can be returned. Read-only.

Represents the response entity body as an array of unsigned bytes. Read-only.

Represents only one of several forms in which the HTTP response can be returned. Read-only.

Represents the response entity body as a string. Read-only.

Represents the parsed response entity body. Read-only.

Finds schema documents
specified

srcText

startMode

startModeURI

status (IXMLHTTPRequest)

status (ServerXMLHTTP/IServerXMLHTTPRequest)

statusText (IXMLHTTPRequest)

statusText (ServerXMLHTTP/IServerXMLHTTPRequest)

stylesheet (IXSLProcessor)

stylesheet (IXSLTemplate)
systemId (IXMLDOMNode)
Contains the system identifier for the entity. Read-only.

systemId (IXMLDOMNotation)
Contains the system identifier for the notation. Read-only.

tagName
Contains the element name. Read-only.

target
Specifies the target for the processing instruction. Read-only.

Represented the text content of the node or the concatenated text representing the node and its descendants. Read-only.

text
url (DOMDocument)
Returns the URL for the last loaded XML document. Read-only.

url (IXMLDOMParseError)
Contains the URL of the XML document containing the last error. Read-only.

validateOnLoad
Indicates whether the schema will be compiled and validated when it is loaded into the schema cache. Read/write.

validateOnParse
Indicates whether the parser should validate this document. Read/write.

value
Contains the attribute value. Read/write.

xml
Contains the XML representation of the node and all its descendants. Read-only.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
allErrors Property

This read-only property returns a parseErrorCollection object containing all errors and warnings found during validation.

[Script]

Script Syntax

```javascript
var oAllErrs = objXMLDOMParseError2.allErrors;
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set oAllErrs = objXMLDOMParseError2.allErrors;
```

[C/C++]

C/C++ Syntax Using Smart Pointers

```c
IXMLDOMParseErrorCollectionPtr oAllErrs = objXMLDOMParseError2->allErrors;
```

C/C++ Syntax

```c
HRESULT get_allErrors(
    IXMLDOMParseErrorCollection **oAllErrs)
```

Parameters

`oAllErrs [out,retval]`

A pointer to the IXMLDOMParseErrorCollection object.

Return Values

S_OK
The error collection was retrieved successfully. The resultant error collection is never empty. If no errors were found during validation, the returned error collection contains the success code.

**E_FAIL**
The `oAllErrs` parameter was called on error item belonging to a `parseErrorCollection` object.

**Remarks**

To retrieve multiple validation errors, you must enable `MultipleErrorMessage` on the DOM. To do so, you must explicitly set the `MultipleErrorMessage` parameter to `true`, as shown in the following JScript code fragment:

```javascript
ObjDom.setProperty("MultipleErrorMessage", true);
```

The `allErrors` parameter is only meaningful when it is called on the top-level `parseError` object returned from a DOM object, as follows:

```javascript
errCollection = domObj.parseError.allErrors;
```

It is an error to call the `allErrors` parameter on a `parseError` object returned at lower levels in the DOM such as in the following example:

```javascript
ErrCollection = domObj.parseError.allErrors.item(0).allErrors;
```

In this case, the `allErrors` parameter returns `NULL`. The corresponding HRESULT value is `E_FAIL`.

**Example**

The example uses two resource files, `books.xml` and `books.xsd`. The first is an XML data file, and the second is the XML Schema for the XML data. The XML document has two invalid `<book>` elements: `<book id="bk002">` and `<book id="bk003">`. The sample application uses several methods and properties of the `IXMLDOMParseError2` interface to examine the resulting parse errors.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (`books.xml` and `books.xsd`)
Applies To

IXMLDOMParseError2

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

errorParameters Method | errorParametersCount Property | errorXPath Property | IXMLDOMParseErrorCollection Interface
Resource Files (books.xml and books.xsd)

These resource files are used in the samples for the following interface members:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXMLDOMParseError2</td>
<td>allErrors Property; errorXPath Property; errorParameters Method</td>
</tr>
<tr>
<td>IXMLDOMParseErrorCollection</td>
<td>_newEnum Property; length Property; item Property; next Property; reset Method</td>
</tr>
</tbody>
</table>

Input XML Data File (books.xml)

```xml
<?xml version="1.0"?>
<x:books xmlns:x="urn:books"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:books books.xsd">
    <book id="bk001">
        <author>Hightower, Kim</author>
        <title>The First Book</title>
        <genre>Fiction</genre>
        <price>44.95</price>
        <pub_date>2000-10-01</pub_date>
        <review>An amazing story of nothing.</review>
    </book>

    <book id="bk003">
        <author>Nagata, Suanne</author>
        <title>Becoming Somebody</title>
        <genre>Biography</genre>
        <review>A masterpiece of the fine art of gossiping.</review>
    </book>

    <book id="bk002">
        <author>Oberg, Bruce</author>
        <title>The Poet's First Poem</title>
        <genre>Poem</genre>
        <price>24.95</price>
        <review>The least poetic poems of the decade.</review>
    </book>
</x:books>
```
XML Schema Definition File (books.xsd)

```xml
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:books"
    xmlns:bks="urn:books">

    <xs:element name="books" type="bks:BooksForm"/>

    <xs:complexType name="BooksForm">
        <xs:sequence>
            <xs:element name="book"
                type="bks:BookForm"
                minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="BookForm">
        <xs:sequence>
            <xs:element name="author" type="xsd:string"/>
            <xs:element name="title" type="xsd:string"/>
            <xs:element name="genre" type="xsd:string"/>
            <xs:element name="price" type="xsd:float"/>
            <xs:element name="pub_date" type="xsd:date"/>
            <xs:element name="review" type="xsd:string"/>
        </xs:sequence>
        <xs:attribute name="id" type="xsd:string"/>
    </xs:complexType>

</xs:schema>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
JScript Code (allErrors.js)

```javascript
var xs, xd;
parseError2();

function parseError2()
{
    try {
        xs = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
        xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
    }
    catch (e) {
        alert("Mirosft XML Core Services (MSXML) 5.0 for Microsoft Offi
            Download and install MSXML 5.0 from http://msdn.microsoft
            before continuing.");
        return;
    }

    try {
        xs.add("urn:books", "books.xsd");
    }
    catch (e) {
        alert("Failed to add to schema cache: "+e.description);
        return;
    }

    try {
        xd.schemas = xs;
        xd.async = false;
        xd.validateOnParse = false;
        xd.load("books.xml");
        xd.setProperty("MultipleErrorMessages", true);
    }
    catch (e) {
        alert("can't load books.xml : " + e.description);
        return;
    }

    var err = xd.validate();
    if (err.errorCode != 0 ) {  
        errsCount = err.allErrors.length;
        var errStr = "Error as returned from validate():\n            \n            \tError Code : " + err.errorCode
            \n            \tError reason :\n            \n            \tError location:\n        +"\n        return;
    }
```
for (var i=0; i<errsCount; i++)
{
    var e = err.allErrors.item(i);
    errStr += "\n\nErrorItem["+i+" ] : ";
    errStr += "\n\treason:\n" + e.reason;
    errStr += "\n\tlocation:\n" + e.errorXPath;
}
alert(errStr);
}
else
    alert("valid dom as follows:\n" + xd.xml);
}

function alert(msg)
{
    WScript.echo(msg);
}

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as allErrors.js in the same directory where you saved books.xml and books.xsd.
4. Double click the allErrors.js file from Windows Explorer to launch the application. Alternatively, you can type "allErrors.js" from a command prompt.

   **Note** Under operating systems other than Windows 2000 or Windows XP, you need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in Output for the allErrors Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer...
Visual Basic Code (allErrors.frm)

' Check "Microsoft XML, v50" from Project->References...
' in Visual Studio.
Dim er, e As IXMLDOMParseError2
Dim errs As IXMLDOMParseErrorCollection
Dim xd As New DOMDocument50
Dim xs As New XMLSchemaCache50

' Add xs to xd.
xs.Add "urn:books", App.Path & "\books.xsd"
Set xd.schemas = xs
xd.async = False
xd.validateOnParse = False
xd.setProperty "MultipleErrorMessages", True

' Load the XML file to xd.
xd.Load App.Path & "\books.xml"
Set er = xd.Validate
If er.errorCode <> 0 Then
    Set errs = er.allErrors
    errsCount = errs.length
    msg = "Error as returned from validate():" & vbCrLf _
    & vbCrLf & " Error Code:" & vbCrLf & er.errorCode _
    & vbCrLf & " Error reason:" & vbCrLf & er.reason _
    & vbCrLf & " Error location:" & vbCrLf & er.errorXPath _
    & vbCrLf & " Error Count:" & vbCrLf & errsCount & vbCrLf _
    & vbCrLf & "Error items from the allErrors collection:" _
    & vbCrLf & vbCrLf
    For i = 0 To errsCount - 1
        Set e = errs.Item(i)
        msg = msg & vbCrLf & vbCrLf & "Error Item[" & i & "]" _
            & vbCrLf & " reason:" & vbCrLf & e.reason _
            & vbCrLf & " localtion:" & vbCrLf & e.errorXPath
    Next
Else
    msg = msg & "DOM is valid:" & vbCrLf & vbCrLf
End If

MsgBox msg

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.

3. Create a Standard EXE project in Visual Basic. Save the empty project as allErrors.vbp to the same directory where you saved books.xml and books.xsd. Name the form file allErrors.frm.

4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.

5. Copy the Visual Basic code listing above, and paste it into the form_load subroutine.

6. Execute the code by selecting Start from the Run menu.

7. Verify that your output is the same as that listed in Output for the allErrors Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
C/C++ Code (allErrors.cpp)

```c
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[])
{
    HRESULT hr;
    CoInitialize(NULL);

    IXMLDOMDocument3Ptr pXMLDoc;
    IXMLDOMDocument3Ptr pXSDDoc;
    IXMLDOMSchemaCollectionPtr pSCache;
    IXMLDOMParseError2Ptr pEitem;
    IXMLDOMParseError2Ptr pError;

    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr))
    {
        printf("Failed to CreateInstance on an XML DOM");
        return -1;
    }
    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
    if (FAILED(hr))
    {
        printf("Failed to enable mulitple validation errors\n");
        return -1;
    }
    if(pXMLDoc->load("books.xml")!=VARIANT_TRUE)
    {
        printf("Failed to load DOM from books.xml. %s\n",
                (LPCSTR)pXMLDoc->parseError->reason);
        return -1;
    }

    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr))
    {
        printf("Failed to CreateInstance on an XSD Doc");
        return -1;
    }
    pXSDDoc->async = VARIANT_FALSE;
```
pXSDDoc->validateOnParse = VARIANT_FALSE;
if(pXSDDoc->load("books.xsd")!=VARIANT_TRUE)
{
    printf("Failed to load XSD from books.xsd. %s\n", (LPCSTR)pXSDDoc->parseError->reason);
    return -1;
}

hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr))
{
    printf("Cannot instantiate XMLSchemaCache50\n");
    return -1;
}
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr))
{
    printf("Cannot add 'urn:books' to schema cache.\n");
    return -1;
}

// Validate the entire DOM object.
pError = pXMLDoc->validate();
if (pError->errorCode != 0)
{
    printf("Error as returned from validate():\n\n");
    printf("\tError Code: \n\n", pError->errorCode);
    printf("\tError reason: \n\n", (LPCSTR)pError->reason);
    printf("\tError location: \n\n", (LPCSTR)pError->errorXPath);
    printf("\tErrors count: \n\n", pError->allErrors->length);
    printf("\n\nError items from the allErrors collection:\n\n");
    for (int i=0; i<pError->allErrors->length; i++)
    {
        pEitem = pError->allErrors->item[i];
        printf("\n\nErrorItem[%d]:\n\n", i);
        printf("\treason: \n\n", (LPCSTR)pEitem->reason);
        printf("\tlocation: \n\n", (LPCSTR)pEitem->errorXPath);
    }
}
else
    printf("valid dom as follows:\n\n", (LPCSTR)pXMLDoc->xml);

pError.Release();
pEitem.Release();
pXMLDoc.Release();
pXSDDoc.Release();
pSCache.Release();
CoUninitialize();
return 0;
}

Try It!

2. From the **File** menu, select **New**. On the **Projects** tab of the **New** dialog box that appears, select **Win32 Console Application** in the left pane. Then type "allErrorsProj" in the **Project name** field. For the project **Location** field, either accept the default setting or choose another location. Click **OK**.
3. The **Win32 Console Application** property page will appear. For the type of Console Application, select **An empty project** and click **Finish**. When the **New Project Information** box displays, click **OK**.
4. Select **FileView** on the project browser, and highlight **allErrorsProj files**. From the **File** menu, select **New**.
5. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "books.xml" in the **File name** text box. Be sure to put quotes around the name of the text file. Otherwise, Visual C++ will generate an unwanted .txt suffix to the XML file.
6. Click **OK**.
7. Copy the XML data file (**books.xml**), and paste it into the text file you just created.
8. Repeat steps 4-7 for the XSD listing (**books.xsd**).

   **Note** You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ source code above (**allErrors.cpp**).
10. Build the sample by selecting **Build allErrorsProj.exe** from the **Build** menu.
11. Execute the sample application by selecting **!Execute allErrorsProj.exe** from the **Build** menu.
12. Verify that the result is the same as that listed in **Output for the allErrors Example**.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
Output for the allErrors Example

When you build and run the allErrors example, you should get the following output, displayed in a message box.

Error as returned from validate():

    Error Code : -1072898028
    Error reason :
    Element 'review' is unexpected according to content model of parent
    Expecting: price.
    Error location:
    /*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/rev
    Errors Count : 2

Error items from the allErrors collection:

ErrorItem[0] :
    reason:
    Element 'review' is unexpected according to content model of parent
    Expecting: price.
    location:
    /*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/rev

ErrorItem[1] :
    reason:
    Element 'review' is unexpected according to content model of parent
    Expecting: pub_date.
    location:
    /*[local-name()="books" and namespace-uri()="urn:books"]/book[3]/rev
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
async Property

Specifies whether asynchronous download is permitted.

[Script]

Script Syntax

`boolValue = oXMLDOMDocument.async;
objXMLDOMDocument.async = boolValue;`

Example

The following example sets the async property of a DOMDocument object to false before loading books.xml.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(xmlDoc.xml);
}
```

[Visual Basic]

Visual Basic Syntax

`boolValue = oXMLDOMDocument.async
objXMLDOMDocument.async = boolValue`

Example

The following example sets the async property of a DOMDocument object to false before loading books.xml.

`Dim xmlDoc As New Msxml2.DOMDocument50`
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox xmlDoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_async(
    VARIANT_BOOL *isAsync);
HRESULT put_async(
    VARIANT_BOOL isAsync);

Parameters

isAsync [out][in]
    True if asynchronous download is permitted; False if not.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG (for get_async only)
    The value returned if isAsync is Null.

Remarks

Boolean. The property is read/write. Returns True if asynchronous download is permitted; False if not. Default is True.

When set to True, the Load method returns control to the caller before the download is finished. You can then use the readyState property to check the status of the download. You can also attach an onreadystatechange handler or connect to the onreadystatechange event to be notified when the ready state changes so that you know when the download is complete.
This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

onreadystatechange Property | onreadystatechange Event | readyState Property

Applies to: DOMDocument
attributes Property

Contains the list of attributes for this node.

[Script]

Script Syntax

```javascript
var objXMLDOMNamedNodeMap = oXMLDOMNode.attributes;
```

Example

The following creates an IXMLDOMNodeNamedNodeMap object from a document's attributes property, and then displays the number of nodes in the object.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oNamedNodeMap;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    oNamedNodeMap = xmlDoc.documentElement.firstChild.attributes;
    alert(oNamedNodeMap.length);
}
```

[Visual Basic]

Visual Basic Syntax

```visualbasic
Set objXMLDOMNamedNodeMap = oXMLDOMNode.attributes
```

Example

The following creates an IXMLDOMNodeNamedNodeMap object from a document's attributes property, and then displays the number of nodes in the object.

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
```
Dim oNamedNodeMap As IXMLDOMNamedNodeMap
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set oNamedNodeMap = xmlDoc.documentElement.firstChild.Attributes
    MsgBox oNamedNodeMap.length
End If

[C/C++]

C/C++ Syntax

HRESULT get_attributes(
    IXMLDOMNamedNodeMap **attributeMap);

Parameters

attributeMap [out, retval]
    An IXMLDOMNamedNodeMap returned for nodes that can return attributes
    (Element, Entity, and Notation nodes). Returns Null for all other node
    types. For the valid node types, the NamedNodeMap is always returned; when
    there are no attributes on the element, the list length is set to zero.

C/C++ Return Values

S_OK
    Success. Returns a nonnull pointer value.
S_FALSE
    The value returned if the node type does not allow attributes.
E_INVALIDARG
    The value returned if attributeMap is Null.

Remarks

The property is read-only. Returns IXMLDOMNamedNodeMap for nodes that can
return attributes (Element, Entity, and Notation nodes). Returns Null for all
other node types. For the valid node types, theNamedNodeMap is always returned;
when there are no attributes on the element, the list length is set to 0.

This value depends on the value of the `nodeType` property.

<table>
<thead>
<tr>
<th><code>nodeType</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ATTRIBUTE</td>
<td>Always returns <code>null</code></td>
</tr>
<tr>
<td>NODE_CDATA_SECTION</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
<tr>
<td>NODE_COMMENT</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
<tr>
<td>NODE_DOCUMENT</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
<tr>
<td>NODE_DOCUMENT_FRAGMENT</td>
<td>For the XML declaration, the version, encoding, and standalone specifications can be accessed as attributes from the corresponding node. Values specified in the <code>&lt;!DOCTYPE ...&gt;</code> can be accessed as attributes from the node corresponding to the document type declaration. The names are &quot;SYSTEM&quot; and &quot;PUBLIC&quot;. They are uppercase.</td>
</tr>
<tr>
<td>NODE_ENTITY_REFERENCE</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
<tr>
<td>NODE_TEXT</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
<tr>
<td>NODE_ELEMENT</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
<tr>
<td>NODE_PROCESSING_INSTRUCTION</td>
<td>For the XML declaration, the version, encoding, and standalone specifications can be accessed as attributes from the corresponding node. Values specified in the <code>&lt;!DOCTYPE ...&gt;</code> can be accessed as attributes from the node corresponding to the document type declaration. The names are &quot;SYSTEM&quot; and &quot;PUBLIC&quot;. They are uppercase.</td>
</tr>
<tr>
<td>NODE_DOCUMENT_TYPE</td>
<td>Returns <code>null</code> for all processing instructions except the XML declaration, for example, <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;windows-1252&quot; standalone=&quot;yes&quot; ?&gt;</code></td>
</tr>
</tbody>
</table>
NODE_ENTITY

Returns IXMLDOMNodeMap that exposes the PUBLIC ID, SYSTEM ID, and NDATA as attributes.

NODE_NOTATION

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [x] in the upper-left corner of the page.

See Also

IXMLDOMNodeMap | nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**baseName Property**

Returns the base name for the name qualified with the namespace.

[Script]

**Script Syntax**

```
strValue = oXMLDOMNode.baseName;
```

**Example**

The following script example assigns the value of a node's `baseName` property to a string, and then displays it.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var MyStr;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  MyStr = xmlDoc.documentElement.childNodes.item(1).baseName;
  alert(MyStr);
}
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
strValue = oXMLDOMNode.baseName
```

**Example**

The following Microsoft® Visual Basic® example assigns the value of a node's `baseName` property to a string, and then displays it.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
```
Dim MyStr As String
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MyStr = xmlDoc.documentElement.childNodes.Item(1).baseName
    MsgBox MyStr
End If

[C/C++]

**C/C++ Syntax**

```
HRESULT get_baseName(
    BSTR *nameString);
```

**Parameters**

- `nameString` [out, retval]
  The right-hand side of a namespace qualified name. For example, it returns `yyy` for the element `<xxx:yyy>`. It always returns a nonempty string.

**C/C++ Return Values**

- **S_OK**
  The value returned if successful.
- **S_FALSE**
  The value returned when there is no base name.
- **E_INVALIDARG**
  The value returned if the `nameString` parameter is Null.

**Remarks**

The property is read-only. It returns the right-hand side of a namespace-qualified name. For example, it returns "yyy" for the element `<xxx:yyy>`. It always returns a nonempty string.

This member is an extension of the World Wide Web Consortium (W3C)
Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMDocument | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
childNodes Property

Contains a node list containing the child nodes.

[Script]

**Script Syntax**

```
objXMLDOMNodeList = oXMLDOMNode.childNodes;
```

**Example**

The following script example uses the childNodes property (collection) to return an IXMLDOMNodeList, and then iterates through the collection, displaying the value of each item's xml property.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var oNodeList;
var Item;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
oNodeList = root.childNodes;
    for (var i=0; i<oNodeList.length; i++) {
        Item = oNodeList.item(i);
        alert(Item.xml);
    }
}
```

[Visual Basic]

**Visual Basic Syntax**

```
objXMLDOMNodeList = oXMLDOMNode.childNodes
```
Example

The following Microsoft® Visual Basic® example uses the childNodes property (collection) to return an IXMLDOMNodeList, and then iterates through the collection, displaying the value of each item's xml property.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim oNodeList As IXMLDOMNodeList
Dim Item As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set oNodeList = root.childNodes
    For Each Item In oNodeList
        MsgBox Item.xml
    Next
End If

[C/C++]

C/C++ Syntax

HRESULT get_childNodes(
    IXMLDOMNodeList **childList);

Parameters

childList [out, retval]
    A list of children in the current node.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the childList parameter is Null.
Remarks

The property (collection) is read-only. IXMLDOMNodeList is returned even if there are no children of the node. In such a case, the length of the list will be set to 0. For information about valid child node types for each node, see XML DOM Enumerated Constants.

This value depends on the value of the nodeType property.

- **NODE_ATTRIBUTE**
- **NODE_DOCUMENT**
- **NODE_DOCUMENT_FRAGMENT** Returns an IXMLDOMNodeList that contains a list of all child nodes for the specified node.
- **NODE_ELEMENT**
- **NODE_ENTITY**
- **NODE_ENTITY_REFERENCE**
- **NODE_CDATA_SECTION**
- **NODE_COMMENT** Returns an IXMLDOMNodeList with a length of 0. These node types cannot have children.
- **NODE_NOTATION**
- **NODE_PROCESSING_INSTRUCTION**
- **NODE_TEXT** Returns an IXMLDOMNodeList that contains a list of all child nodes for the IXMLDOMDocumentType node. The node list for the document type node can contain entities and notations.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

IXMLDOMNodeList | nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElemnet | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
context Property

Gets the node (subtree) that is applied to the selection.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = objXMLDOMSelection.context;
objXMLDOMSelection.context = objXMLDOMNode;
```

Example

The following script example shows what is contained in the context of a selection after a query is executed. It also shows that the selection is reset when this property is changed.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var selection;
xmlDoc.loadXML("<Customer>Microsoft</Customer>");
xmlDoc.setProperty("SelectionLanguage", "XPath");
selection = xmlDoc.selectNodes("//Customer");
alert(selection.context.xml);

The selection.context.xml contains "<Customer>Microsoft</Customer>";
selection.length = 1.

xmlDoc.loadXML("<new>none</new>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(selection.context.xml);
}

The selection.context.xml contains "<new>none</new>"; selection.length = 0.
```

[Visual Basic]
**Visual Basic Syntax**

Set objXMLDOMNode = objXMLDOMSelection.context
objXMLDOMSelection.context = objXMLDOMNode

**Example**

The following Microsoft® Visual Basic® example shows what is contained in the context of a selection after a query is executed. It also shows that the selected set is reset when this property is changed.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim selection As Msxml2.IXMLDOMSelection

xmlDoc.loadXML ("<Customer>Microsoft</Customer>")
xmlDoc.setProperty "SelectionLanguage", "XPath"
Set selection = xmlDoc.selectNodes("//Customer")
MsgBox selection.context.xml
```

The `selection.context.xml` contains "<Customer>Microsoft</Customer>"; `selection.length = 1`.

```vbscript
xmlDoc.loadXML ("<new>none</new>")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox selection.context.xml
End If
```

The `selection.context.xml` contains "<new>none</new>"; `selection.length = 0`.

**[C/C++]**

**C/C++ Syntax**

```c
HRESULT get_context(IXMLDOMNode ** ppNode);
HRESULT putref_context(IXMLDOMNode * pNode);
```

**Parameters**
**ppNode** [out, retval]

The subtree to apply the pattern to.

**pNode** [in]

The subtree to apply the pattern to.

**C/C++ Return Values**

S_OK

The value returned if the method is successful.

S_FALSE (for get_context only)

The value returned if the context is invalid.

E_INVALIDARG (for get_context only)

The value returned if the input argument is Null.

E_FAIL (for putref_context only)

The value returned if a node with a different threading model is provided.

**Example**

The following C/C++ example shows what is contained in the context property after selectNodes is executed.

```c
#import "msxml5.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface) \ 
if(pInterface) \ 
{\ 
pInterface->Release();\ 
pInterface = NULL;\ 
}\n
#define RELEASE(pInterface) \ 
{\ 
pInterface->Release();\ 
pInterface = NULL;\ 
}\n
BOOL DOMSelectionContextDemo()
{
    BOOL bResult = FALSE;
    short sResult = FALSE;
    IXMLDOMSelection *pIXMLDOMSelection=NULL;
    IXMLDOMNode *pIXMLDOMNode=NULL;
```
IXMLDOMDocument2 *pIXMLDOMDocument2 = NULL;
HRESULT hr;
BSTR bstrValue;

try
{
    hr = CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_SERVER,
                           IID_IXMLDOMDocument2, (LPVOID*)(&pIXMLDOMDocument2));
    SUCCEEDED(hr) ? 0 : throw hr;

    if(pIXMLDOMDocument2)
    {
        hr = pIXMLDOMDocument2->put_async(VARIANT_FALSE);
        if(SUCCEEDED(hr))
        {
            hr = pIXMLDOMDocument2->load(_variant_t(_T("d:\inetpub\wwwroot\samplexml.xml")), &sResult);
            if(SUCCEEDED(hr) && (sResult == VARIANT_TRUE))
            {
                hr = pIXMLDOMDocument2->selectNodes(_T("*/BOOK[TITLE='Lover Birds']"),
                                                    (IXMLDOMNodeList**) &pIXMLDOMSelection);
                if(SUCCEEDED(hr))
                {
                    if(SUCCEEDED(hr) && pIXMLDOMSelection)
                    {
                        hr = pIXMLDOMSelection->get_context(&pIXMLDOMNode);
                        if(SUCCEEDED(hr) && pIXMLDOMNode)
                        {
                            hr = pIXMLDOMNode->get_xml(&bstrValue);
                            if(SUCCEEDED(hr))
                                ::MessageBox(NULL, bstrValue, _T("Current Selection Context"), MB_OK);
                            bResult = TRUE;
                            RELEASE(pIXMLDOMNode);
                        }
                        RELEASE(pIXMLDOMSelection);
                    }
                    RELEASE(pIXMLDOMDocument2);
                }
                catch(...) { }
            }
        }
    }
    RELEASE(pIXMLDOMNode);
    RELEASE(pIXMLDOMSelection);
    CHECK_AND_RELEASE(pIXMLDOMDocument2);
}
DisplayErrorToUser();
}
return bResult;
}

Data File: d:\inetpub\wwwroot\samplexml.xml


Output (in a message box)


Remarks

Because the selectNodes method can be called on any node type, context can also be assigned any node type. A node from a document different from the one that originally created IXMLDOMSelection can also be specified as long as it has the same threading model. Calling context also resets the state of the node list so that nextNode starts over.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSelection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
data Property
(IXMLDOMCharacterData)

Stores the node data depending on the node type.

[Script]

Script Syntax

strValue = oXMLDOMCharacterData.data;
objXMLDOMCharacterData.data = strValue;

Example

The following script example walks the document tree and checks for comment node types. If one is found, it displays its contents with the data property.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
var root;
xmlDoc.async = false;
xmlDoc.loadXML("<root><!-- Hello --></root>" Alla); if (xmlDoc.parseError.errorCode <> 0) { var myErr = xmlDoc.parseError;
alert("You have error " + myErr.reason); }
else {
    root = xmlDoc.documentElement;
    for (var i=0; i<root.childNodes.length; i++) { if (root.childNodes.item(i).nodeType == 8) {
        comment = root.childNodes.item(i);
        alert(comment.data);
    }
}
}
```

[Visual Basic]

Visual Basic Syntax
strValue = oXMLDOMCharacterData.data
objXMLDOMCharacterData.data = strValue

**Example**

The following Microsoft® Visual Basic® example walks the document tree and checks for comment node types. If one is found, it displays its contents with the data property.

```vbnet
Dim xmlDoc As New DOMDocument50
Dim comment As IXMLDOMComment
Dim root As IXMLDOMElement
xmlDoc.async = False
xmlDoc.loadXML("<root><!-- Hello --></root>"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    For i = 0 To (root.childNodes.length - 1)
        If root.childNodes.Item(i).nodeType = NODE_COMMENT Then
            Set comment = root.childNodes.Item(i)
            MsgBox comment.Data
        End If
    Next
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_data(
    BSTR *strData);
HRESULT put_data(
    BSTR strData);
```

**Parameters**

*strData* [out, retval][in]

The same value as the *nodeValue* for this node.

**C/C++ Return Values**
S_OK
   The value returned if successful.
S_FALSE (for get_data only)
   The value returned if there is no character data node.
E_INVALIDARG (for get_data only)
   The value returned if the data parameter is Null.
E_FAIL (for put_data only)
   The value returned if an error occurs.

Remarks

String. The property is read/write. It contains the same value as the nodeValue for this node. The meaning of the value depends on the nodeType property of the IXMLDOMNode, as follows.

- **NODE_CDATA_SECTION** A string representing the text stored in the CDATA section.
- **NODE_COMMENT** The content of the comment, exclusive of the comment start and end sequence.
- **NODE_TEXT** A string representing the text stored in the text node.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ⓐ in the upper-left corner of the page.

See Also

IXMLDOMNode | nodeType Property | nodeValue Property

Applies to: IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
data Property
(IXMLDOMProcessingInstruction)

Retrieves and sets the content of the processing instruction, excluding the target.

[Script]

Script Syntax

strValue = oXMLDOMProcessingInstruction.data;
objXMLDOMProcessingInstruction.data = strValue;

Example

The following script example iterates through the document's child nodes. If it finds a node of type NODE_PROCESSING_INSTRUCTION (7), it displays the node's data.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var pi;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    for (var i=0; i<xmlDoc.childNodes.length; i++) {
        if (xmlDoc.childNodes.item(i).nodeType == 7) {
            pi = xmlDoc.childNodes.item(i);
            alert(pi.data);
        }
    }
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
strValue = oXMLDOMProcessingInstruction.data
objXMLDOMProcessingInstruction.data = strValue
```
Example

The following Microsoft® Visual Basic® example iterates through the document's child nodes. If it finds a node of type NODE_PROCESSING_INSTRUCTION (7), it displays the node's data.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim pi As IXMLDOMProcessingInstruction
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    For i = 0 To (xmlDoc.childNodes.length - 1)
        If xmlDoc.childNodes.Item(i).nodeType = _
            NODE_PROCESSING_INSTRUCTION Then
            Set pi = xmlDoc.childNodes.Item(i)
            MsgBox pi.Data
        End If
    Next
End If
```

[C/C++]

C/C++ Syntax

```c
HRESULT get_data(
    BSTR *value);
HRESULT put_data(
    BSTR value);
```

Parameters

`value [out, retval][in]`

The content of the processing instruction for this target. The data has the same value as the `nodeValue` property.

C/C++ Return Values

`S_OK`

The value returned if successful.
Remarks

String. This property is read/write. The data has the same value as the `nodeValue` property.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

`nodeValue Property`

Applies to: `IXMLDOMProcessingInstruction`
**dataType Property**

Specifies the data type for this element or attribute. When a DTD is used, the `dataType` property is only applicable to attributes, in which case `dataType` returns the attribute-type according to the DTD. The value returned will use XDR naming conventions, however. In other words, the type is returned in lower-case. This is different from the DTD convention. When using XDR, or when an element has a `dt:dt` attribute, elements may also have datatypes.

**[Script]**

**Script Syntax**

```
objValue = oXMLDOMNode.dataType;
objXMLDOMNode.dataType = objValue;
```

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.loadXML("<root/>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    var root = xmlDoc.documentElement;
    root.dataType = "int";
    root.nodeTypedValue = 5;
    alert(xmlDoc.xml);
}
```

**[Visual Basic]**

**Visual Basic Syntax**

```
objValue = oXMLDOMNode.dataType
oxmldomnode.dataType = objValue
```

**Example**
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
xmlDoc.async = False
xmlDoc.loadXML "<root/>
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    root.dataType = "int"
    root.nodeTypedValue = 5
    MsgBox xmlDoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_dataType(
    VARIANT *dataTypeName);
HRESULT put_dataType(
    BSTR dataTypeName);

Parameters

dataTypeName [out, retval][in]
    The VARIANT can be a BSTR that contains the data type name, or VT_NULL if no data type is defined.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE (for get_dataType only)
    The value returned if the node is not typed.
E_INVALIDARG (for get_dataType only)
    The value returned if the dataTypeName parameter is Null.
E_FAIL (for put_dataType only)
    The value returned if an error occurs.

Remarks
The property is read/write. The VARIANT can be a BSTR that contains the data type name, or VT_NULL if no data type is defined. This property applies to DTDs only, not to XML-Data Reduced (XDR) schemas or XML Schemas (XSD). This value depends on the value of the nodeType property of the IXMLDOMNode.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

- **NODE_ATTRIBUTE**: Contains the string representation of the data type specifier included in the schema, or Null if no data type is specified.
- **NODE_CDATA_SECTION**
- **NODE_COMMENT**
- **NODE_DOCUMENT**
- **NODE_DOCUMENT_FRAGMENT**: Contains the constant string value "string". These node types do not have data types, so the dataType property cannot be set for these node types. Attempts to set this property are ignored.
- **NODE_DOCUMENT_TYPE**
- **NODE_ENTITY**
- **NODE_NOTATION**
- **NODE_PROCESSING_INSTRUCTION**
- **NODE_TEXT**: Contains the string representation of the data type specifier included on the instance of the element or in the schema, or Null if no data type is specified.
- **NODE_ELEMENT**: Contains the string representation of the data type specifier for the referenced entity. This is limited to entities with a single data type on the
NODE_ENTITY_REFERENCE

Root level element of the subtree corresponding to the entity. Contains Null if no data type is specified. It is not possible to set the data type of an entity reference. The data type must be set directly on the referenced element. Attempts to set the value of the dataType property are ignored.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeType Property

Applies to: IXMLElement | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
**definition Property**

Returns the definition of the node in the document type definition (DTD) or schema.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNode.definition;
```

[Visual Basic]

**Visual Basic Syntax**

```vbasic
Set objXMLDOMNode = oXMLDOMNode.definition
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_definition(
    IXMLDOMNode **definitionNode);
```

**Parameters**

`definitionNode [out, retval]`

   The node that contains the definition for the entity referenced.

**C/C++ Return Values**

S_OK
   The value returned if successful.
S_FALSE
   The value returned when no definition is found.
E_INVALIDARG
The value returned if the `definitionNode` parameter is Null.

Example

The following C/C++ example shows the retrieval of the `definition` property from an `IXMLDOMNode`.

```cpp
#import "msxml3.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface) \  
if(pInterface) \ 
{\ 
pInterface->Release();\ 
pInterface = NULL;\ 
}\
#define RELEASE(pInterface) \ 
{\ 
pInterface->Release();\ 
pInterface = NULL;\ 
}\

BOOL DOMNodeGetDefinition()
{
    BOOL bResult = FALSE;
    BSTR bstrValue;
    IXMLDOMElement *pIXMLDOMElement = NULL;
    IXMLDOMNode *pIXMLDOMNode = NULL;
    IXMLDOMDocument2 *pIXMLDOMDocument = NULL;
    HRESULT hr;

    try
    {
        // Create the document and initialize pIXMLDOMDocument.
        // Load the document (sample.xml).
        // Get the documents root.
        hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXMLDOMElement)
        {
            hr = pIXMLDOMElement->get_definition(&pIXMLDOMNode);
            if(SUCCEEDED(hr) && pIXMLDOMNode)
            {
                hr = pIXMLDOMNode->get_xml(&bstrValue);
                if(SUCCEEDED(hr) && bstrValue)
            }
        }
    }
    catch(std::exception &e)
    {
        hr = E_FAIL;
    }
    return bResult;
}
```
Sample.xml

```xml
<?xml version="1.0"?>
<SomeDoc xmlns="x-schema:collschema.xml">
    <SomeElement Id="first1"/>
</SomeElement>
</SomeDoc>
```

collschema.xml

```xml
<?xml version='1.0'?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <AttributeType name='Id' dt:type="id"/>
    <ElementType name='SomeElement' content="mixed" model="open">
        <attribute type='Id'/>
    </ElementType>

    <ElementType name='SomeDoc' content="mixed" model="open">
        <attribute type='Id'/>
        <element type='SomeElement' minOccurs='1' maxOccurs='*'/>
    </ElementType>
</Schema>
```

Output (in a message box)
Remarks

The property is read-only. Returns the node that contains the definition for the entity referenced.

The definition property is not supported when using inline schemas. This behavior is by design because the node being referenced can be deleted or modified after the document is loaded, yielding unpredictable results.

This value depends on the value of the nodeType property.

**NODE_ENTITY_REFERENCE**

Returns the node for the entity referenced. For example, given the ENTITYREF &x;, the definition property returns the node in the DOCTYPE node's entity collection that defines the corresponding ENTITY: `<!ENTITY x "y">`.

**NODE_ENTITY**

For unparsed entities, returns the NOTATION definition from the DOCTYPE node's notation collection. For example, given the ENTITY `<!ENTITY networth SYSTEM "networth NDATA XLS">`, the definition property returns the node corresponding to the notation `<!NOTATION XLS PUBLIC "http://www.microsoft.com/office/"`.

For parsed entities, returns Null.

**NODE_ATTRIBUTE**

Returns the XML-Data Schema AttributeType for a given ATTRIBUTE node. For example, the definition property for the attribute `myAttribute="123"` returns the node corresponding to the element `<Attribute name= "myAttribute">` in the schema. Null when a DTD is used or when no schema is present.
Returns the XML-Data Schema Element a given ELEMENT node. For example, given element <myelement>, the definition returns the node <ElementType name="myelement"> in the specified external schema. Returns Null when a DTD is used or when no schema is present.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button  in the upper-left corner of the page.

See Also

nodeName Property

Applies to: IXMLDOMNodeAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
doctype Property

Contains the document type node that specifies the document type definition (DTD) for this document.

[Script]

Script Syntax

```javascript
var objXMLDOMDocumentType = oXMLDOMDocument.doctype;
```

Example

The following script example creates an IXMLDOMDocumentType object, and then displays the name property of the object.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var MyDocType;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    MyDocType = xmlDoc.doctype;
    if (MyDocType != null) {
        alert(MyDocType.name);
    }
}
```

[Visual Basic]

Visual Basic Syntax

```visualbasic
Set objXMLDOMDocumentType = oXMLDOMDocument.doctype
```

Example

The following Microsoft® Visual Basic® example creates an
IXMLDOMDocumentType object, and then displays the name property of the object.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim MyDocType As IXMLDOMDocumentType
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set MyDocType = xmlDoc.doctype
    If Not MyDocType Is Nothing Then
        MsgBox MyDocType.Name
    End If
End If

[C/C++]

C/C++ Syntax

HRESULT get_doctype(
    IXMLDOMDocumentType **documentType);

Parameters

documentType [out, retval]
    For XML, the address of the node of type NODE_DOCUMENT_TYPE that
    specifies the DTD. Returns Null for for HTML documents and XML
    documents without a DTD.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned when there is no document type node.
E_INVALIDARG
    The value returned if the documentType parameter is Null.

Remarks
The property is read-only. For XML, it points to the node of type `NODE_DOCUMENT_TYPE` that specifies the DTD. It returns Null for HTML documents and XML documents without a DTD.

An XML document can contain a document type declaration before the first element in the document. It starts with the tag `<!DOCTYPE>` and can specify an external DTD.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [DOMDocument](#) | [IXMLDOMDocumentType](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**documentElement Property**

Contains the root element of the document.

[Script]

**Script Syntax**

```javascript
var objXMLDOMElement = oXMLDOMDocument.documentElement;
objXMLDOMDocument.documentElement = objXMLDOMElement;
```

**Example**

The following script example creates an IXMLDOMElement object and sets it to the root element of the document with the documentElement property. It then walks the document tree.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    for (var i=0; i<root.childNodes.length; i++) {
        alert(root.childNodes.item(i).childNodes.item(0).text);
    }
}
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
Set objXMLDOMElement = oXMLDOMDocument.documentElement
Set objXMLDOMDocument.documentElement = objXMLDOMElement
```

**Example**
The following Microsoft® Visual Basic® example creates an IXMLDOMElement object and sets it to the root element of the document with the documentElement property. It then walks the document tree.

```
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
xmlDoc.async = False
xmlDoc.Load(“books.xml”)
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox(“You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    For i = 0 To (root.childNodes.length - 1)
        MsgBox root.childNodes.Item(i).childNodes.Item(0).Text
    Next
End If

[C/C++]

C/C++ Syntax

HRESULT get_documentElement(
    IXMLDOMElement** DOMElement);
HRESULT putref_documentElement(
    IXMLDOMElement* DOMElement);

Parameters

DOMElement [out, retval][in]
    The IXMLDOMElement object that represents the single element representing the root of the XML document tree. Returns Null if no root exists.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE (for get_documentElement only)
    The value returned if there is no document element.
E_INVALIDARG (for get_documentElement only)
    The value returned if the DOMElement parameter is Null.
**Remarks**

The property is read/write. It returns an `IXMLDOMElement` that represents the single element that represents the root of the XML document tree. It returns Null if no root exists.

When setting the `documentElement` property, the specified element node is inserted into the child list of the document after any document type node. To precisely place the node within the children of the document, call the `insertBefore` method of the `IXMLDOMNode`.

The `parentNode` property is reset to the document node as a result of this operation.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`insertBefore Method` | `IXMLDOMElement` | `IXMLDOMNode` | `parentNode` Property

Applies to: [DOMDocument](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
entities Property

Contains a list of the entities declared in the <!DOCTYPE> declaration.

[Script]

Script Syntax

var objXMLDOMNamedNodeMap = oXMLDOMDocumentType.entities;

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNamedNodeMap = oXMLDOMDocumentType.entities

[C/C++]

C/C++ Syntax

HRESULT get_entities(
    IXMLDOMNamedNodeMap **entityMap);

Parameters

entityMap [out, retval]
    A list of the general entities, both external and internal, that are present in this document.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the entityMap parameter is Null.
Example

IXMLDOMEntity *GetRootEntity()
{
    HRESULT hr;
    IXMLDOMNodeMap *pIXMLDOMNodeMapEntities = NULL;
    IXMLDOMEntity *pIXMLDOMEntity = NULL;
    IXMLDOMNode *pIXMLDOMNode = NULL;
    IXMLDOMDocument *pIXMLDOMDocument = NULL;
    IXMLDOMDocumentType *pIXMLDOMDocumentType = NULL;

    try
    {
        // Create an instance of DOMDocument and initialize
        // pIXMLDOMDocument.
        // Load/create an XML fragment.
        hr = pIXMLDOMDocument->get_doctype(&pIXMLDOMDocumentType);
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXMLDOMDocumentType)
        {
            hr = pIXMLDOMDocumentType->get_entities
                (&pIXMLDOMNodeMapEntities);
            if(SUCCEEDED(hr) && pIXMLDOMNodeMapEntities)
            {
                hr = pIXMLDOMNodeMapEntities->nextNode(&pIXMLDOMNode);
                if(SUCCEEDED(hr) && pIXMLDOMNode)
                {
                    hr = pIXMLDOMNode->QueryInterface(IID_IXMLDOMEntity,
                        (void**)&pIXMLDOMEntity);
                    SUCCEEDED(hr) ? 0 : throw hr;
                    pIXMLDOMNode->Release();
                } // else pIXMLDOMNode = NULL;
            } // else pIXMLDOMNodeMapEntities = NULL;
        } // else pIXMLDOMDocumentType = NULL;
    }
    catch(...) { }
    return pIXMLDOMEntity;
} // Release pIXMLDOMEntity when finished using it.
Remarks

The property is read-only. In the following DOCTYPE declaration, for example, the node list in this property contains the entities "bat" and "bar".

```xml
<!DOCTYPE ex SYSTEM "ex.dtd" [
  <!ENTITY bat "bat">
  <!ENTITY bar "bar">
]>```

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button 📚 in the upper-left corner of the page.

See Also

Applies to: IXMLDOMDocumentType
errorCode Property

Contains the error code of the last parse error.

[Script]

Script Syntax

lValue = oXMLDOMParseError.errorCode;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it displays the error code.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert(xmlDoc.parseError.errorCode);
} else {
    alert(xmlDoc.documentElement.xml);
}
```

[Visual Basic]

Visual Basic Syntax

lValue = oXMLDOMParseError.errorCode

Example

The following Microsoft® Visual Basic® example attempts to load an XML document. If it encounters a parse error, it displays the error code.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
```
MsgBox xmlDoc.parseError.errorCode
Else
    MsgBox xmlDoc.documentElement.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_errorCode(
    long *errorCode);

Parameters

errorCode [out, retval]
    The error code returned in a decimal value.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned if no error occurred.

Remarks

Long integer. The property is read-only. It returns the error code in decimal.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMParseError
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**errorParametersCount Property**

This read-only property returns the number of error parameters used in an error message. A zero value means that the error message has no error parameters. The index is zero-based; therefore, when the number of error parameters is non-zero, the value of this property equals the index of the last parameter, minus one.

**[Script]**

**Script Syntax**

```javascript
intParams = objXMLDOMParseError2.errorParametersCount;
```

**[Visual Basic]**

**Visual Basic Syntax**

```visualbasic
intParams = objXMLDOMParseError2.errorParametersCount
```

**[C/C++]**

**C/C++ Syntax Using Smart Pointers**

```c
INT intParams = objXMLDOMParseError2->errorParametersCount;
```

**C/C++ Syntax**

```c
HRESULT get_errorParametersCount (  
    INT * intParams);
```

**Parameters**

`intParams` [out, retval]

A pointer to an integer corresponding to the number of the error parameters.

**Return Values**
S_OK
   The parameter has been retrieved successfully.
E_FAIL
   Operation failed.

Example

See the example for the errorParameters method.

Applies To

IXMLDOMParseError2 Interface

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMParseError2 Interface
**errorXPath Property**

This read-only property returns an XPath expression that locates the node where an error occurred. Currently, this property is set only when an error is encountered in schema validation (using XSD). In all other cases, the returned string will be empty.

If there are multiple validation errors, the returned XPath expression points to the first node in the document where errors occurred.

**[Script]**

**Script Syntax**

```javascript
strXPath = objXMLDOMParseError2.errorXPath;
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
strXPath = objXMLDOMParseError2.errorXPath;
```

**[C/C++]**

**C/C++ Syntax Using Smart Pointers**

```c++
_bstr_t strXPath = objXMLDOMParseError2->errorXPath;
```

**C/C++ Syntax**

```c
HRESULT get_errorXPath ( 
   BSTR *strXPath);
```

**Parameters**

- `strXPath [out,retval]`
  
  A pointer to a BSTR string that contains an XPath expression, or is empty.
Return Values

S_OK
  The property was retrieved successfully.
E_FAIL
  Failed to retrieve the property and the resultant strXPath parameter is an empty string.

Example

This example uses the same two resource files used in the allErrors example, books.xml and books.xsd. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (books.xml and books.xsd)
- JScript Code (errorXPath.js)
- Visual Basic Code (errorXPath.frm)
- C/C++ Code (errorXPath.cpp)
- Output for the errorXPath Example

Applies To

IXMLDOMParseError2 Interface

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

allErrors Property | errorParameters Method | errorParametersCount Property | validate Method | validateNode Method
var xs, xd;

testErrorXPath();

function testErrorXPath()
{
    try {
        xs = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
        xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
    }
    catch (e) {
        alert("Mirosoft XML Core Services (MSXML) 5.0 for Microsoft Offi
+"Download and install MSXML 5.0 from http://msdn.microsoft.com/
+"before continuing.");
        return;
    }

    try {
        xs.add("urn:books", "books.xsd");
        xd.schemas = xs;
        xd.async = false;
        xd.validateOnParse = false;
    }
    catch (e) {
        alert("Failed to add to schema cache: "+e.description);
        return;
    }

    try {
        xd.load("books.xml");
    }
    catch (e) {
        alert("can't load books.xml : " + e.description);
        return;
    }

    msg = "Validating DOM...\n";
    var err = xd.validate();
    if (err.errorCode != 0 ) {
        msg += "invalid dom: \n" + err.reason
        + "errorXPath:\n" + err.errorXPath
        + "\n";
    }
    else
msg += "DOM is valid:\n" + xd.xml+"\n";

msg += "\n\nValidating nodes...\n";
var nlist = xd.selectNodes("//book");
for (var node = nlist.nextNode(); node!=null; node=nlist.nextNode( 
{
    var err = xd.validateNode(node);
    if (err.errorCode != 0)
        msg += "\nNode is invalid:\n" + err.reason
        + "\nerrorXPath: \n" + err.errorXPath
        + "\n";
    else
        msg += "\nNode is valid:\n"+ node.xml+"\n";
}
alert(msg);
}

function alert(msg)
{
    WScript.echo(msg);
}

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as errorXPath.js, in the same directory where you saved books.xml and books.xsd.
4. Double click the errorXPath.js file from Windows Explorer to launch the application. Alternatively, you can type "errorXPath.js" from a command prompt.

   Note   Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in Output for the errorXPath Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Visual Basic Code (errorXPath.frm)

' Check "Microsoft XML, v50" from Project->References...
' in Visual Studio.
Set xd = CreateObject("MSXML2.DOMDocument.5.0")
Set xs = CreateObject("MSXML2.XMLSchemaCache.5.0")
' Add xs to xd.
xs.Add "urn:books", App.Path & "\books.xsd"
Set xd.schemas = xs
xd.async = False
xd.validateOnParse = False

xd.Load App.Path & "\books.xml"
msg = "Validating DOM..." & vbCrLf
Set er1 = xd.Validate
If er1.errorCode <> 0 Then
    msg = msg & "invalid dom:" & vbCrLf & er1.reason _
    & "errorXPath:" & vbCrLf & er1.errorXPath & vbCrLf
Else
    msg = msg & "DOM is valid:" & vbCrLf & vbCrLf & vbCrLf & vbCrLf
End If

msg = msg & vbCrLf & "Validating nodes..." & vbCrLf
Set nlist = xd.selectNodes("//book")
For Each node In nlist
    Set er1 = xd.validateNode(node)
    If er1.errorCode <> 0 Then
        msg = msg & vbCrLf & "Node is invalid:" & vbCrLf & vbCrLf & vbCrLf & vbCrLf
    Else
        msg = msg & vbCrLf & "Node is valid:" & vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbCrLf
    End If
Next

MsgBox msg

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as `errorXPath.vbp` to the same directory where you saved `books.xml` and `books.xsd`. Name the form file `errorXPath.frm`.
4. Create a reference to MSXML 5.0. To do this, select `References...` from the `Project` menu, then check the box for `Microsoft XML, v50`.
5. Copy the Visual Basic code listing above, and paste it into the `form_load` subroutine.
6. Execute the code by selecting `Start` from the `Run` menu.
7. Verify that your output is the same as that listed in `Output for the errorXPath Example`. 
C/C++ Code (errorXPath.cpp)

```c
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[])
{
    HRESULT hr;

    CoInitialize(NULL);

    IXMLDOMDocument3Ptr pXMLDoc;
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr))
    {
        printf("Failed to CreateInstance on an XML DOM");
        return -1;
    }

    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
    if (FAILED(hr))
    {
        printf("Failed to enable mutliple validation errors\n");
        return -1;
    }

    if(pXMLDoc->load("books.xml")!=VARIANT_TRUE)
    {
        printf("Failed to load DOM from books.xml. %s\n",
                (LPCSTR)pXMLDoc->parseError->reason);
        return -1;
    }

    IXMLDOMDocument3Ptr pXSDDoc;
    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr))
    {
        printf("Failed to CreateInstance on an XSD Doc");
        return -1;
    }

    pXSDDoc->async = VARIANT_FALSE;
    pXSDDoc->validateOnParse = VARIANT_FALSE;
    if(pXSDDoc->load("books.xsd")!=VARIANT_TRUE)
    {
        printf("Failed to load XSD from books.xsd. %s\n",
                (LPCSTR)pXSDDoc->parseError->reason);
        return -1;
    }
}
```
(LPCSTR)pXSDDoc->parseError->reason);
return -1;
}

IXMLDOMSchemaCollectionPtr pSCache;
hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr)) {
    printf("Cannot instantiate XMLSchemaCache50\n");
    return -1;
}
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr)) {
    printf("Cannot add 'urn:books' to schema cache.\n");
    return -1;
}

// Validate the entire DOM.
printf("Validating DOM...\n");
IXMLDOMParseError2Ptr pError = pXMLDoc->validate();
if (pError->errorCode != 0) {
    printf("invalid dom:\n%s", (LPCSTR)pError->reason);
    printf("errorXPath:\n%s\n", (LPCSTR)pError->errorXPath);
} else {
    printf("DOM is valid:\n%s\n", (LPCSTR)pXMLDoc->xml);
}

// Validate DOM fragments.
printf("\n\nvalidating nodes...\n");
IXMLDOMNodeListPtr pNList;
pNList = pXMLDoc->selectNodes("//book");
IXMLDOMNodePtr pNode;
for (int i=0; i<pNList->length; i++) {
    pNode = pNList->item[i];
pError = pXMLDoc->validateNode(pNode);
    if (pError->errorCode != 0) {
        printf("\nNode is invalid:\n%s", (LPCSTR)pError->reason);
        printf("errorXPath:\n%s\n", (LPCSTR)pError->errorXPath);
    } else {
        printf("\nNode is valid:\n%s\n", (LPCSTR)pNode->xml);
    }
}
Try It!

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "errorXPathProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
4. Select FileView on the project browser, and highlight ErrorXPathProj files. From the File menu, select New.
5. On the Files tab of the New dialog box, highlight C++ Source File. Then type "books.xml" in the File name text box. Be sure to put quotes around the name of the text file. Otherwise, Visual C++ will generate an unwanted .txt suffix to the XML file.
6. Click OK.
7. Copy the XML data file (books.xml), and paste it into the text file you just created.
8. Repeat steps 4-7 for the XSD listing (books.xsd).

Note You can also copy these files into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ source code above (errorXPath.cpp).
10. Build the sample by selecting Build errorXPathProj.exe from the Build menu.
11. Execute the sample application by selecting !Execute errorXPathProj.exe from the Build menu.
12. Verify that your output is the same as that listed in Output for the errorXPath Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Output for the errorXPath Example

When you build and run the errorXPath application, you should get the following output in a message box.

Validating DOM...
invalid dom:
Content for element 'review' is invalid according to the DTD/Schema.
Expecting: price.
errorXPath:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/review[1]

Validating nodes...

Node is valid:
<book id="bk001">
    <author>Hightower, Kim</author>
    <title>The First Book</title>
    <genre>Fiction</genre>
    <price>44.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>An amazing story of nothing.</review>
</book>

Node is invalid:
Content for element 'review' is invalid according to the DTD/Schema.
Expecting: price.
errorXPath:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/review[1]

Node is invalid:
Content for element 'review' is invalid according to the DTD/Schema.
Expecting: pub_date.
errorXPath:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[3]/review[1]
expr Property

Gets or sets the XML Path Language (XPath) expression.

[Script]

Script Syntax

strExpression = objXMLDOMSelection.expr;
objXMLDOMSelection.expr = strExpression;

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var selection;
xmlDoc.loadXML("<Customer><Name>Microsoft</Name></Customer>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xmlDoc.setProperty("SelectionLanguage", "XPath");
    selection = xmlDoc(selectNodes("Customer/Name");
    alert(selection.expr + " -- " + selection.item(0).xml);
}

This displays "Customer/Name -- <Name> Microsoft</Name>".

    selection.expr = "/Customer";

The selection is immediately refreshed with a list corresponding to the new expression.

    alert(selection.expr + " -- " + selection.item(0).xml);
}

This displays "/Customer -- <Customer><Name> Microsoft</Name> </Customer>".

[Visual Basic]

Visual Basic Syntax
strExpression = objXMLDOMSelection.expr
objXMLDOMSelection.expr = strExpression

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim selection As Msxml2.IXMLDOMSelection

xmlDoc.loadXML ("<Customer><Name>Microsoft</Name></Customer>")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    xmlDoc.setProperty "SelectionLanguage", "XPath"
    Set selection = xmlDoc.selectNodes("Customer/Name")
    MsgBox selection.expr + " -- " + selection.Item(0).xml

This displays "Customer/Name -- <Name> Microsoft</Name>".

    selection.expr = "/Customer"

The selection is immediately refreshed with a list corresponding to the new expression.

    MsgBox selection.expr + " -- " + selection.Item(0).xml
End If

This displays "/Customer -- <Customer><Name> Microsoft</Name> </Customer>".

[C/C++]

C/C++ Syntax

HRESULT get_expr(BSTR* expression);
HRESULT put_expr(BSTR expression);

Parameters

expression [out, retval][in]
    A string specifying the XPath expression to be returned or set.
C/C++ Return Values

S_OK
   The value returned if method is successful.

E_FAIL and formatted error message through IErrorInfo if expression is invalid.

Example

The following C/C++ example shows the resetting of the selected nodes list if the expr property is changed.

```c
#import "msxml3.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface) \
if(pInterface) \
{\n  pInterface->Release();\n  pInterface = NULL;\n}\n
#define RELEASE(pInterface) \
{\n  pInterface->Release();\n  pInterface = NULL;\n}\n
BOOL DOMSelectionExprDemo()
{
  BOOL bResult = FALSE;
  short sResult = FALSE;
  long lvalue;
  IXMLDOMSelection *pIXMLDOMSelection=NULL;
  IXMLDOMDocument2 *pIXMLDOMDocument2=NULL;
  HRESULT hr;
  BSTR bstrValue;

  try
  {\n    hr=CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_SERVER, \n        IID_IXMLDOMDocument2, (LPVOID*)(&pIXMLDOMDocument2));\n    SUCCEEDED(hr) ? 0 : throw hr;

    if(pIXMLDOMDocument2)
```
```c++
{
    hr=pIXMLDOMDocument2->put_async(VARIANT_FALSE);
    if(SUCCEEDED(hr))
    {
        hr=pIXMLDOMDocument2->load(_variant_t(_T("d:\inetpub\wwwroot\sample.xml")), &sResult);
        if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
        {
            hr=pIXMLDOMDocument2->selectNodes(_T("*/BOOK[TITLE='Lover Birds']"),
                            (IXMLDOMNodeList**)pIXMLDOMSelection);
            if(SUCCEEDED(hr))
            {
                if(SUCCEEDED(hr) && pIXMLDOMSelection)
                {
                    hr = pIXMLDOMSelection->get_expr(&bstrValue);
                    if(SUCCEEDED(hr))
                    {
                        ::MessageBox(NULL, bstrValue, _T("Current Expression"), MB_OK);
                        bResult=TRUE;
                        hr = pIXMLDOMSelection->get_length(&lvalue);
                        hr = pIXMLDOMSelection->put_expr(_T("*/BOOK"));
                        hr = pIXMLDOMSelection->get_length(&lvalue);
                    }
                }
                RELEASE(pIXMLDOMSelection);
            }
            RELEASE(pIXMLDOMDocument2);
        }
    }
    catch(...)
    {
        CHECK_AND_RELEASE(pIXMLDOMSelection);
        CHECK_AND_RELEASE(pIXMLDOMDocument2);
        DisplayErrorToUser();
    }
    return bResult;
}

File Name: d:\inetpub\wwwroot\sample.xml

<?xml version='1.0'?>
<COLLECTION xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <DATE dt:dt="datetime">1998-10-13T15:56:00</DATE>
  <BOOK>
    <TITLE>Lover Birds</TITLE>
  </BOOK>
</COLLECTION>
```
Output (in a message box)

The first message box will display "1".

The second message box will display "2". When the expression changes, the nodes in the selection are recomputed using the new expression. No changes are made to the context property.

Remarks

Immediately setting a new expression resets the state of this node list to the beginning unless the expression is invalid and an error is returned, in which case it has no effect. It does not reset the context property.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSelection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
filepos Property

Contains the absolute file position where the error occurred.

[Script]

Script Syntax

lValue = oXMLDOMParseError.filepos;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it displays the value of the filepos property.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmldoc.async = false;
xmldoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert(xmlDoc.parseError.filepos);
} else {
    alert(xmlDoc.documentElement.xml);
}
```

[Visual Basic]

Visual Basic Syntax

lValue = oXMLDOMParseError.filepos

Example

The following example attempts to load an XML document. If it encounters a parse error, it displays the value of the filepos property.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument5
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
MsgBox xmlDoc.parseError.filepos
Else
    MsgBox xmlDoc.documentElement.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_filepos(
    long *filePosition);

Parameters

filePosition [out, retval]
    The absolute file position where the error occurred.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned if no error occurred.

Remarks

Long integer. The property is read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMParseError
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**firstChild Property**

Contains the first child of the node.

**[Script]**

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNode.firstChild;
```

**Example**

The following example sets `currNode` to the first child node of the top-level node.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.firstChild;
    alert(currNode.xml);
}
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
Set objXMLDOMNode = oXMLDOMNode.firstChild
```

**Example**

The following example sets `currNode` to the first child node of the top-level node.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
```
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.firstChild
    MsgBox currNode.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_firstChild(
    IXMLDOMNode **firstChild);

Parameters

firstChild [out, retval]
    The first child node. If there are no such children, it returns Null.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned when there are no children.
E_INVALIDARG
    The value returned if the firstChild parameter is Null.

Remarks

The property is read-only. Returns the first child node. If there are no such children, returns Null.

This value depends on the value of the nodeType property.

NODE_ATTRIBUTE
NODE_DOCUMENT

NODE_DOCUMENT_FRAGMENT

NODE_DOCUMENT_TYPE

Returns the first child node. If there are no children, returns Null.

NODE_ELEMENT

NODE ENTITY

NODE ENTITY_REFERENCE

NODECDATA_SECTION

NODE COMMENT

NODE NOTATION

Returns Null. These node types cannot have children.

NODE_PROCESSING_INSTRUCTION

NODE TEXT

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMNode | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMText | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
implementation Property

Contains the IXMLDOMImplementation object for the document.

[Script]

Script Syntax

var objXMLDOMImplementation = oXMLDOMDocument.implementation;

Example

The following script example creates an IXMLDOMImplementation object.

    var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    var oImplementation;
    xmlDoc.async = false;
    xmlDoc.load("books.xml");
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    } else {
        oImplementation = xmlDoc.implementation;
    }

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMImplementation = oXMLDOMDocument.implementation

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMImplementation object.

    Dim xmlDoc As New Msxml2.DOMDocument50
    Dim oImplementation As IXMLDOMImplementation
    xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set oImplementation = xmlDoc.implementation
End If

[C/C++]

C/C++ Syntax

HRESULT get_implementation(
    IXMLDOMImplementation **impl);

Parameters

impl [out, retval]
    The IXMLDOMImplementation for this document.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the impl parameter is Null.

Example

IXMLDOMDocument * pIXMLDOMDocument = NULL;
IXMLDOMImplementation *pIXMLDOMImplementation = NULL;
VARIANT_BOOL varbFlag;
BSTR bstrOutput = NULL;
BSTR bstrFeature = ::SysAllocString(_T("MS-DOM"));

try
{
    // Initialize pIXMLDOMDocument (create a DOMDocument).
    // Load document.
    hr = pIXMLDOMDocument->get_implementation (&pIXMLDOMImplementation)
    if(SUCCEEDED(hr) && pIXMLDOMImplementation)
{
    pIXMLDOMImplementation->hasFeature(bstrFeature, _T("1.0"),
        &varbFlag);
    if(varbFlag == VARIANT_TRUE )
        bstrOutput = ::SysAllocString(_T("Feature Supported"));
    else
        bstrOutput = ::SysAllocString(_T("Feature not Supported"));

    ::MessageBox(NULL, bstrOutput, bstrFeature, MB_OK);
    pIXMLDOMImplementation->Release();
    pIXMLDOMImplementation = NULL;
    ::SysFreeString(bstrOutput);
    bstrOutput = NULL;
}
::SysFreeString(bstrFeature);
bstrFeature = NULL;
}
catch(...)
{
    if(pIXMLDOMImplementation)
        pIXMLDOMImplementation->Release();
    if(bstrOutput)
        ::SysFreeString(bstrOutput);
    if(bstrFeature)
        ::SysFreeString(bstrFeature);
        DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.

Remarks

The property is read-only. An XML Document Object Model (DOM) application can use objects from multiple implementations. This property provides access to IXMLDOMImplementation that handles this document.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMImplementation

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
input Property

Specifies which XML input tree to transform.

[Script]

Script Syntax

var objXMLDOMNode = objXSLProcessor.input;
objXMLDOMXSLProcessor.input = objXMLDOMNode;

Example

var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  xslt.stylesheet = xslDoc;
  var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
  xmlDoc.async = false;
  xmlDoc.load("books.xml");
  if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
  } else {
    xslProc = xslt.createProcessor();
    xslProc.input = xmlDoc;
    xslProc.transform();
    alert(xslProc.output);
  }
}

Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
<xsl:output method="html"/>
<xsl:param name="param1"/>
<xsl:template match="/">
  Hello
</xsl:template>
<xsl:template match="/" mode="edit">
  In Edit Mode
</xsl:template>
<xsl:template match="/" mode="view">
  In View Mode
</xsl:template>
</xsl:stylesheet>

[Visual Basic]

**Visual Basic Syntax**

```
Set objXMLDOMNode = objXSLProcessor.input
objXMLDOMXSLProcessor.input = objXMLDOMNode
```

**Example**

```vbnet
Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set xslt.stylesheet = xslDoc
  xmlDoc.async = False
  xmlDoc.Load "books.xml"
  If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
  Else
    Set xslProc = xslt.createProcessor()
    xslProc.input = xmlDoc
    xslProc.Transform
    MsgBox xslProc.output
  End If
End If
```
Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="html"/>
 <xsl:param name="param1"/>
 <xsl:template match="/">
   Hello
 </xsl:template>
 <xsl:template match="/" mode="edit">
   In Edit Mode
 </xsl:template>
 <xsl:template match="/" mode="view">
   In View Mode
 </xsl:template>
</xsl:stylesheet>
```

[C/C++]

C/C++ Syntax

```c
HRESULT get_input (VARIANT * pVar);
HRESULT put_input (VARIANT pVar);
```

Parameters

`pVar` [out, retval][in]

The input XML to which to apply the transformation. This can be a URL (string/BSTR), a DOMDocument object, an IXMLDOMNode or any object that supports ISequentialStream, IStream, or IPersistStream. If applying the transformation to an IXMLDOMNode, any node type and any threading model can be specified. If this argument is Null, the existing input will be cleared.

C/C++ Return Values

E_FAIL
The value returned if readyState is READYSTATE_INTERACTIVE.
E_INVALIDARG
The value returned if the variant is not VT_DISPATCH or VT_UNKNOWN.
E_NOINTERFACE
The value returned if the argument does not support IStream or
Remarks

Setting this property resets the state of the processor.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMNode

Applies to: IXSLProcessor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
item Property

Returns the item from the collection with the specified index. The index starts at zero.

[Script]

Script Syntax

var objErr = IXMLDOMParseErrorCollection.item(index);

[Visual Basic]

Visual Basic Syntax

Set objErr = IXMLDOMParseErrorCollection.item(index);

[C/C++]

C/C++ Syntax Using Smart Pointers

IXMLDOMParseError2Ptr objErr=objIXMLDOMParseErrorCollection->item[index];

C/C++ Syntax

HRESULT get_item(
    LONG index,
    IXMLDOMParseError2 **objErr);

Parameters

index
    A LONG value specifying a zero-based index as the input.

objErr
    A pointer to an IXMLDOMParseError2 object corresponding to the specified item index in the error collection.
Return Values

S_OK
   The value returned if successful. The resultant objErr parameter is not NULL.

E_FAIL
   The value returned if the index is out of bounds.

Example

The following code performs an XSD validation on an XML document that has two invalid <book> nodes. The code then iterates through the resultant error collection, and examines each item retrieved.

This example uses the same two resource files used in the allErrors example. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (books.xml and books.xsd)
- JScript Code (item.js)
- Visual Basic Code (item.frm)
- C/C++ Code (item.cpp)
- Output for the item Example

Applies To

IXMLDOMParseErrorCollection Interface

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

next Property | reset Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
JScript Code (item.js)

```javascript
var xs, xd;

main();

function main()
{
    try {
        xs = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
        xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
    }
    catch (e) {
        alert("Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office is not installed.
        Download and install MSXML 5.0 from http://msdn.microsoft.com
        before continuing.");
        return;
    }

    try {
        xs.add("urn:books", "books.xsd");
    }
    catch (e) {
        alert("Failed to add to schema cache: "+e.description);
        return;
    }

    try {
        xd.schemas = xs;
        xd.async = false;
        xd.validateOnParse = false;
        xd.load("books.xml");
        xd.setProperty("MultipleErrorMessages", true);
    }
    catch (e) {
        alert("can't load books.xml : " + e.description);
        return;
    }

    var err = xd.validate();
    if (err.errorCode != 0 ) {
        for (i=0; i<err.allErrors.length; i++)
        {
            var e=err.allErrors.item(i);
            alert("errorItem["+i+"]: "+e.reason+"\n");
        }
    }
```
function alert(msg)
{
    WScript.echo(msg);
}

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd in the same directory where you saved books.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as item.js, in the same directory where you saved books.xml and books.xsd.
4. Double click the item.js file from Windows Explorer to launch the application. Alternatively, you can type "item.js" from a command prompt.

   **Note** Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that the output is the same as that listed in Output for the item Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Visual Basic Code (item.frm)

Dim er As IXMLDOMParseError2
Dim errs As IXMLDOMParseErrorCollection
Dim xd As New DOMDocument50
Dim xs As New XMLSchemaCache50

' Add xs to xd.
xs.Add "urn:books", App.Path & "\books.xsd"
Set xd.schemas = xs
xd.async = False
xd.validateOnParse = False
xd.setProperty "MultipleErrorMessages", True

' Load the XML file to xd.
xd.Load App.Path & "\books.xml"

' Validate the XML.
Set er = xd.Validate
If er.errorCode <> 0 Then
    Set errs = er.allErrors
    msg = ""
    For i = 0 To errs.length - 1
        Set er = errs.item(i)
        msg = msg & "errorItem[" & i & "]:	" & er.reason & vbCrLf
    Next
Else
    msg = "DOM is valid:" & vbCrLf & xd.xml
End If
MsgBox msg

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd in the same directory where you saved books.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as item.vbp to the same directory where you saved books.xml and books.xsd. Name the form file item.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.
5. Copy the Visual Basic code listing above, and paste it into the form_load
subroutine.
6. Execute the code selecting **Start** from the **Run** menu.
7. Verify that your output is the same as that listed in the *Output for the item Example*. 
C/C++ Code (item.cpp)

```c
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    HRESULT hr;

    CoInitialize(NULL);

    IXMLDOMDocument3Ptr pXMLDoc;
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XML DOM\n");
        return -1;
    }
    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
    if (FAILED(hr)) {
        printf("Failed to enable multiple validation errors\n");
        return -1;
    }
    if (pXMLDoc->load("books.xml") != VARIANT_TRUE) {
        printf("Failed to load DOM from books.xml. %s\n",
               (LPCSTR)pXMLDoc->parseError->reason);
        return -1;
    }

    IXMLDOMDocument3Ptr pXSDDoc;
    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XSD Doc\n");
        return -1;
    }
    pXSDDoc->async = VARIANT_FALSE;
    pXSDDoc->validateOnParse = VARIANT_FALSE;
    if (pXSDDoc->load("books.xsd") != VARIANT_TRUE) {
        printf("Failed to load XSD from books.xsd. %s\n",
               (LPCSTR)pXSDDoc->parseError->reason);
        return -1;
    }
    return 0;
}
```
IXMLDOMSchemaCollectionPtr pSCache;
hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr)) {
    printf("Cannot instantiate XMLSchemaCache50\n");
    return -1;
}
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr)) {
    printf("Cannot add 'urn:books' to schema cache.\n");
    return -1;
}

// Validate the entire DOM object.
IXMLDOMParseError2Ptr pError = pXMLDoc->validate();
if (pError->errorCode != 0) {
    IXMLDOMParseErrorCollectionPtr pErrs;
    pErrs = pError->allErrors;
    for (int i=0; i<pErrs->length; i++)
    {
        pError = pErrs->item[i];
        printf("errorItem[%d]: %s\n", i, (LPCSTR)pError->reason);
    }
    else
    {
        printf("DOM is valid: \n%s\n", (LPCSTR)pXMLDoc->xml);
    }
    pXMLDoc.Release();
pXSDDoc.Release();
pSCache.Release();
pError.Release();
CoUninitialize();

    return 0;
}

Try It!

2. From the **File** menu, select **New**. On the **Projects** tab of the **New** dialog box that appears, select **Win32 Console Application** in the left pane. Then type "itemProj" in the **Project name** field. For the project **Location** field, either accept the default setting or choose another location. Click **OK**.

3. The **Win32 Console Application** property page will appear. For the type of Console Application, select **An empty project** and click **Finish**. When the **New Project Information** box displays, click **OK**.

4. Select **FileView** on the project browser, and highlight **itemProj files**. From the **File** menu, select **New**.

5. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "books.xml" in the **File name** text box.

6. Click **OK**.

7. Copy the XML data file (**books.xml**), and paste it into the text file you just created.

8. Repeat steps 4-7 for the XSD listing (**books.xsd**).

   **Note** You can also copy these files into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ listing above (**item.cpp**).

10. Build the sample by selecting **Build itemProj.exe** from the **Build** menu.

11. Execute the sample application by selecting **!Execute itemProj.exe** from the **Build** menu.

12. Verify that your output is the same as that listed in the **Output for the item Example**.
Output for the item Example

When you build and run the item code, you should get the following output.

errorItem[0]: Content for element 'review' is invalid according to t  Expecting: price.

errorItem[1]: Content for element 'review' is invalid according to t  Expecting: pub_date.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
lastChild Property

Returns the last child node.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNode.lastChild;
```

Example

The following script example creates a new IXMLDOMNode (element) object, and then inserts it before the last child of the top-level node.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var newNode;
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    var root = xmlDoc.documentElement;
    alert(root.xml);
    var newNode = xmlDoc.createNode(1, "VIDEOS", "");
    currNode = root.insertBefore(newNode, root.lastChild);
    alert(root.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objXMLDOMNode = oXMLDOMNode.lastChild
```

Example
The following Microsoft® Visual Basic® example creates a new IXMLDOMNode (element) object, and then inserts it before the last child of the top-level node.

```vba
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim newNode As IXMLDOMNode
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    MsgBox root.xml
    Set newNode = xmlDoc.createNode(NODE_ELEMENT, "VIDEOS", "")
    Set currNode = root.insertBefore(newNode, root.lastChild)
    MsgBox root.xml
End If
```

[C/C++]

C/C++ Syntax

```c++
HRESULT get_lastChild(
    IXMLDOMNode **lastChild);
```

**Parameters**

*lastChild [out, retval]*

The last child node. If there are no children, it returns Null.

**C/C++ Return Values**

S_OK

The value returned if successful.

S_FALSE

The value returned when there are no children.

E_INVALIDARG

The value returned if the *lastChild* parameter is Null.
Remarks

The property is read-only. If there are no children, it returns Null.

This value depends on the value of the nodeType property.

NODE_ATTRIBUTE

NODE_DOCUMENT

NODE_DOCUMENT_FRAGMENT

NODE_DOCUMENT_TYPE

NODE_ELEMENT

NODE_ENTITY

NODE_ENTITY_REFERENCE

NODE_CDATA_SECTION

NODE_COMMENT

NODE_NOTATION

NODE_PROCESSING_INSTRUCTION

NODE_TEXT

Returns the last child node. If there are no children, returns Null.

Returns Null. These node types cannot have children.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument |
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
length Property
(IXMLDOMCharacterData)

Specifies the length, in characters, of the data.

[Script]

Script Syntax

lValue = oXMLDOMCharacterData.length;

Example

The following example creates an IXMLDOMComment object and then assigns the length property to a variable.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
var lValue;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    comment = xmlDoc.createComment("Hello World!");
    lValue = comment.length;
    alert(lValue);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
lValue = oXMLDOMCharacterData.length
```

Example
The following example creates an IXMLDOMComment object and then assigns the length property to a variable.

```vba
Dim xmlDoc As New Msxml2.DOMDocument50
Dim comment As IXMLDOMComment
Dim lValue As Long
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set comment = xmlDoc.createComment("Hello World!")
    lValue = comment.length
    MsgBox lValue
End If

[C/C++]

C/C++ Syntax

HRESULT get_length(
    long *dataLength);

Parameters

dataLength [out, retval]
    The length, in Unicode characters, of the string present in the data property.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the dataLength parameter is Null.

Remarks

Long integer. The property is read-only. It specifies the length, in Unicode characters, of the string present in the data property.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[data Property](#)

Applies to: [IXMLDOMCDATASection](#) | [IXMLDOMCharacterData](#) | [IXMLDOMComment](#) | [IXMLDOMText](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**length Property**  
*(IXMLDOMNamedNodeMap)*

Indicates the number of items in the collection.

[Script]

**Script Syntax**

\[ lValue = oXMLDOMNamedNodeMap.length; \]

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, namedNodeMap;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    namedNodeMap = nodeBook.attributes;
    alert(namedNodeMap.length);
}
```

[Visual Basic]

**Visual Basic Syntax**

\[ lValue = oXMLDOMNamedNodeMap.length \]

**Example**

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim namedNodeMap As IXMLDOMNamedNodeMap
```
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    Set namedNodeMap = nodeBook.Attributes
    MsgBox(namedNodeMap.length)
End If

[C/C++]

C/C++ Syntax

HRESULT get_length(
    long *listLength);

Parameters

listLength [out, retval]
    The number of items in the collection.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the listLength is Null.

Example

IXMLDOMNode *pIXMLDOMNode = NULL;
IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMap = NULL;
IXMLDOMElement *pIXMXMLDOMElement = NULL;
long lLength;
TCHAR strTemp[10];
HRESULT hr;
IXMLDOMDocument *pIXMXMLDOMDocument = NULL;

try
{  
  // Create an instance of DOMDocument and initialize pIXMLDOMDocum
  // Load/create an XML fragment.  
  hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMEleme
  SUCCEEDED(hr) ? 0 : throw hr;

  if(pIXMLDOMElement)
  {
    hr = pIXMLDOMElement->get_attributes(&pIXMLOMNamedNodeMap);
    if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMap)
    {
      hr = pIXMLDOMNamedNodeMap->get_length(&lLength);
      if(SUCCEEDED(hr))
      {
        _itot(lLength, strTemp, 10);
        ::MessageBox(NULL, strTemp, _T("Number of Attributes"),
        }
        pIXMLDOMNamedNodeMap->Release();
        pIXMLDOMNamedNodeMap = NULL;
      }
    }
    pIXMLDOMElement->Release();
    pIXMLDOMElement = NULL;
  }
  catch(...)
  {
    if(pIXMLDOMElement)
      pIXMLDOMElement->Release();
    if(pIXMLDOMNamedNodeMap)
      pIXMLDOMNamedNodeMap->Release();
      DisplayErrorToUser();
    }
  // Release pIXMLDOMDocument when finished with it.

  Remarks

  Long integer. The property is read-only.

  To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button \[\] in the upper-left corner of the page.

  See Also

  Applies to: IXMLDOMNamedNodeMap
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
length Property
(IXMLDOMNodeList)

Indicates the number of items in the collection.

[Script]

Script Syntax

lValue = oXMLDOMNodeList.length;

Example

The following script example creates an IXMLDOMNodeList object and then uses its length property to support iteration.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    objNodeList = xmlDoc.getElementsByTagName("author");
    for (var i=0; i<objNodeList.length; i++) {
        alert(objNodeList.item(i).text);
    }
}
```

[Visual Basic]

Visual Basic Syntax

lValue = oXMLDOMNodeList.length

Example
The following Microsoft® Visual Basic® example creates an IXMLDOMNodeList object and then uses its length property to support iteration.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLDOMNodeList
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set objNodeList = xmlDoc.getElementsByTagName("author")
    For i = 0 To (objNodeList.length - 1)
        MsgBox objNodeList.Item(i).Text
    Next
End If

[C/C++]

C/C++ Syntax

HRESULT get_length(
    long *listLength);

Parameters

listLength [out, retval]
    The number of items in the collection.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the listLength parameter is Null.

Remarks

Long integer. The property is read-only.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [IXMLDOMNodeList](#) | [IXMLDOMSelection](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
length Property
(IXMLDOMParseErrorCollection)

This read-only property returns the number of items in the error collection.

[Script]

Script Syntax

count = objIXMLDOMParseErrorCollection.length;

[Visual Basic]

Visual Basic Syntax

count = objIXMLDOMParseErrorCollection.length;

[C/C++]

C/C++ Syntax Using Smart Pointers

count = objIXMLDOMParseErrorCollection->length;

C/C++ Syntax

HRESULT get_length(
    LONG *count);

Parameters

count [out, retval]
    A pointer to a LONG value holding the number of items in the collection.

Return Values
The value returned if successful. This property never fails.

Example

The following code performs an XSD validation on an XML document that has two invalid <book> nodes. The code then outputs the number of errors in the resultant error collection.

The example uses the same two resource files used in the allErrors example. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (books.xml and books.xsd)
- JScript Code (length.js)
- Visual Basic Code (length.frm)
- C/C++ Code (length.cpp)
- Output for the length Example

Applies To

IXMLDOMParseErrorCollection Interface

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

item Property | next Property | reset Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer...
window.onload = function(){
  var xs, xd;
  main();

  function main()
  {
    try {
      xs = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
      xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
    } catch (e) {
      alert("Mirossoft XML Core Services (MSXML) 5.0 for Microsoft Offi
      +"Download and install MSXML 5.0 from http://msdn.microsoftof
      +"before continuing.");
      return;
    }

    try {
      xs.add("urn:books", "books.xsd");
    } catch (e) {
      alert("Failed to add to schema cache: "+e.description);
      return;
    }

    try {
      xd.schemas = xs;
      xd.async = false;
      xd.validateOnParse = false;
      xd.load("books.xml");
      xd.setProperty("MultipleErrorMessages", true);
    } catch (e) {
      alert("can't load books.xml : " + e.description);
      return;
    }

    var err = xd.validate();
    if (err.errorCode != 0 ) {
      errsCount = err.allErrors.length;
      alert("there are "+errsCount+ " errors in the collection");
    } else
      alert("valid dom as follows:\n" + xd.xml);
  }
};
function alert(msg)
{
    WScript.echo(msg);
}

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as length.js, in the same directory where you saved books.xml and books.xsd.
4. Double click the length.js file from Windows Explorer to launch the application. Alternatively, you can type "length.js" from a command prompt.

   Note   Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in Output for the length Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Visual Basic Code (length.frm)

```vbnet
Dim er As IXMLDOMParseError2
Dim errs As IXMLDOMParseErrorCollection
Dim xd As New DOMDocument50
Dim xs As New XMLSchemaCache50

' Add xs to xd.
xs.Add "urn:books", App.Path & "\books.xsd"
Set xd.schemas = xs
xd.async = False
xd.validateOnParse = False
xd.setProperty "MultipleErrorMessages", True

' Load the XML file to xd.
xd.Load App.Path & "\books.xml"

' Validate the XML.
Set er = xd.Validate
If er.errorCode <> 0 Then
    Set errs = er.allErrors
    errsCount = errs.length
    MsgBox "there are " & errsCount & " errors in the collection"
Else
    msg = "DOM is valid:" & vbCrLf & xd.xml
End If
```

Try It!

1. Copy the XML data ([books.xml](#)), and paste it into a text file. Save the file as [books.xml](#).
2. Copy the XSD listing ([books.xsd](#)), and paste it into a text file. Save the file as [books.xsd](#) in the same directory where you saved [books.xml](#).
3. Create a Standard EXE project in Visual Basic Save the empty project as [length.vbp](#) to the same directory where you saved [books.xml](#) and [books.xsd](#). Name the form file [length.frm](#).
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.
5. Copy the Visual Basic code above, and paste it into the `form_load` subroutine.
6. Execute the code selecting **Start** from the **Run** menu.
7. Verify that your output is the same as that listed in **Output for the length**
Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
```c
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
  HRESULT hr;
  CoInitialize(NULL);

  IXMLDOMDocument3Ptr pXMLDoc;
  hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
  if (FAILED(hr)) {
    printf("Failed to CreateInstance on an XML DOM");
    return -1;
  }
  pXMLDoc->async = VARIANT_FALSE;
  pXMLDoc->validateOnParse = VARIANT_FALSE;
  hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
  if (FAILED(hr)) {
    printf("Failed to enable multiple validation errors\n");
    return -1;
  }
  if(pXMLDoc->load("books.xml")!=VARIANT_TRUE) {
    printf("Failed to load DOM from books.xml. %s\n",
       (LPCSTR)pXMLDoc->parseError->reason);
    return -1;
  }

  IXMLDOMDocument3Ptr pXSDDoc;
  hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
  if (FAILED(hr)) {
    printf("Failed to CreateInstance on an XSD Doc");
    return -1;
  }
  pXSDDoc->async = VARIANT_FALSE;
  pXSDDoc->validateOnParse = VARIANT_FALSE;
  if(pXSDDoc->load("books.xsd")!=VARIANT_TRUE) {
    printf("Failed to load XSD from books.xsd. %s\n",
```
(LPCSTR)pXSDDoc->parseError->reason);
    return -1;
}

IXMLDOMSchemaCollectionPtr pSCache;
hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr)) {
    printf("Cannot instantiate XMLSchemaCache50\n");
    return -1;
}
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr)) {
    printf("Cannot add 'urn:books' to schema cache.\n");
    return -1;
}

// Validate the entire DOM object.
IXMLDOMParseError2Ptr pError =pXMLDoc->validate();
if (pError->errorCode != 0) {
    printf("there are %d errors in the collection\n",
        pError->allErrors->length);
} else
    printf("DOM is valid:\n%s\n",(LPCSTR)pXMLDoc->xml);

pXMLDoc.Release();
pXSDDoc.Release();
pSCache.Release();
pError.Release();
CoUninitialize();

return 0;

Try It!

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "lengthProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the
**New Project Information** box displays, click OK.

4. Select **FileView** on the project browser, and highlight **lengthProj files**.
   From the **File** menu, select **New**.

5. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "books.xml" in the **File name** text box.

6. Click **OK**.

7. Copy the XML data file (**books.xml**), and paste it into the text file you just created.

8. Repeat steps 4-7 for the XSD listing (**books.xsd**).

   **Note** You can also copy these files into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ listing above (**length.cpp**).

10. Build the sample by selecting **Build lengthProj.exe** from the **Build** menu.

11. Execute the sample application by selecting !**Execute lengthProj.exe** from the **Build** menu.

12. Verify that your output is the same as that listed in **Output for the length Example**.
Output for the length Example

When you build and run the length code, you should get the following output in a message box.

there are 2 errors in the collection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
length Property
(XMLSchemaCache/IXMLSchemaCo|

Retrns the number of namespaces currently in the collection.

**Note**  The namespaces collection only includes namespaces explicitly declared as attribute values using the `xmlns` attribute. If a namespace is not explicitly used in an XML document when it loads into the DOM, it will not be added to the collection.

[Script]

**Script Syntax**

```
1Value = objXMLDOMSchemaCollection.length;
```

[Visual Basic]

**Visual Basic Syntax**

```
1Value = objXMLDOMSchemaCollection.length
```

[C/C++]

**C/C++ Syntax**

```
HRESULT length(long * length);
```

**Parameters**

*length [out, retval]*

The number of namespaces in the collection.

**C/C++ Return Values**
S_OK

The value returned if successful.

Remarks

Long integer. The property is read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
line Property

Specifies the line number that contains the error.

[Script]

Script Syntax

lValue = oXMLDOMParseError.line;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it displays the line where the error occurred.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert("A parse error occurred on line " + xmlDoc.parseError.line);
} else {
    alert(xmlDoc.documentElement.xml);
}
```

[Visual Basic]

Visual Basic Syntax

lValue = oXMLDOMParseError.line

Example

The following Microsoft® Visual Basic® example attempts to load an XML document. If it encounters a parse error, it displays the line where the error occurred.

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
```
xmlDoc.Load ("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
   MsgBox ("A parse error occurred on line " & xmlDoc.parseError.Line
Else
   MsgBox xmlDoc.documentElement.xml
End If

[C/C++]

**C/C++ Syntax**

HRESULT get_line(
   long *lineNumber);

**Parameters**

*lineNumber* [out, retval]
   The line number where the error occurred.

**C/C++ Return Values**

S_OK
   The value returned if successful.
S_FALSE
   The value returned if no error occurred.

**Remarks**

Long integer. The property is read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: **IXMLDOMParseError**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
linepos Property

Contains the character position within the line where the error occurred.

[Script]

Script Syntax

lValue = oXMLDOMParseError.linepos;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it displays the line and position within that line where the error occurred.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert("A parse error occurred on line " + xmlDoc.parseError.line + " at position " + xmlDoc.parseError.linepos);
} else {
    alert(xmlDoc.documentElement.xml);
}
```

[Visual Basic]

Visual Basic Syntax

lValue = oXMLDOMParseError.linepos

Example

The following Microsoft® Visual Basic® example attempts to load an XML document. If it encounters a parse error, it displays the line and position within that line where the error occurred.

```vbnet
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert("A parse error occurred on line " + xmlDoc.parseError.line + " at position " + xmlDoc.parseError.linepos);
} else {
    alert(xmlDoc.documentElement.xml);
}
```
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.Load("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
    MsgBox("A parse error occurred on line " & xmlDoc.parseError.Line & " at position " & xmlDoc.parseError.linepos)
Else
    MsgBox xmlDoc.documentElement.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_linepos(
    long *linePosition);

Parameters

linePosition [out, retval]
    The character position within the line where the error occurred.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned if no error occurred.

Remarks

Long integer. The property is read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☑ in the upper-left corner of the page.

See Also

Applies to: IXMLDOMParseError
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
name Property (IXMLDOMAttribute)

Contains the attribute name.

[Script]

Script Syntax

strValue = oXMLDOMAttribute.name;

Example

The following script example creates an IXMLDOMAttribute object from an attribute of the first child of the root, and then displays the value of its name.

```javascript
var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var root;
var objDOMatt;
xmlDoc.async = false;
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xmlDoc.load("books.xml");
    root = xmlDoc.documentElement;
    objDOMatt = root.firstChild.attributes.item(0);
    alert(objDOMatt.name);
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMAttribute.name

Example
The following Microsoft® Visual Basic® example creates an IXMLDOMAttribute object from an attribute of the first child of the root, and then displays the value of its name.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim objDOMatt As IXMLDOMAttribute
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set objDOMatt = root.firstChild.Attributes.Item(0)
    MsgBox(objDOMatt.Name)
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_name(
    BSTR *attributeName);
```

**Parameters**

`attributeName` [out, retval]

Name of the attribute. The value is the same as the nodeName property of the IXMLDOMNode object.

**C/C++ Return Values**

S_OK  
Value returned if successful.

S_FALSE  
Value when returning Null.

E_INVALIDARG  
Value returned if the attributeName parameter is Null.
Remarks

String. The property is read-only. It returns the name of the attribute. The value is the same as the nodeName property of the IXMLDOMNode.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeName Property | IXMLDOMNode

Applies to: IXMLDOMAttribute
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
name Property (IXMLDOMDocumentType)

Contains the name of the document type.

[Script]

Script Syntax

strValue = oXMLDOMDocumentType.name;

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMDocumentType.name

[C/C++]

C/C++ Syntax

HRESULT get_name(  
    BSTR *rootName);

Parameters

rootName [out, retval]  
    When specified, this corresponds to the single root element of the document tree.

C/C++ Return Values

S_OK  
    Value returned if successful.
S_FALSE
Value when returning Null.

Example

BOOL DOMDocGetDocType()
{
    BOOL bResult = FALSE;
    IXMLDOMDocumentType *pIXMLDOMDocumentType = NULL;
    IXMLDOMDocument *pIXMLDOMDocument = NULL;
    IXMLDOMDocumentType *pIXMLDOMDocumentType = NULL;
    BSTR bstrDTDName;
    HRESULT hr;

    try
    {
        // Create an instance of DOMDocument and initialize
        // pIXMLDOMDocument.
        // Load/create an XML fragment.
        hr = pIXMLDOMDocument->get_doctype(&pIXMLDOMDocumentType);
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXMLDOMDocumentType)
        {
            pIXMLDOMDocumentType->get_name(&bstrDTDName);
            ::MessageBox(NULL, bstrDTDName, _T("Doc Type"), MB_OK);
            bResult = TRUE;
            pIXMLDOMDocumentType->Release();
        }
    }
    catch(...)
    {
        if(pIXMLDOMDocumentType)
            pIXMLDOMDocumentType->Release();
        DisplayErrorToUser();
    }
    return bResult;
}

Remarks

String. The property is read-only. When specified, this corresponds to the single root element of the document tree.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ▼ in the upper-left corner of the page.
See Also

Applies to: IXMLDOMDocumentType
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
namespaces Property

Returns the list of namespaces used in the document as XMLSchemaCache.

[Script]

Script Syntax

objXMLDOMSchemaCollection = objIXMLDOMDocument2.namespaces;

Example

```javascript
// Load a document (and any schemas referenced via X-Schema).
xmlDoc.load("somedata.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    // Get the namespace of the root element.
    var namespaceURI = xmlDoc.documentElement.namespaceURI;
    // Find the schema associated with the namespace.
    var schema = xmlDoc.namespaces.get(namespaceURI);
}
```

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMSchemaCollection = objIXMLDOMDocument2.namespaces

Example

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
Dim namespaceURI As String
Dim schema As XML SchemaCache50
' Load a document (and any schemas referenced via X-Schema).
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
```
MsgBox("You have error " & myErr.reason)
Else
    ' Get the namespace of the root element.
    namespaceURI = xmlDoc.documentElement.namespaceURI
    ' Find the schema associated with the namespace.
    Set schema = xmlDoc.namespaces.get(namespaceURI)
End If

[C/C++]

C/C++ Syntax

HRESULT get_namespaces (IXMLDOMSchemaCollection ** namespaceCollecti

Parameters

namespaceCollection [out, retval]
    Collection of all namespaces for the document.

Remarks

If there are no schemas, it returns an empty collection.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
namespaceURI Property (IXMLDOMNode)

Returns the Uniform Resource Identifier (URI) for the namespace.

[Script]

**Script Syntax**

```javascript
strValue = oXMLDOMNode.namespaceURI;
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
strValue = oXMLDOMNode.namespaceURI
```

[C/C++]

**C/C++ Syntax**

```cpp
HRESULT get_namespaceURI(
    BSTR *namespaceURI);
```

**Parameters**

`namespaceURI [out, retval]`

URI for the namespace. This refers to the "uuu" portion of the namespace declaration `xmlns:nnn="uuu"`.

**C/C++ Return Values**

- S_OK
  
  Value returned if successful.
- S_FALSE
Value returned when there is no namespace.

E_INVALIDARG
Value returned if the namespaceURI parameter is Null.

Remarks

String. The property is read-only. Returns the URI for the namespace. This refers to the "uuu" portion of the namespace declaration xmlns:nnn="uuu".

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMEElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
namespaceURI Property (XMLSchemaCache/IXMLSchemaCollection)

Returns the namespace at the specified index.

[Script]

Script Syntax

strNamespaceURI = objXMLDOMSchemaCollection.namespaceURI(index);

Parameters

index
The index between 0 and (count-1).

[Visual Basic]

Visual Basic Syntax

strNamespaceURI = objXMLDOMSchemaCollection.namespaceURI(index)

Parameters

index
The index between 0 and (count-1).

[C/C++]

C/C++ Syntax

HRESULT namespaceURI(long index, BSTR* length);

Parameters
index [in]
The index between 0 and count-1.

length [out, retval]
The namespace URI at the given index. This string is a normalized version of the one provided in the add method (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

C/C++ Return Values

E_INVALIDARG
Value returned if the index is out of range.

Remarks

String. The namespaceURI property allows you to iterate through the collection to discover its contents. You can then use the get method (namespace) to see if there is an associated schema.

This is the default collection property, so you can do the following.

var uri = xmldoc.namespaces(1)

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

add Method | get Method

Applies to: XMLSchemaCache/IXMLDOMSchemaCollection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
_newEnum Property

Returns the enumerator object for the error collection, so that items in the collection can be enumerated using the methods and properties of this interface. The newEnum property is implemented to support the enumeration of the error collection in Visual Basic For Each... In... Next constructions.

[Visual Basic]

**Visual Basic Syntax**

This property is used automatically in For Each... In... Next constructions.

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_newEnum(   
    IUnknown **ppUnk);
```

**Parameters**

`ppUnk [out,retval]`

Pointer to an enumerator object for the error collection.

**Return Values**

`S_OK`

The enumerator object was retrieved successfully.

`E_FAIL`

The operation failed.

**Example**

This example uses a For Each... In... Next construction that implicitly calls
the `_newEnum` property to iterate an error collection.

This example uses the same two resource files used in the `allErrors` example. The source code for the sample is in Visual Basic.

- Resource Files (books.xml and books.xsd)
- Visual Basic Code (enumErrors.frm)
- Output for the enumErrors Example

**Applies To**

IXMLDOMParseErrorCollection Interface

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

item Property | length Property | next Property | reset Method
Visual Basic Code (enumErrors.frm)

Dim er, e As IXMLDOMParseError2
Dim errs As IXMLDOMParseErrorCollection
Dim xd As New DOMDocument50
Dim xs As New XMLSchemaCache50

' Add xs to xd.
xs.Add "urn:books", App.Path & "\books.xsd"
Set xd.schemas = xs
xd.async = False
xd.validateOnParse = False
xd.setProperty "MultipleErrorMessages", True

' Load the XML file to xd.
xd.Load App.Path & "\books.xml"

' Validate the XML.
Set er = xd.Validate
If er.errorCode <> 0 Then
  Set errs = er.allErrors
  errsCount = errs.length
  msg = "Error as returned from validate():" & vbCrLf _
  & vbCrLf & " Error Code:" & vbCrLf & er.errorCode _
  & vbCrLf & " Error reason:" & vbCrLf & er.reason _
  & vbCrLf & " Error location:" & vbCrLf & er.errorXPath _
  & vbCrLf & " Error Count:" & vbCrLf & errsCount & vbCrLf _
  & vbCrLf & "Error items from the allErrors collection:"

  ' Enumerate the error collection, invoking _newEnum implicitly
  i = 0
  For Each e In errs
    msg = msg & vbCrLf & vbCrLf & "ErrorItem[" & i & "]":" & vbCrLf _
    & vbCrLf & " reason:" & vbCrLf & e.reason _
    & vbCrLf & " location:" & vbCrLf & e.errorXPath
    i = i + 1
  Next
Else
  msg = msg & "DOM is valid:" & vbCrLf & vbCrLf & vbCrLf & "xml"
End If

MsgBox msg

Try It!
1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as enumErrors.vbp to the same directory where you saved books.xml and books.xsd. Name the form file enumErrors.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.
5. Copy the Visual Basic code above, and paste it into the form_load subroutine.
6. Execute the code by selecting Start from the Run menu.
7. Verify that your output is the same as that listed in Output for the enumErrors Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Output for the enumErrors Example

When you build and run the enumErrors code, you should get the following output in a message box.

Error as returned from validate():

Error Code : -1072898028
Error reason : Content for element 'review' is invalid according to the DTD/Schema. Expecting: price.

Error location:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/review
Errors Count : 2

Error items from the allErrors collection:

ErrorItem[0] :
  reason: Content for element 'review' is invalid according to the DTD/Schema. Expecting: price.
  location: /*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/review

ErrorItem[1] :
  reason: Content for element 'review' is invalid according to the DTD/Schema. Expecting: pub_date.
  location: /*[local-name()="books" and namespace-uri()="urn:books"]/book[3]/review
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
next Property

Returns the next item in the error collection object.

[Script]

Script Syntax

var objErr = objIXMLDOMParseErrorCollection.next;

[Visual Basic]

Visual Basic Syntax

Set objErr = objIXMLDOMParseErrorCollection.next;

[C/C++]

C/C++ Syntax Using Smart Pointers

IXMLDOMParseError2Ptr objErr = objIXMLDOMParseErrorCollection->next;

C/C++ Syntax

HRESULT get_next(IXMLDOMParseError2 **objErr);

Parameters

objErr [out, retval]
A pointer to an IXMLDOMParseError2 object from the error collection.

Return Values

S_OK
The value returned if successful. The resultant objErr parameter returns a valid IXMLDOMParseError2 object.
**E_FAIL**

The value returned if the end of the list has been reached. The resultant `objErr` parameter is NULL.

**Example**

This example uses the same two resource files used in the allErrors example. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- [Resource Files (books.xml and books.xsd)]
- [JScript Code (next.js)]
- [Visual Basic Code (next.frm)]
- [C/C++ Code (next.cpp)]
- [Output for the next Example]

**Applies To**

[IXMLDOMParseErrorCollection Interface]

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
JScript Code (next.js)

```javascript
var xs, xd;
main();

function main()
{
    try {
        xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
    }
    catch (e) {
        alert("Mirosoft XML Core Services (MSXML) 5.0 for Microsoft Offi
"+"Download and install MSXML 5.0 from http://msdn.microsoft
"+"before continuing.");
        return;
    }

    try {
        xd.async = false;
        xd.validateOnParse = false;
        xd.load("books.xml");
        xd.setProperty("MultipleErrorMessages", true);
    }
    catch (e) {
        alert("can't load books.xml : " + e.description);
        return;
    }

    // Validate the XML document, relying on the xsi:schemaLocation
    // attribute of the document element to resolve where to read the
    // schema definition.
    var err = xd.validate();
    if (err.errorCode != 0 ) {
        var allErrs = err.allErrors;
        var str=""
        i = 0
        while (null != (e=allErrs.next)) {
            str += ("errorItem["+ i++ +"]": "+e.reason+"\n";
        }
        alert(str);
    } else
        alert("valid dom as follows:\n" + xd.xml);
}```
function alert(msg)
{
    WScript.echo(msg);
}

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as next.js, in the same directory where you saved books.xml and books.xsd.
4. Double click the next.js file from Windows Explorer to launch the application. Alternatively, you can type "next.js" from a command prompt.

    Note  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in Output for the next Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Visual Basic Code (next.frm)

Private Sub Form_Load()
  Dim er As IXMLDOMParseError2
  Dim errs As IXMLDOMParseErrorCollection
  Dim xd As New DOMDocument50
  xd.async = False
  xd.validateOnParse = False
  xd.setProperty "MultipleErrorMessages", True

  ' Load XML to xd
  xd.Load App.Path & "books.xml"

  ' Validate the XML, relying on the xsi:schemaLocation attribute
  ' of the XML document element to resolve where to read the schema
  ' definition.
  Set er = xd.Validate
  If er.errorCode <> 0 Then
    Set errs = er.allErrors
    i = 0
    msg = ""
    Set er = errs.Next
    While Not (er Is Nothing)
      msg = msg + "errorItem[" + CStr(i) + "]: " + er.reason
      msg = msg + vbNewLine
      Set er = errs.Next
      i = i + 1
    Wend
  Else
    msg = "DOM is valid:" & vbNewLine & xd.xml
  End If
  MsgBox msg
End Sub

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as next.vbp to the same directory where you saved books.xml and books.xsd.
Name the form file \texttt{next.frm}.

4. Create a reference to MSXML 5.0. To do this, select \texttt{References...} from the \texttt{Project} menu, then check the box for \texttt{Microsoft XML, v50}.

5. Copy the Visual Basic code listing above, and paste it into the \texttt{form_load} subroutine.

6. Execute the code by selecting \texttt{Start} from the \texttt{Run} menu.

7. Verify that your output is the same as that listed in \texttt{Output for the next Example}.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    HRESULT hr;
    CoInitialize(NULL);
    IXMLDOMDocument3Ptr pXMLDoc;
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XML DOM");
        return -1;
    }
    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
    if (FAILED(hr)) {
        printf("Failed to enable mulitple validation errors
");
        return -1;
    }
    if(pXMLDoc->load("books.xml")!=VARIANT_TRUE)
    {
        printf("Failed to load DOM from books.xml. %s\n",
               (LPCSTR)pXMLDoc->parseError->reason);
        return -1;
    }

    // Validate the document as a whole, relying on the
    // xsi:schemalocation attribute to resolve where to
    // to read the schema definition.
    IXMLDOMParseError2Ptr pError =pXMLDoc->validate();
    if (pError->errorCode != 0) {
        IXMLDOMParseErrorCollectionPtr pErrSet;
        pErrSet = pError->allErrors;
        IXMLDOMParseError2Ptr pErr=NULL;
        int i=0;
        while (NULL != (pErr = pErrSet->next)) {
            printf("errorItem[%d]: %s\n", i, (LPCSTR)pErr->reason);
            i++;}
    }
}
else
    printf("DOM is valid:\n%s\n",(LPCSTR)pXMLDoc->xml);

pXMLDoc.Release();
pError.Release();
CoUninitialize();

return 0;
}

Try It!

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "nextProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
4. Select FileView on the project browser, and highlight nextProj files. From the File menu, select New.
5. On the Files tab of the New dialog box, highlight C++ Source File. Then type "books.xml" in the File name text box. Click OK.
6. Copy the XML data file (books.xml), and paste it into the text file you just created.
7. Repeat steps 4-7 for the XSD listing (books.xsd).

   Note  You can also copy these files into the project's main directory using Windows Explorer (or a command prompt).
8. Repeat steps 4-7 for the C++ listing above (next.cpp).
9. Build the sample by selecting Build nextProj.exe from the Build menu.
10. Execute the sample application by selecting !Execute nextProj.exe from the Build menu.
11. Verify that your output is the same as that listed in Output for the next Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Output for the next Example

When you build and run the next example, you should get the following output.

errorItem[0]: Content of element 'review' is invalid according to the DTD/Schema.
Expecting: price.

errorItem[1]: Content of element 'review' is invalid according to the DTD/Schema.
Expecting: pub_date.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
nextSibling Property

Contains the next sibling of the node in the parent's child list.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNode.nextSibling;
```

Example

The following script example creates an IXMLDOMNode object and sets it to the next sibling of the current node.

```javascript
var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var currNode;
var nextNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    nextNode = currNode.nextSibling;
    alert(nextNode.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objXMLDOMNode = oXMLDOMNode.nextSibling
```

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode object and sets it to the next sibling of the current node.
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
Dim nextNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    Set nextNode = currNode.nextSibling
    MsgBox(nextNode.xml)
End If

[C/C++]

C/C++ Syntax

HRESULT get_nextSibling(
    IXMLDOMNode **nextSibling);

Parameters

nextSibling [out, retval]
    Right sibling of this node.

C/C++ Return Values

S_OK
    Value returned if successful.
S_FALSE
    Value returned when there is no next sibling.
E_INVALIDARG
    Value returned if the nextSibling parameter is Null.

Remarks

The property is read-only. It returns the right sibling of this node.

This value depends on the value of the nodeType property.
NODE_ATTRIBUTE

NODE_DOCUMENT

NODE_DOCUMENT_FRAGMENT

NODE_CDATA_SECTION

NODE_COMMENT

NODE_DOCUMENT_TYPE

NODE_ELEMENT

Returns the node immediately following this node in the list of children of this node's parent. If no such node exists, returns Null.

NODE_ENTITY

NODE_ENTITY_REFERENCE

NODE_NOTATION

NODE_PROCESSING_INSTRUCTION

NODE_TEXT

Always returns Null; these node types do not appear as children of any other nodes.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
nodeName Property

Returns the qualified name for attribute, document type, element, entity, or notation nodes. Returns a fixed string for all other node types. Read-only.

[Script]

Script Syntax

strValue = oXMLDOMNode.nodeName;

Example

The following script example creates an IXMLDOMNode object and displays its name.

```javascript
var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    alert(currNode.nodeName);
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMNode.nodeName

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode object and displays its name.
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    MsgBox(currNode.nodeName)
End If

[C/C++]

C/C++ Syntax

HRESULT get_nodeName(
    BSTR *name);

Parameters

name [out, retval]
    Node name, which varies depending on the node type.

C/C++ Return Values

S_OK
    Value returned if successful.
E_INVALIDARG
    Value returned if the name parameter is Null.

Remarks

String. The property is read-only and always contains a non-empty string. The
nodeName property contains the qualified name for the element, attribute, or
entity reference. For example, it returns xxx:yyy for the element <xxx:yyy>.

The node name value varies, depending on the nodeType property.

NODE_ATTRIBUTE Contains the name of the attribute.
| NODE_CDATA_SECTION | Contains the literal string "#cdata-section". |
| NODE_COMMENT | Contains the literal string "#comment". |
| NODE_DOCUMENT | Contains the literal string "#document". |
| NODE_DOCUMENT_TYPE | Contains the name of the document type; for example, xxx in `<!DOCTYPE xxx ...>`. |
| NODE_DOCUMENT_FRAGMENT | Contains the literal string "#document-fragment". |
| NODE_ELEMENT | Contains the name of the XML tag, with any namespace prefix included if present. |
| NODE_ENTITY | Contains the name of the entity. |
| NODE_ENTITY_REFERENCE | Contains the name of the entity referenced. Note that the name does not include the leading ampersand or the trailing semicolon. The name includes the namespace if one is present. |
| NODE_NOTATION | Contains the name of the notation. |
| NODE_PROCESSING_INSTRUCTION | Contains the target; the first token following the `<?` characters. |
| NODE_TEXT | Contains the literal string "#text". |

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`nodeType Property`

Applies to: [IXMLDOMAttribute](#) | [IXMLDOMCDATASection](#) | [IXMLDOMCharacterData](#) | [IXMLDOMComment](#) | [DOMDocument](#) | [IXMLDOMDocumentFragment](#) | [IXMLDOMDocumentType](#) | [IXMLDOMElement](#) | [IXMLDOMEntity](#) | [IXMLDOMEntityReference](#)
IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
**nodeValue Property**

Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes.

**Script Syntax**

`lValue = oXMLDOMNode.nodeType;`

**Example**

The following script example creates an `IXMLDOMNode` object and displays its type enumeration, in this case, 1 for `NODE_ELEMENT`.

```javascript
var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error "+myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    alert(currNode.nodeType);
}
```

**Visual Basic Syntax**

`lValue = oXMLDOMNode.nodeType`

**Example**

The following Microsoft® Visual Basic® example creates an `IXMLDOMNode` object and displays its type enumeration, in this case, 1 for `NODE_ELEMENT`.
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    MsgBox currNode.nodeType
End If

[C/C++]

C/C++ Syntax

HRESULT get_nodeType(
    DOMNodeType *type);

Parameters

type [out, retval]
    Type of the node.

C/C++ Return Values

S_OK
    Value returned if successful.
E_INVALIDARG
    Value returned if the type parameter is Null.

Remarks

Enumeration. The property is read-only. It indicates the type of the node. Use the
nodeTypeString property to return the node type in string form.

The IXMLDOMNodeType enumeration defines the following valid values that can
appear in the nodeType property.

    The node represents an element. An
NODE ELEMENT (1)

An element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection, and EntityReference. An element node can be the child of the Document, DocumentFragment, EntityReference, and Element nodes.

The node represents an attribute of an element. An attribute node can have the following child node types: Text and EntityReference. An attribute does not appear as the child node of any other node type; note that it is not considered a child node of an element.

NODE ATTRIBUTE (2)

The node represents the text content of a tag. A text node cannot have any child nodes. A text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.

NODE TEXT (3)

The node represents a CDATA section in the XML source. CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATA section node cannot have any child nodes. A CDATA section node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.

NODE CDATA SECTION (4)

The node represents a reference to an entity in the XML document. This applies to all entities, including character entity references. An entity reference node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. An entity reference

NODE ENTITY REFERENCE (5)
node can appear as the child of the Attribute, DocumentFragment, Element, and EntityReference nodes.
The node represents an expanded entity. An entity node can have child nodes that represent the expanded
entity (for example, Text and EntityReference nodes). An entity node can appear as the child of the
DocumentType node.

The node represents a processing instruction from the XML document. A processing instruction node cannot
have any child nodes. A processing instruction node can appear as the child of the Document,
DocumentFragment, Element, and EntityReference nodes.

The node represents a comment in the XML document. A comment node cannot have any child nodes. A
comment node can appear as the child of Document, DocumentFragment, Element, and EntityReference nodes.

The node represents a document object, which, as the root of the document tree, provides access to the
entire XML document. It is created using the progID "Msxml2.DOMDocument", or through a data island using <XML> or
<SCRIPT LANGUAGE=XML>. A document node can have the following child node types: Element (maximum of
one), ProcessingInstruction, Comment, and DocumentType. A document node cannot appear as the
child of any node types.
The node represents the document type declaration, indicated by the `<!DOCTYPE>` tag. A document type node can have the following child node types: Notation and Entity. A document type node can appear as the child of the Document node.

The node represents a document fragment. A document fragment node associates a node or subtree with a document without actually being contained within the document. A document fragment node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. A DocumentFragment node cannot appear as the child of any node types.

A node represents a notation in the document type declaration. A notation node cannot have any child nodes. A notation node can appear as the child of the DocumentType node.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMEElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText | XML DOM Enumerated Constants
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
nodeTypedValue Property

Contains the node value expressed in its defined data type.

[Script]

Script Syntax

```javascript
objValue = oXMLDOMNode.nodeTypedValue;
objXMLDOMNode.nodeTypedValue = objValue;
```

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
xmlDoc.documentElement = xmlDoc.createElement("Test");
xmlDoc.documentElementdataType = "bin.hex";
xmlDoc.documentElement.nodeTypedValue = "ffab123d";
alert(xmlDoc.xml);
```

Result (in a message box)

```xml
<Test xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="bin.hex">
```

[Visual Basic]

Visual Basic Syntax

```vba
objValue = oXMLDOMNode.nodeTypedValue
objXMLDOMNode.nodeTypedValue = objValue
```

Example

```vba
Dim xmlDoc As New Msxml2.FreeThreadedDOMDocument50
xmlDoc.documentElement = xmlDoc.createElement("Test")
xmlDoc.documentElement.dataType = "bin.hex"
xmlDoc.documentElement.nodeTypedValue = "ffab123d"
MsgBox xmlDoc.xml
```

Result (in a message box)
C/C++ Syntax

HRESULT get_nodeTypedValue(  
    VARIANT *typedValue);
HRESULT put_nodeTypedValue(  
    VARIANT typedValue);

Parameters

typedValue [out, retval][in]  
    Value of this node in its defined data type, if such a data type is defined. If this node has no content, the value of the typedValue parameter is Null.

C/C++ Return Values

S_OK  
    Value returned if successful.
E_INVALIDARG  
    Value returned if the typedValue parameter is Null.
E_FAIL (for put_nodeTypedValue only)  
    Value returned if an error occurs.

Remarks

Variant. The property is read/write. It returns the value of this node in its defined data type, if such a data type is defined. If the node has no content, this property returns Null.

This value depends on the value of the nodeType property of the IXMLDOMNode.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

The data type for this node is defined in its dataType property. This property also depends on the value of the node type, in the nodeType property.
Although you can explicitly set the nodeTypedValue property with a particular data type in the VARIANT, the set operation does not change the data type. The value in the VARIANT is coerced to the data type specified in the dataType property, and the set operation returns an error if the type coercion is not possible.

Changes to this property are also immediately available from the nodeValue property.

NODE_ATTRIBUTE

Returns the value in the data representation of the type specified in the schema for the attribute. (Data types are specified using the dt:dt attribute, and because attributes cannot be applied to attributes, the attribute's data type can be specified only in the schema.) If no type was specified, returns a string identical to the nodeValue property. If the new value is not of the type specified in the schema or cannot be coerced to the type specified in the schema, then an error occurs.

NODE_CDATA_SECTION

NODE_COMMENT

NODE_DOCUMENT

NODE_DOCUMENT_FRAGMENT

NODE_DOCUMENT_TYPE

NODE_ENTITY

NODE_NOTATION

NODE_PROCESSING_INSTRUCTION

Returns a string that is equal to the nodeValue property. These node types do not have data types.
Returns the value in the data representation of the type specified on the element instance (using the `dt:dt` attribute) or specified in the schema. If no type was specified on the instance or in the schema, returns a string identical to the `nodeValue` property. If the element has an attribute data type, returns null.

Returns the value in the data representation of the type specified for the referenced entity. This is limited to entities with a single data type on the root level element of the subtree corresponding to the entity. If no data type was specified, returns a string identical to the `nodeValue` property.

Returns the value in the data representation of the type of the containing element. If the containing element was not typed, returns a string identical to the `nodeValue` property.

### Data Type Mapping

<table>
<thead>
<tr>
<th>VARIANT returned</th>
<th>Corresponding element data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_BSTR</td>
<td>string</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>number</td>
</tr>
<tr>
<td>VT_I4</td>
<td>Int</td>
</tr>
<tr>
<td>VT_CY</td>
<td>Fixed.14.4</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>Boolean</td>
</tr>
<tr>
<td>VT_DATE</td>
<td>dateTime</td>
</tr>
<tr>
<td>VT_DATE</td>
<td>dateTime.tz</td>
</tr>
</tbody>
</table>
VT_DATE    Date
VT_DATE    Time
VT_DATE    Time.tz
VT_I1      i1 byte
VT_I2      i2
VT_I4      i4, int
VT_UI1     ui1
VT_UI2     ui2
VT_UI4     ui4
VT_FLOAT   r4
VT_DOUBLE  r8, float
VT_BSTR    uuid
VT_ARRAY   bin.hex
VT_ARRAY   bin.base64

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

See Also

dataType Property  |  nodeType Property  |  nodeValue Property

Applies to: IXMLDOMAttribute  |  IXMLDOMCDATASection  |  IXMLDOMCharacterData  |  IXMLDOMComment  |  DOMDocument  |  IXMLDOMDocumentFragment  |  IXMLDOMDocumentType  |  IXMLDOMElement  |  IXMLDOMEntity  |  IXMLDOMEntityReference  |  IXMLDOMNode  |  IXMLDOMNotation  |  IXMLDOMProcessingInstruction  |  IXMLDOMText
nodeTypeString Property

Returns the node type in string form.

[Script]

Script Syntax

strValue = oXMLDOMNode.nodeTypeString;

Example

The following script example creates an IXMLDOMNode object and displays its node type in string form, in this case, "element".

var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    alert(currNode.nodeTypeString);
}

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMNode.nodeTypeString

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode object and displays its node type in string form, in this case, "element".

Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    MsgBox currNode.nodeTypeString
End If

[C/C++]

**C/C++ Syntax**

HRESULT get_nodeTypeString(
    BSTR *nodeType);

**Parameters**

*nodeType* [out, retval]
String version of the node type.

**C/C++ Return Values**

S_OK
Value returned if successful.
E_INVALIDARG
Value returned if the *nodeType* parameter is Null.

**Remarks**

String. The property is read-only. It contains the string version of the node type. To return the enumeration value, use the *nodeType* property.

This value depends on the value of the *nodeType* property.

**NODE_ATTRIBUTE**
attribute

**NODE_CDATA_SECTION**
cdatasection
This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

definition

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
nodeValue Property

Contains the text associated with the node.

[Script]

Script Syntax

```javascript
objValue = oXMLDOMNode.nodeValue;
objXMLDOMNode.nodeValue = objValue;
```

Example

The following script example creates an IXMLDOMNode object and tests if it is a comment node. If it is, it displays its value.

```javascript
var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.loadXML("<root><!-- Hello --></root>");
if (xmlDoc.parseError.errorCode != 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    if (currNode.nodeTypeString == "comment") {
        alert(currNode.nodeValue);
    }
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
objValue = oXMLDOMNode.nodeValue
objXMLDOMNode.nodeValue = objValue
```

Example
The following Microsoft® Visual Basic® example creates an IXMLDOMNode object and tests if it is a comment node. If it is, it displays its value.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    If currNode.nodeTypeString = "comment" Then
        MsgBox currNode.nodeValue
    End If
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_nodeValue(
    VARIANT *value);
HRESULT put_nodeValue(
    VARIANT value);
```

**Parameters**

`value` [out, retval][in]

Node value; depends on the `nodeType` property.

**C/C++ Return Values**

S_OK

Value returned if successful.

S_FALSE (for `get_nodeValue` only)

Value when returning Null.

E_INVALIDARG (for `get_nodeValue` only)

Value returned if the `value` parameter is Null.

E_FAIL (for `get_nodeValue` only)

Value returned if an error occurs.
**Remarks**

Variant. The property is read/write.

This value depends on the value of the `nodeType` property.

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NODE_ATTRIBUTE</code></td>
<td>Contains a string representing the value of the attribute. For attributes with subnodes, this is the concatenated text of all subnodes with entities expanded. Setting this value deletes all children of the node and replaces them with a single text node containing the value written.</td>
</tr>
<tr>
<td><code>NODE_CDATA_SECTION</code></td>
<td>Contains a string representing the text stored in the CDATA section.</td>
</tr>
<tr>
<td><code>NODE_COMMENT</code></td>
<td>Contains the content of the comment, exclusive of the comment's start and end sequence.</td>
</tr>
<tr>
<td><code>NODE_DOCUMENT</code></td>
<td></td>
</tr>
<tr>
<td><code>NODE_DOCUMENT_TYPE</code></td>
<td></td>
</tr>
<tr>
<td><code>NODE_DOCUMENT_FRAGMENT</code></td>
<td></td>
</tr>
<tr>
<td><code>NODE_ELEMENT</code></td>
<td>Contains Null. Note that attempting to set the value of nodes of these types generates an error.</td>
</tr>
<tr>
<td><code>NODE_ENTITY</code></td>
<td></td>
</tr>
<tr>
<td><code>NODE_ENTITY_REFERENCE</code></td>
<td></td>
</tr>
<tr>
<td><code>NODE_NOTATION</code></td>
<td></td>
</tr>
<tr>
<td><code>NODE_PROCESSING_INSTRUCTION</code></td>
<td>Contains the content of the processing instruction, excluding the target. (The target appears in the <code>nodeName</code> property.)</td>
</tr>
<tr>
<td><code>NODE_TEXT</code></td>
<td>Contains a string representing the text stored in the text node.</td>
</tr>
</tbody>
</table>
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

nodeName Property | nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMEElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
notationName Property

Contains the notation name.

[Script]

Script Syntax

strValue = oXMLDOMEntity.notationName;

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMEntity.notationName

[C/C++]

C/C++ Syntax

HRESULT get_notationName(
    BSTR *name);

Parameters

name [out, retval]
    For unparsed entities, the name of the notation for the entity (the name following NDATA). For parsed entities, contains the empty string.

C/C++ Return Values

S_OK
    Value returned if successful.
S_FALSE
    Value returned when no notation is specified.
E_INVALIDARG
Value returned if the name parameter is Null.

Example

BOOL DOMEntityNotationName()
{
    BOOL bResult = FALSE;
    IXMLDOMEntity *pIXMLDOMEntity = NULL;
    BSTR bstrName;
    HRESULT hr;

    try
    {
        pIXMLDOMEntity = GetFirstEntity();

        if(pIXMLDOMEntity)
        {
            hr = pIXMLDOMEntity->get_notationName(&bstrName);
            ::MessageBox(NULL, bstrName, _T("Notation name"), MB_OK);
            bResult = TRUE;
            ::SysFreeString(bstrName);
            pIXMLDOMEntity->Release();
        }
    }
    catch(...) {
        if(pIXMLDOMEntity)
            pIXMLDOMEntity->Release();
        DisplayErrorToUser();
        return bResult;
    }

    IXMLDOMEntity* GetFirstEntity()
    {
        IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMapEntities = NULL;
        IXMLDOMEntity *pIXMLDOMEntity = NULL;
        IXMLDOMNode *pIXMLDOMNode = NULL;
        IXMLDOMDocument *pIXMLDOMDocument = NULL;
        HRESULT hr;

        try
        {
            // Create an instance of DOMDocument and initialize
            // pIXMLDOMDocument.
            // Load/create an XML fragment.
            hr = pIXMLDOMDocument->get_doctype(&pIXMLDOMDocumentType);
            SUCCEEDED(hr) ? 0 : throw hr;
        }
        catch(...) {
            if(pIXMLDOMDocument)
                pIXMLDOMDocument->Release();
            DisplayErrorToUser();
if(pIXMLDOMDocumentType)
{
    hr = pIXMLDOMDocumentType->get_entities
        (&pIXMLDOMNamedNodeMapEntities);
    if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMapEntities)
    {
        hr = pIXMLDOMNamedNodeMapEntities->nextNode(&pIXMLDOMNode);
        if(SUCCEEDED(hr) && pIXMLDOMNode)
        {
            hr = pIXMLDOMNode->QueryInterface(IID_IXMLDOMEntity,
                                            (void**)&pIXMLDOMEntity);
            SUCCEEDED(m_hr) ? 0 : throw m_hr;
            pIXMLDOMNode->Release();
        }
        pIXMLDOMNamedNodeMapEntities->Release();
    }
    pIXMLDOMDocumentType->Release();
}
}
catch(...)
{
    if(pIXMLDOMNode) pIXMLDOMNode->Release();
    if(m_pIXMLDOMNamedNodeMapEntities)
        m_pIXMLDOMNamedNodeMapEntities->Release();
    if(m_pIXMLDOMDocumentType)
        m_pIXMLDOMDocumentType->Release();
    DisplayErrorToUser();
    // Release pIXMLDOMEntity when finished using it.
    // Release pIXMLDOMDocument when finished using it.
    return pIXMLDOMEntity;
}

Remarks

String. The property is read-only. For unparsed entities, the notationName property contains the name of the notation for the entity (the name following NDATA). For parsed entities, it contains the empty string. Note that the name of the entity is available using the nodeName property.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
nodeName Property

Applies to: IXMLDOMEntity
notations Property

Contains a list of the IXMLDOMNodeotation members that are present in the document type declaration.

[Script]

Script Syntax

var objXMLDOMNamedNodeMap = oXMLDOMDocumentType.notations;

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNamedNodeMap = oXMLDOMDocumentType.notations

[C/C++]

C/C++ Syntax

HRESULT get_notations(
    IXMLDOMNodeNamedNodeMap **notationMap);

Parameters

notationMap [out, retval]
    List of IXMLDOMNodeotation that are present in the document type declaration.

C/C++ Return Values

S_OK
    Value returned if successful.
E_INVALIDARG
    Value returned if the notationMap parameter is Null.
Example

BOOL DOMDocTypeProps()
{
    BOOL bResult = FALSE;
    IXMLDOMDocument *pIXMLDOMDocument = NULL;
    IXMLDOMDocumentType *pIXMLDOMDocumentType = NULL;
    IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMapNotations = NULL;
    HRESULT hr;

    try
    {
        // Create an instance of DOMDocument and initialize
        // pIXMLDOMDocument.
        // Load/create an XML fragment.
        hr = pIXMLDOMDocument->get_doctype(&pIXMLDOMDocumentType);
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXMLDOMDocumentType)
        {
            hr = pIXMLDOMDocumentType->get_notations
                 (&pIXMLDOMNamedNodeMapNotations);
            if(pIXMLDOMNamedNodeMapNotations)
            {
                // DisplayMapEntries(pIXMLDOMNamedNodeMapNotations)
                bResult = TRUE;
                pIXMLDOMNamedNodeMapNotations->Release();
            }
            pIXMLDOMDocumentType->Release();
        }
    }
    catch(...)
    {
        if(pIXMLDOMDocumentType)
            pIXMLDOMDocumentType->Release();
        if(pIXMLDOMNamedNodeMapNotations)
            pIXMLDOMNamedNodeMapNotations->Release();
        DisplayErrorToUser();
    }
    return bResult;
}

Remarks

The property is read-only. Notations identify by name the format of unparsed entities, the format of elements that bear a notation attribute, or the application to which a processing instruction is addressed.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXMLDOMNotation

Applies to: IXMLDOMDocumentType
**ondaAvailable Property**

Specifies the event handler for the ondataavailable event.

**[Script]**

**Script Syntax**

```javascript
oXMLDOMDocument.ondataavailable = funcMyEventHandler;
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
objXMLDOMDocument.ondataavailable = value

Private WithEvents XmlDoc As Msxml2.DOMDocument
Private Sub XmlDoc_ondataavailable()
```

**[C/C++]**

**C/C++ Syntax**

```c
HRESULT put_ondataavailable(
    VARIANT ondataavailableSink);
```

**Parameters**

- **ondataavailableSink [in]**
  
  Name of the function that should be called when the readyState property value changes to the value INTERACTIVE, indicating that data is available.

**C/C++ Return Values**

- **S_OK**
  
  Value returned if successful.
Remarks

The property is write-only. When handling asynchronous data, it is useful to be able to start processing in parallel with the download as soon as data becomes available. The ondataavailable event fires each time a new chunk of data arrives. The readyState property defines several states that specify the current status of the asynchronous download.

You can call QueryInterface method of the DOMDocument to obtain IConnectionPointContainer, which is used in event management. The DISPID for this event is DISPID_XMLODOMEVENT_ONDATAAVAILABLE.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

ondataavailable Event | readyState Property

Applies to: DOMDocument
onreadystatechange Property (DOMDocument)

Specifies the event handler to be called when the readyState property changes.

[Script]

Script Syntax

```
oXMLDOMDocument.onreadystatechange = funcMyEventHandler;
objXMLDOMDocument.onreadystatechange = value;
```

Example

The following script example specifies the handler CheckState gets called when the readyState property changes.

```
<script>
var xmldoc;
function Load(){
    xmldoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
xmldoc.onreadystatechange = CheckState;
xmldoc.load(URL.value);
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    }
}
function CheckState(){
    var state = xmldoc.readyState;
    RESULTS.innerHTML += "readyState = " + state + "<BR>"
    if (state == 4){
        var err = xmldoc.parseError;
        if (err.errorCode != 0) {
            RESULTS.innerHTML += err.reason + "<BR>"
        else RESULTS.innerHTML += "Success" + "<BR>"
    }
}
</script>
URL: <input type=text size=60 id=URL><input type=button value=LOAD onbuttonclick=jscript:Load()>
```
In Microsoft® Visual Basic® Scripting Edition (VBScript), you can get a function pointer using the syntax `getRef("HandleStateChange").`

[Visual Basic]

**Visual Basic Syntax**

```vbnet
Private WithEvents XmlDoc As Msxml2.DOMDocument
Private Sub XmlDoc_onreadystatechange()

Example

The following Microsoft Visual Basic example specifies the handler `CheckState` gets called when the `readyState` property changes.

```vbnet
Private WithEvents XmlDoc As Msxml2.DOMDocument50
Private Sub cmdEvents_Click()
    Set XmlDoc = New Msxml2.DOMDocument50
    XmlDoc.Load "books.xml"
    If (xmlDoc.parseError.errorCode <> 0) Then
        Dim myErr
        Set myErr = xmlDoc.parseError
        MsgBox("You have error " & myErr.reason)
    End If
End Sub
Private Sub XmlDoc_onreadystatechange()
    MsgBox "onreadystatechange: " & XmlDoc.readyState
End Sub
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT put_onreadystatechange(
    VARIANT readystatechangeSink);
```

**Parameters**

*readystatechangeSink [in]*

Name of the function that should be called when the `readyState` property
value changes.

C/C++ Return Values

S_OK
Value returned if successful.

Remarks

The property is write-only. When using scripting languages, this property can be set in ways other than directly accessing the property through DOMDocument. It can also be set using the onreadystatechange attribute of the <XML> tag, and the SCRIPT FOR... construct.

You can call the QueryInterface method of the DOMDocument to obtain IConnectionPointContainer for event management. The DISPID for this event is DISPID_XMLDOMEVENT_ONREADYSTATECHANGE.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

onreadystatechange Event | readyState Property

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
onreadystatechange Property
(IXMLHttpRequest)

Specifies the event handler to be called when the readyState property changes. Note that onreadystatechange is designed for use in scripting environments and is not readily accessible in Microsoft® Visual Basic® or C++.

[Script]

Script Syntax

oXMLHttpRequest.onreadystatechange = funcMyHandler;

Example

The following script example specifies the handler HandleStateChange gets called when an IXMLHttpRequest object's readyState property changes. A button on a page is enabled when the readyState property indicates that all data has been received (readystate == 4).

```<script>
var xmlhttp=null;
function PostOrder(xmldoc)
{
   var xmlhttp = new ActiveXObject5.0("Msxml2.XMLHTTP.5.0");
   xmlhttp.Open("POST", "http://myserver/orders/processorder.asp", false);
   xmlhttp.onreadystatechange= HandleStateChange;
   xmlhttp.Send(xmldoc);
   myButton.disabled = true;
}
function HandleStateChange()
{
   if (xmlhttp.readyState == 4)
   {
      myButton.disabled = false;
      alert("Result = " + xmlhttp.responseXML.xml);
   }
}
</script>
In Microsoft Visual Basic® Scripting Edition (VBScript), you can get a function pointer using the syntax `getRef("HandleStateChange")`.

**Visual Basic Syntax**

In Visual Basic, you need to write the following statement where `xmlDoc` is a variable and implements a subroutine called `xmlDoc_onreadystatechange()`.

```vbs
Dim WithEvents xmlDoc As DOMDocument50
```

**Example**

See [Use OnReadyStateChange Property in Visual Basic and Visual C/C++]

**C/C++ Syntax**

Use connection points to trap `readystatechange` events.

The following macros declare the handler (DOMDocCheckState) for the event:

```c
BEGIN_DISPATCH_MAP(CXMLDOMSamples, CCmdTarget)
    DISP_FUNCTION_ID(CXMLDOMSamples, "Ready State Event Handler", DISPID_XMLDOMEVENT_ONREADYSTATECHANGE, DOMDocCheckState, VT_EMPTY)
END_DISPATCH_MAP()
```

```c
"m_hr = pCP->Advise(GetIDispatch(TRUE), &m_pdwCookie); "
```

The last line registers this class (CXMLDOMSamples) as an event listener with the document Connection Point Container.

**Parameters**

`pReadyStateSink [in]`

Name of the function that should be called when the `readyState` property value changes.
C/C++ Return Values

S_OK
    Value returned if successful.

Example


Remarks

The property is write-only. When using scripting languages, this property can be set in ways other than directly accessing the property through IXMLHTTPRequest. It can also be set using the onreadystatechange attribute of the &lt;XML&gt; tag, and the SCRIPT FOR... construct.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☐ in the upper-left corner of the page.

See Also

readyState Property

Applies to: IXMLHTTPRequest
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
onreadystatechange Property
(ServerXMLHTTP/IServerXMLHTTP)

Specifies the event handler to be invoked when the readyState property changes. Note that onreadystatechange is designed for use in scripting environments and is not readily accessible in Microsoft® Visual Basic® or C++.

Remarks

This event has the same syntax as the onreadystatechange event.

[Script]

Script Syntax

oServerXMLHTTPRequest.onreadystatechange = funcMyHandler;

[Visual Basic]

Visual Basic Syntax


[C/C++]

C/C++ Syntax

HRESULT put_onreadystatechange(VARIANT* readystatechangeSink);

Parameters

readystatechangeSink [in]

Name of the function that should be called when the readyState property value changes.
C/C++ Return Values

S_OK
Value returned if successful.

Example


Remarks

The property is write-only. When using scripting languages, this property can be set in ways other than directly accessing the property through IServerXMLHTTPRequest. It can also be set using the onreadystatechange attribute of the <XML> tag, and the SCRIPT FOR... construct.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

onreadystatechange Event | readyState Property

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
ontransformnode Property

Specifies the event handler for the ontransformnode event.

**Important** Sample code used for demonstration purposes in this topic assumes a side-by-side installation of another previous version of MSXML. The use of "http://www.w3.org/TR/WD-xsl" to declare the older deprecated XSL namespace causes this requirement.

[Script]

**Script Syntax**

oXMLDOMDocument.ontransformnode = funcMyEventHandler;

**Example**

In this example, the XML data island named "target" contains the XML source data, and the XML data island named "ss" contains the XSL (XSL) style sheet.

```xml
<XML id=target>
    <PEOPLE>
        <PERSON name="Atkinson, Teresa" />
        <PERSON name="Debroux, Steve" />
        <PERSON name="O'Hara, Robert" />
        <PERSON name="Porzuczek, Peter" />
    </PEOPLE>
</XML>

<XML id=ss>
    <xsl:template match="/" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:for-each select="/PEOPLE/PERSON/@name" >
            <P><xsl:value-of /></P>
        </xsl:for-each>
    </xsl:template>
</XML>
```

This script contains the Verify function, which sets the ontransformnode property to the event handler, onxform. The event handler is called before the transformation operation on each node, so given the sample data provided in the data islands, this sample calls the alert several times.
document.expando = false;
Verify();
</SCRIPT>

function onxform ( code, doc ) {
    alert("Target
document
node
being
transformed:

" + doc.xml + 
"\n\nStylesheet
node
used
for
the
transform:\n\n" + code.xml)
    if (doc.value == "O'Hara, Robert")
        return false; // stop processing
    else
        return true;
}
function Verify() {
    target.ontransformnode = onxform;
    var sOut = target.XMLDocument.transformNode(ss.XMLDocument);
    div.innerHTML += sOut;
}
</Script>

The return value from the event handler indicates whether to continue processing. In this example, the event handler checks for a particular data value and returns the value "false" to stop transformation processing.

[Visual Basic]

Visual Basic Syntax

oXMLDOMDocument.ontransformnode = funcMyEventHandler

[C/C++]

C/C++ Syntax

HRESULT put_ontransformnode(
    VARIANT ontransformnodeSink);

Parameters

ontransformnodeSink [in]
    Name of the function that should be called when a transformation is about
Remarks

The property is write-only. A transformation involves two trees: the XML source and the style sheet that is applied to it. The ontransformnode event fires before each node in the style sheet is applied to each node in the XML source.

The ontransformnode property provides a way for developers to specify an event handler to override the default handler.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

ontransformnode Event

Applies to: DOMDocument | transformNode Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
output Property

Gets a custom output to write the result of the transformation.

[Script]

Script Syntax

output = objXSLProcessor.output;
objXSLProcessor.output (output);

Parameters

output
The object to which to write the output of the transformation.

Example

var xslt = new ActiveXObject5.0("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject5.0("Msxml2.FreeThreadedDOMDocument.5.
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xmlDoc.stylesheet = xslDoc;
    var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
    xmlDoc.async = false;
    xmlDoc.load("books.xml");
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    } else {
        xslProc = xslt.createProcessor();
        xslProc.input = xmlDoc;
        xslProc.transform();
        alert(xslProc.output);
    }
}
Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
output = objXSLProcessor.output
objXSLProcessor.output (output)
```

**Parameters**

*output*

The object to which to write the output of the transformation.

**Example**

```vbnet
Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xmlDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set xslt.stylesheet = xslDoc
  xmlDoc.async = False
  xmlDoc.Load "books.xml"
  If (xmlDoc.parseError.errorCode <> 0) Then
```
Dim myErr
Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
Set xslProc = xslt.createProcessor()
xslProc.input = xmlDoc
xslProc.Transform
MsgBox xslProc.output
End If

Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>

[C/C++]

C/C++ Syntax

HRESULT get_output (VARIANT* pOutput);
HRESULT put_output (VARIANT output);

Parameters

pOutput [out, retval]
The custom output object or a BSTR containing the transformation result.
output [in]
The object to write the output of the transformation to.

C/C++ Return Values

E_NOINTERFACE
Value returned if the given output object does not support a supported interface.

E_FAIL
Value returned if the readyState property is READYSTATE_INTERACTIVE.

Remarks

The output property can be any object/interface that supports IStream, IPersistStream, DOMDocument, ASP IResponse, ADODB.Stream, or IMXWriter.

When a new transform is started, the processor will use a QueryInterface this output for IStream. When the transform is complete or reset is called, IStream is released. The only method that is used on IStream is write. The bytes written to the stream will be encoded according to the encoding attribute on the <xsl:output> element.

If you do not provide a custom output, then you will get a string when you read this property. The string contains the incrementally buffered transformation result.

Reading this property has the side effect of resetting that internal buffer so that each time you read the property you get the next chunk of output. In this case, the output is always generated in the Unicode encoding, and the encoding attribute on the <xsl:output> element is ignored.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor
ownerDocument Property

Returns the root of the document that contains the node.

[Script]

Script Syntax

var objXMLDOMDocument = oXMLDOMNode.ownerDocument;

Example

The following script example uses the ownerDocument property to return the parent DOMDocument object, and then displays that object's root element tag name.

```javascript
var xmlDoc = new ActiveXObject5.0("Msxml2.DOMDocument.5.0");
var currNode;
var owner;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0).childNodes.item(1);
    owner = currNode.ownerDocument;
    alert(owner.documentElement.tagName);
}
```

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMDocument = oXMLDOMNode.ownerDocument

Example

The following Microsoft® Visual Basic® example uses the ownerDocument
property to return the parent DOMDocument object, and then displays that object's root element tag name.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
Dim owner As DOMDocument50
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0).childNodes.Item(1)
    Set owner = currNode.ownerDocument
    MsgBox owner.documentElement.tagName
End If

[C/C++]

C/C++ Syntax

HRESULT get_ownerDocument(
    IXMLDOMDocument **DOMDocument);

Parameters

DOMDocument [out, retval]
    Address of the parent document object that represents the root of the document.

C/C++ Return Values

S_OK
    Value returned if successful.
E_INVALIDARG
    Value returned if the DOMDocument parameter is Null.

Remarks

The property is read-only. It returns the parent document that represents the root
of the document to which this node belongs.

All nodes are created in the context of a document, and the ownerDocument property is maintained until the node is added to another document. For a node removed from a document, this property indicates the document in which the node was last included.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☰ in the upper-left corner of the page.

See Also

Applies to: IXMLDOMNodeAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMEElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
ownerTemplate Property

Returns the style sheet template that was used to create IXSLProcessor.

[Script]

Script Syntax

```javascript
var objXSLTemplate = objXSLProcessor.ownerTemplate;

Example

```javascript
var xslt = new ActiveXObject5.0("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject5.0("Msxml2.FreeThreadedDOMDocument.5.
var xslt.stylesheet = xslDoc;
xslProc = xslt.createProcessor();
alert(xslProc.ownerTemplate.stylesheet.xml);
``` 

Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
   <xsl:output method="html"/>
   <xsl:param name="param1"/>
   <xsl:template match="/">
       Hello
   </xsl:template>
   <xsl:template match="/" mode="edit">
       In Edit Mode
   </xsl:template>
   <xsl:template match="/" mode="view">
       In View Mode
   </xsl:template>
</xsl:stylesheet>
```
Visual Basic Syntax

Set objXSLTemplate = objXSLProcessor.ownerTemplate

Example

Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set xslt.stylesheet = xslDoc
  Set xslProc = xslt.createProcessor()
  MsgBox xslProc.ownerTemplate.stylesheet.xml
End If

Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>

C/C++ Syntax

HRESULT get_ownerTemplate (IXSLTemplate** ppTemplate);
Parameters

`ppTemplate` [out, retval]

The returned style sheet template property.

Remarks

The processor keeps the template active, so even if the user releases the template, this property can still be used to return the original template object.

**Note**  The stylesheet property on the returned template may be different from the stylesheet property.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: [IXSLProcessor](#)
parentNode Property

Contains the parent node.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNode.parentNode;
```

Example

The following script example obtains a reference to the IXMLDOMNode object that is another node object's parent and displays its XML.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
var newNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(1).childNodes.item(0);
    newNode = currNode.parentNode;
    alert(newNode.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objXMLDOMNode = oXMLDOMNode.parentNode
```

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode object from another node object's parent and displays its XML.
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
Dim newNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(1).childNodes.Item(0)
    Set newNode = currNode.parentNode
    MsgBox(newNode.xml)
End If

[C/C++]

C/C++ Syntax

HRESULT get_parentNode(
    IXMLDOMNode **parent);

Parameters

parent [out, retval]
    The parent of the given node instance.

C/C++ Return Values

S_FALSE
    The value returned if there is no parent.
S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the parent parameter is Null.

Remarks

The property is read-only. All nodes except Document, DocumentFragment, and Attribute nodes can have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, the parent is
Null.

This value depends on the value of the `nodeType` property.

**NODE_ATTRIBUTE**

**NODE_DOCUMENT**

Returns Null; these nodes do not have parents.

**NODE_DOCUMENT_FRAGMENT**

Returns the element or entity reference containing the CDATA section.

**NODE_CDATA_SECTION**

Returns the element, entity reference, document type, or document containing the comment.

**NODE_COMMENT**

Returns the document node.

**NODE_DOCUMENT_TYPE**

Returns the parent node of the element. If the element is the root node in the tree, the parent is the document node. If the node is the document node, `parentNode` is Null.

**NODE_ELEMENT**

Returns the document type node.

**NODE_ENTITY**

Returns the element, attribute, or entity reference containing the entity reference.

**NODE_ENTITY_REFERENCE**

Returns the document type node.

**NODE_NOTATION**

Returns the document type node.

**NODE_PROCESSING_INSTRUCTION**

Returns the parent element, attribute, or entity reference.

**NODE_TEXT**

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**
**nodeType Property**

Applies to: `IXMLDOMAttribute` | `IXMLDOMCDATASection` | `IXMLDOMCharacterData` | `IXMLDOMComment` | `DOMDocument` | `IXMLDOMDocumentFragment` | `IXMLDOMDocumentType` | `IXMLDOMElem` | `IXMLDOMEntity` | `IXMLDOMEntityReference` | `IXMLDOMNode` | `IXMLDOMNotation` | `IXMLDOMProcessingInstruction` | `IXMLDOMText`
parsed Property

Indicates the parsed status of the node and child nodes.

[Script]

Script Syntax

boolValue = oXMLDOMNode.parsed;

Example

The following script example displays whether or not the top-level node (root) and all its descendants are parsed.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
xmlDoc.async = true;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    alert(root.parsed);
}

[Visual Basic]

Visual Basic Syntax

boolValue = oXMLDOMNode.parsed

Example

The following Microsoft® Visual Basic® example displays whether the top-level node (root) and all its descendants are parsed.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
xmlDoc.async = True
xmlDoc.Load("books.xml")
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    Set root = xmlDoc.documentElement
    MsgBox root.parsed
End If

[C/C++]

C/C++ Syntax

HRESULT parsed(
    VARIANT_BOOL *isParsed);

Parameters

isParsed [out, retval]
    True if this node and all descendants have been parsed; False if any descendants remain to be parsed.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the isParsed parameter is Null.

Remarks

Boolean. The property is read-only. During asynchronous access, not all of the document tree may be available. Before performing some operations, such as XSLT or pattern-matching operations, it is useful to know whether the entire tree below this node is available for processing.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

XSLT Reference

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
parseError Property

Returns an IXMLDOMParseError object that contains information about the last parsing error.

[Script]

Script Syntax

```javascript
var objError = objXMLDOMDocument.parseError;
```

Example

The following script example uses the document's parseError property to create an IXMLDOMParseError object. It then tests the error and displays a message if one occurs.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var myErr;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    myErr = xmlDoc.parseError;
    if (myErr.errorCode != 0) {
        alert("You have error " + myErr.reason);
    }
}
```

[Visual Basic]

Visual Basic Syntax

```visualbasic
Set objError = objXMLDOMDocument.parseError
```

Example
The following Microsoft® Visual Basic® example uses the document's parseError property to create an IXMLDOMParseError object. It then tests the error and displays a message if one occurs.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim myErr As IXMLDOMParseError
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set myErr = xmlDoc.parseError
    If (myErr.errorCode <> 0) Then
        MsgBox ("You have error " & myErr.reason)
    End If
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_parseError(
    IXMLDOMParseError **errorObj);
```

**Parameters**

`errorObj [out, retval]`

The DOMParseError object. The `errorObj` parameter is always a valid object.

**C/C++ Return Values**

`S_OK`

The value returned if successful.

`E_INVALIDARG`

The value returned if the `errorObj` parameter is Null.

**Example**

```c
IXMLDOMParseError *pIParseError = NULL;
```
HRESULT hr;
IXMLDOMDocument *pIXMLDOMDocument = NULL;
wstring strOutput = _T("Load Error");

    // Initialize pIXMLDOMDocument (create a DOMDocument).
    // Load document.
    try
    {
        hr = pIXMLDOMDocument->get_parseError(&pIParseError);
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIParseError )
        {
            hr = pIParseError->get_errorCode(&value);
            if(SUCCEEDED(hr))
            {
                strOutput = (value != 0) ? _T("Load Error") : _T("Load Succeeded");
            }
            pIParseError->Release();
            pIParseError = NULL;
        }
    } catch(...) {
        if(pIParseError)
        
            pIParseError->Release();
            DisplayErrorToUser();
    }
    // Release the IXMLDOMDocument interface when finished with it.

Remarks

The property is read-only. It returns IXMLDOMParseError.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMParseError
Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
prefix Property

Returns the namespace prefix.

[Script]

Script Syntax

strValue = oXMLDOMNode.prefix;

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMNode.prefix

[C/C++]

C/C++ Syntax

HRESULT get_prefix(
    BSTR *prefixString);

Parameters

prefixString [out, retval]
    The namespace prefix specified on the element, attribute, or entity reference. For example, for the element <xxx:yyy>, it returns xxx. It returns an empty string, """, if no prefix is specified.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned when there is no prefix.
E_INVALIDARG

The value returned if the prefixString parameter is Null.

Remarks

String. The property is read-only. It returns the namespace prefix specified. For example, for the element <xxx:yyy>, it returns "xxx". It returns an empty string, "", if no prefix is specified.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: [IXMLDOMAttribute] [IXMLDOMCDATASection] [IXMLDOMCharacterData] [IXMLDOMComment] [DOMDocument] [IXMLDOMDocumentFragment] [DOMDocumentType] [IXMLDOMElement] [IXMLDOMEntity] [IXMLDOMEntityReference] [IXMLDOMNode] [IXMLDOMNotation] [IXMLDOMProcessingInstruction] [IXMLDOMText]
preserveWhiteSpace Property

Specifies the default white space handling.

[Script]

Script Syntax

```javascript
boolVal = objXMLDOMDocument.preserveWhiteSpace;
objXMLDOMDocument.preserveWhiteSpace = boolVal;
```

Example

The following script example first loads and displays a file with the preserveWhiteSpace property set to True and then reloads and displays the file with the preserveWhiteSpace property set to False.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.preserveWhiteSpace = true;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(xmlDoc.xml);
    xmlDoc.async = false;
    xmlDoc.preserveWhiteSpace = false;
    xmlDoc.load("books.xml");
    alert(xmlDoc.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
boolVal = objXMLDOMDocument.preserveWhiteSpace;
objXMLDOMDocument.preserveWhiteSpace = boolVal;
```

Example
The following Microsoft® Visual Basic® example first loads and displays a file with the preserveWhiteSpace property set to True and then reloads and displays the file with the preserveWhiteSpace property set to False.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.preserveWhiteSpace = True
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  MsgBox xmlDoc.xml
  xmlDoc.async = False
  xmlDoc.preserveWhiteSpace = False
  xmlDoc.Load("books.xml")
  MsgBox xmlDoc.xml
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_preserveWhiteSpace(
    VARIANT_BOOL *isPreserving);
HRESULT put_preserveWhiteSpace(
    VARIANT_BOOL isPreserving);
```

**Parameters**

*isPreserving* [out, retval][in]

The value that indicates whether default processing preserves white space.

**C/C++ Return Values**

S_OK

The value returned if successful.

E_INVALIDARG (for get_preserveWhiteSpace only)

The value returned if the isPreserving property is Null.

**Remarks**
Boolean. The property is read/write. This property is initialized to False.

The `preserveWhiteSpace` property specifies the default white space handling. When `preserveWhiteSpace` is True, all white space is preserved, regardless of any `xml:space` attributes specified in the document type definition (DTD). It is equivalent to having an `xml:space="preserve"` attribute on every element.

When `preserveWhiteSpace` is False, the values of any `xml:space` attributes determine where white space is preserved.

The `xml` property does not preserve white space exactly as it appears in the original document. Instead, the object model replaces white space present in the original document with a single newline character in the representation returned by the `xml` property. In a similar way, the `text` property contains a representation without the leading and trailing spaces and replaces multiple intervening white space characters between words with a single space character.

The `text` and `xml` properties preserve white space when the `preserveWhiteSpace` property is set to True and `xml:space` on the XML element has the value "preserve."

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[Text Property](#) | [Xml Property](#)

Applies to: [DOMDocument](#)
**previousSibling Property**

Contains the previous sibling of the node in the parent's child list.

[Script]

**Script Syntax**

```
var objXMLDOMNode = oXMLDOMNode.previousSibling;
```

**Example**

The following script example creates an IXMDOMNode object and sets it to the previous sibling of the current node.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
var prevNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(1);
    prevNode = currNode.previousSibling;
    alert(prevNode.xml);
}
```

[Visual Basic]

**Visual Basic Syntax**

```
Set objXMLDOMNode = oXMLDOMNode.previousSibling
```

**Example**

The following Microsoft® Visual Basic® example creates an IXMDOMNode object and sets it to the previous sibling of the current node.
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
Dim prevNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(1)
    Set prevNode = currNode.previousSibling
    MsgBox prevNode.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_previousSibling(
    IXMLDOMNode **previousSibling);

Parameters

previousSibling [out, retval]
The left sibling of this node.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned when there is no previous sibling.
E_INVALIDARG
    The value returned if the previousSibling parameter is Null.

Remarks

The property is read-only.

This value depends on the value of the nodeType property.
NODE_ATTRIBUTE

NODE_DOCUMENT

NODE_DOCUMENT_FRAGMENT

NODE_CDATA_SECTION

NODE_COMMENT

NODE_DOCUMENT_TYPE

NODE_ELEMENT

NODE_ENTITY

NODE_ENTITY_REFERENCE

NODE_NOTATION

NODE_PROCESSING_INSTRUCTION

NODE_TEXT

Always returns Null; these node types do not appear as children of any other nodes.

Returns the node immediately preceding this node in its parent's child list. Returns Null if no such node exists.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeType Property

Applies to: IXLMDOMAttribute | IXLMDOMCDATASection | IXLMDOMCharacterData | IXLMDOMComment | DOMDocument | IXLMDOMDocumentFragment | IXLMDOMDocumentType | IXLMDOMElement | IXLMDOMEntity | IXLMDOMEntityReference | IXLMDOMNode | IXLMDOMNotation | IXLMDOMProcessingInstruction | IXLMDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer...
publicId Property
(IXMLDOMEntity)

Contains the public identifier associated with the entity.

[Script]

Script Syntax

strValue = oXMLDOMEntity.publicId;

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMEntity.publicId

[C/C++]

C/C++ Syntax

HRESULT get_publicId(
    BSTR *publicID);

Parameters

publicID [out, retval]
    The public identifier associated with the entity, if specified. If the public identifier is not specified, contains the empty string.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
The value returned if no public ID is specified.

E_INVALIDARG
    The value returned if the publicID parameter is Null.

Example

IXMLDOMEntity *pIXMLDOMEntity = NULL;
VARIANT varName;
HRESULT hr;

try
{
    // See help on IXMLDOMEntity::get_notationName method for definition
    // the GetFirstEntity() method.
    pIXMLDOMEntity = GetFirstEntity();

    if(pIXMLDOMEntity)
    {
        hr = pIXMLDOMEntity->get_publicId(&varName);
        if(SUCCEEDED(hr))
            ::MessageBox(NULL, _bstr_t(varName), _T("Public ID"), MB_OK
                pIXMLDOMEntity->Release();
    }
} catch(...) 
{
    if(pIXMLDOMEntity)
        pIXMLDOMEntity->Release();
    DisplayErrorToUser();
}

Remarks

String. The property is read-only. If the public identifier is not specified, the property contains the empty string.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMEntity
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
publicId Property
(IXMLDOMNotation)

Contains the public identifier for the notation.

[Script]

Script Syntax

strValue = oXMLDOMNotation.publicId;

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMNotation.publicId

[C/C++]

C/C++ Syntax

HRESULT get_publicId(
    VARIANT *publicID);

Parameters

publicID [out, retval]
    The public identifier for the notation. If the public identifier is not specified,
    this contains an empty string.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
The value returned if the `publicID` parameter is Null.

Remarks

Variant. The property is read-only. If the public identifier is not specified, the property contains the empty string.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: `IXMLDOMNotation`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
readyState Property
(DOMDocument)

Indicates the current state of the XML document.

[Script]

Script Syntax

1Value = oXMLDOMDocument.readyState;

Example

The following script example demonstrates the sequence of ready states when asynchronously loading a document.

```javascript
<script>
var xmlDoc;
function Load()
{
    xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmlDoc.onreadystatechange = CheckState;
    xmlDoc.load(URL.value);
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    }
}
function CheckState()
{
    var state = xmlDoc.readyState;
    RESULTS.innerHTML += "readyState = " + state + "<BR>
    if (state == 4)
    {
        var err = xmlDoc.parseError;
        if (err.errorCode != 0)
            RESULTS.innerHTML += err.reason + "<BR>"
        else RESULTS.innerHTML += "success" + "<BR>"
    }
}
```
Visual Basic Syntax

lValue = oXMLDOMDocument.readyState

Example

The following Microsoft® Visual Basic® example demonstrates the sequence of ready states when asynchronously loading a document.

Dim WithEvents xmldoc As Msxml2.DOMDocument50
Private Sub Command1_Click()
    Set xmldoc = New Msxml2.DOMDocument50
    xmldoc.Load "books.xml"
    If (xmlDoc.parseError.errorCode <> 0) Then
        Dim myErr
        Set myErr = xmlDoc.parseError
        MsgBox("You have error " & myErr.reason)
    End If
End Sub

Private Sub xmldoc_onreadystatechange()
    Dim state As Integer
    state = xmldoc.readyState
    Text1.Text = Text1.Text & "readyState = " & state & vbCrLf
    If state = 4 Then
        If xmlDoc.parseError.errorCode <> 0 Then
            Text1.Text = Text1.Text & xmlDoc.parseError.reason & vbCrLf
        Else
            Text1.Text = Text1.Text & "success" & vbCrLf
        End If
    End If
End Sub

[C/C++]

C/C++ Syntax

HRESULT get_readyState(
    long * value);
**Parameters**

`value [out, retval]`  
The value that indicates the instantiation and download state of the XML document object. It can be one of the following:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOADING (1)</td>
<td>The load is in progress. Reading persisted properties, but not yet parsing data. For <code>readyState</code> definitions, data should be considered equivalent to binary large object (BLOB) properties.</td>
</tr>
<tr>
<td>LOADED (2)</td>
<td>Reading of the persisted properties completed. Reading and parsing data, but the object model is not yet available.</td>
</tr>
<tr>
<td>INTERACTIVE (3)</td>
<td>Some data has been read and parsed, and the object model is now available on the partially retrieved data set. Although the object model is available during this state, it is read-only.</td>
</tr>
<tr>
<td>COMPLETED (4)</td>
<td>The document has been completely loaded, successfully or unsuccessfully.</td>
</tr>
</tbody>
</table>

**C/C++ Return Values**

- **S_OK**  
The value returned if successful.
- **E_INVALIDARG**  
The value returned if the `value` parameter is Null.

**Example**

See the example in [onreadystatechange Event](#).

**Remarks**

Long integer. The property is read-only. It returns a value that indicates the instantiation and download state of the XML document object. The value can be one of the following:
<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOADING (1)</td>
<td>The load is in progress. Reading persisted properties, but not yet parsing data. For <code>readyState</code> definitions, data should be considered equivalent to binary large object (BLOB) properties.</td>
</tr>
<tr>
<td>LOADED (2)</td>
<td>Reading of the persisted properties completed. Reading and parsing data, but the object model is not yet available. Some data has been read and parsed, and the object model is now available on the partially retrieved data set.</td>
</tr>
<tr>
<td>INTERACTIVE (3)</td>
<td>Some data has been read and parsed, and the object model is available on the partially retrieved data set. Although the object model is available during this state, it is read-only.</td>
</tr>
<tr>
<td>COMPLETED (4)</td>
<td>The document has been completely loaded, successfully or unsuccessfully.</td>
</tr>
</tbody>
</table>

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

- `onreadystatechange Event`

Applies to: `DOMDocument`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
readyState Property
(IXMLHTTPRequest)

Represents the state of the request.

Example

```javascript
var XmlHttp;
XmlHttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");

function send() {
    XmlHttp.onreadystatechange = doHttpReadyStateChange;
    XmlHttp.open("GET", "http://localhost/sample.xml", true);
    XmlHttp.send();
}

function doHttpReadyStateChange() {
    if (XmlHttp.readyState == 4) {
        alert("Done");
    }
}

[Script]

Script Syntax

lValue = oXMLHttpRequest.readyState;

[Visual Basic]

Visual Basic Syntax

lValue = oXMLHttpRequest.readyState

[C/C++]

C/C++ Syntax
HRESULT get_readyState(long* plState);

Parameters

*plState* [out, retval]
State of the request, as an I4 (4-byte integer). The following values are defined:

(0) **UNINITIALIZED**  The object has been created, but not initialized (open has not been called).

(1) **LOADING**  The object has been created, but the send method has not been called.

(2) **LOADED**  The send method has been called and the status and headers are available, but the response is not yet available.

(3) **INTERACTIVE**  Some data has been received. You can call the responseBody and responseText properties to get the current partial results.

(4) **COMPLETED**  All the data has been received, and the complete data is available in the responseBody and responseText properties.

C/C++ Return Values

**S_OK**  Value returned if successful.

Remarks

Variant. The property is read-only. It represents the state of the request as an I4 (4-byte integer). The following values are defined.

0 (UNINITIALIZED)  The object has been created, but not initialized (the open method has not been called).

(1) **LOADING**  The object has been created, but the send method has not been called.

The send method has been called and the status and
(2) LOADED  Headers are available, but the response is not yet available.

(3) INTERACTIVE  Some data has been received. You can call the responseBody and responseText properties to get the current partial results.

(4) COMPLETED  All the data has been received, and the complete data is available in the responseBody and responseText properties.

This property returns a 4-byte integer.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

open Method | responseBody Property | responseText Property | send Method | status Property | statusText Property

Applies to: IXMLHTTPRequest
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
**readyState Property (IXSLProcessor)**

Returns the current state of the processor.

[Script]

**Script Syntax**

$lReadyState = objXSLProcessor.readyState;

**Example**

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
xslt.stylesheet = xslDoc;
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslProc = xslt.createProcessor();
    alert(xslProc.readyState);
    xslProc.input = xmlDoc;
    alert(xslProc.readyState);
}
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
lReadyState = objXSLProcessor.readyState
```

**Example**

```vbnet
Dim xslt As New Msxml2.XSLTemplate50
```
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
    xmlDoc.async = False
    xmlDoc.Load "books.xml"
    Set xslProc = xslt.createProcessor()
    MsgBox xslProc.readyState
    xslProc.input = xmlDoc
    MsgBox xslProc.readyState
End If

[C/C++]

C/C++ Syntax

HRESULT readyState (long* pReadyState);

Parameters

pReadyState [out, retval]
  The state of the current transformation.

Remarks

The readyState property returns the current state of the processor as follows.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READYSTATE_UNINITIALIZED (0)</td>
<td>This is the initial state until all the required properties (in particular, the input property) are set. Calling the transform method at this point returns an error.</td>
</tr>
<tr>
<td>READYSTATE_LOADING (1)</td>
<td>Not used by IXSLProcessor.</td>
</tr>
<tr>
<td>READYSTATE_LOADED (2)</td>
<td>This is the state in which all required properties are set. You can now call the</td>
</tr>
</tbody>
</table>
transform method.
The transform method has been called, but is not finished yet. Not used by IXSLProcessor.
The transformation is finished, but there may still be some output to read if you are reading the output as a string.

When you call the reset method or set any property, it reverts to READYSTATE_LOADED. If a required property is cleared, it reverts to READYSTATE_UNINITIALIZED.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**readyState Property**
*(ServerXMLHTTP/IServerXMLHTTP)*

Represents the state of the request.

**[Script]**

**Script Syntax**

```javascript
lValue = oServerXMLHTTPRequest.readyState;
```

**Example**

```javascript
var XmlServerHttp;
XmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
XmlServerHttp.onreadystatechange = doServerHttpReadyStateChange;
XmlServerHttp.open("GET", "http://localhost/sample.xml", true);
XmlServerHttp.send();
function doServerHttpReadyStateChange()
{
    if (XmlServerHttp.readyState == 4)
    {
        alert("Done");
    }
}
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
lValue = oServerXMLHTTPRequest.readyState
```

**[C/C++]**

**C/C++ Syntax**

```c
HRESULT get_readyState(long* pvarVal);
```

**Parameters**
pvarVal [out, retval]
The state of the request, as an I4 (4-byte integer). The following values are defined.

(0) UNINITIALIZED The object has been created but has not been initialized because the open method has not been called.

(1) LOADING The object has been created but the send method has not been called.

(2) LOADED The send method has been called and the status and headers are available, but the response is not yet available.
Some data has been received. You can call responseBody and responseText to get the current partial results.

(3) INTERACTIVE All the data has been received, and the complete data is available in responseBody and responseText.

C/C++ Return Values

S_OK The value returned if successful.

Remarks

This property returns a 4-byte integer.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

open Method | responseBody Property | responseText Property | send Method | status Property | statusText Property

Applies to: IXMLHTTPRequest/ServerXMLHTTP
reason Property

Describes the reason for the error.

[Script]

Script Syntax

strValue = oXMLDOMParseError.reason;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it uses the ParseError object's reason property to display the reason the error occurred.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert(xmlDoc.parseError.reason);
} else {
    alert(xmlDoc.documentElement.xml);
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMParseError.reason

Example

The following Microsoft® Visual Basic® example attempts to load an XML document. If it encounters a parse error, it uses the ParseError object's reason property to display the reason the error occurred.

```vbasic
Dim xmlDoc As New Msxml2.DOMDocument50
```
xmlDoc.async = False
xmlDoc.Load("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
    MsgBox xmlDoc.parseError.reason
Else
    MsgBox xmlDoc.documentElement.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_reason(
    BSTR *reasonString);

Parameters

reasonString [out, retval]
The reason for the error. This displays a description of the error. A validation error also includes the URL of the schema and the node within the schema that corresponds to the error.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value returned if no error occurred.

Remarks

String. The property is read-only. It displays a description of the error. A validation error also includes the URL of the schema and the node within the schema that corresponds to the error.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
Applies to: **IXMLDOMParseError**
resolveExternals Property

Indicates whether external definitions, resolvable namespaces, document type definition (DTD) external subsets, and external entity references, are to be resolved at parse time, independent of validation.

[Script]

Script Syntax

boolValue = oXMLDOMDocument.resolveExternals;
objXMLDOMDocument.resolveExternals = boolVal;

Example

The following script example sets the resolveExternals property to false before loading a document.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.resolveExternals = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
}
```

[Visual Basic]

Visual Basic Syntax

boolValue = oXMLDOMDocument.resolveExternals
objXMLDOMDocument.resolveExternals = boolVal;

Example

The following Microsoft® Visual Basic® example sets the resolveExternals property to false before loading a document.
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.resolveExternals = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
End If

[C/C++]

C/C++ Syntax

HRESULT get_resolveExternals(
    VARIANT_BOOL *isResolving);
HRESULT put_resolveExternals(
    VARIANT_BOOL isResolving);

Parameters

isResolving [out, retval][in]
    True if external definitions are to be resolved at parse time; False otherwise.
    The default is True.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG (for get_resolveExternals only)
    The value returned if the isResolving parameter is Null.

Remarks

Boolean. The property is read/write.

When the isResolving parameter is True, external definitions are resolved at parse time. This allows default attributes and data types to be defined on elements from the schema and allows use of the DTD as a file inclusion mechanism.
This setting is independent of whether validation is to be performed, as indicated by the value of the `validateOnParse` property. If externals cannot be resolved during validation, a validation error occurs. When the value of `isResolving` is False, externals are not resolved and validation is not performed.

A resolvable namespace is indicated by a namespace Universal Resource Identifier (URI) that begins with "x-schema:" as its prefix.

The default setting is True.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`validateOnParse` Property

Applies to: [DOMDocument](#)
**responseBody Property**  
*(IXMLHTTPRequest)*

Represents only one of several forms in which the HTTP response can be returned.

[Script]

**Script Syntax**

```javascript
strValue = oXMLHttpRequest.responseBody;
```

**Example**

```javascript
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmlhttp.open("GET", "http://localhost/sample.xml", false);
xmlhttp.send();
alert(xmlhttp.responseBody);
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
strValue = oXMLHttpRequest.responseBody
```

**Example**

```vbnet
Dim xmlhttp As New Msxml2.XMLHTTP50
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.responseBody
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_responseBody(VARIANT* pvarBody);
```
Parameters

`pvarBody [out, retval]`
The response entity body as an array of unsigned bytes.

C/C++ Return Values

S_OK
The value returned if successful.
E_PENDING
The value returned if the data is unavailable.

Remarks

Variant. The property is read-only. Represents the response entity body as an array of unsigned bytes, namely a SAFEARRAY of type VT_ARRAY | VT_UI1. This contains the raw undecoded bytes as received directly from the server. Therefore, depending on what the server sent, this may appear as binary-encoded data (UTF-8, UCS-2, UCS-4, Shift_JIS, and so on).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

responseStream Property | responseText Property | responseXML Property

Applies to: IXMLHTTPRequest
**responseBody Property**

**(ServerXMLHTTPRequest/IServerXMLHTTPRequest)**

Represents the response entity body as an array of unsigned bytes, namely a SAFEARRAY of type VT_ARRAY | VT_UI1. This contains the raw undecoded bytes as received directly from the server. Depending on what the server sent, this can appear as binary-encoded data (UTF-8, UCS-2, UCS-4, Shift_JIS, and so on).

[Script]

**Script Syntax**

```javascript
strValue = oServerXMLHTTPRequest.responseBody;
```

[Visual Basic]

**Visual Basic Syntax**

```vbscript
strValue = oServerXMLHTTPRequest.responseBody
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_responseBody(VARIANT* pvarVal);
```

**Parameters**

`pvarVal` [out, retval]

The response entity body as an array of unsigned bytes.

**C/C++ Return Values**

S_OK
The value returned if successful.

E_PENDING
The value returned if the data is unavailable.

Remarks

This property represents only one of several forms in which the HTTP response can be returned.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

responseStream Property | responseText Property | responseXML Property

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
responseStream Property (IXMLHTTPRequest)

Represents only one of several forms in which the HTTP response can be returned.

[Script]

**Script Syntax**

```javascript
strValue = oXMLHttpRequest.responseStream;
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
strValue = oXMLHttpRequest.responseStream
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_responseStream(VARIANT* pvarBody);
```

**Parameters**

- `pvarBody` [out, retval]
  
  The response entity body as an IStream.

**C/C++ Return Values**

- **S_OK**
  
  The value returned if successful.
- **E_PENDING**
  
  The value returned if the data is unavailable.
Example

HRESULT hr;
IXMLHttpRequest *pIXMLHttpRequest = NULL;
IStream *pIStream = NULL;
VARIANT varValue;

try
{
    // Create XMLHttpRequest object and initialize pIXMLHttpRequest.
    hr = pIXMLHttpRequest->get_responseStream(&varValue);
    if(SUCCEEDED(hr))
    {
        pIStream = (IStream*)varValue.punkVal;
        if(pIStream)
        {
            pIStream->Release();
            pIStream = NULL;
        }
    }
}
catch(...)
{
    if(pIStream)
        pIStream->Release();
    DisplayErrorToUser();

    // Release pIXMLHttpRequest when finished with it.
}

Remarks

Variant. The property is read-only. Represents the response entity body as an IStream. This stream returns the raw undecoded bytes as received directly from the server. Therefore, depending on what the server sent, this may appear as binary-encoded data (UTF-8, UCS-2, UCS-4, Shift_JIS, and so on).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

responseBody Property | responseText Property | responseXML Property

Applies to: IXMLHTTPRequest
responseStream Property
(ServerXMLHTTP/IServerXMLHTTP)

Represents the response entity body as an IStream. This stream returns the raw undecoded bytes as received directly from the server. Depending on what the server sent, this can appear as binary-encoded data (UTF-8, UCS-2, UCS-4, Shift_JIS, and so on).

[Script]

Script Syntax

strValue = oXMLHttpDispatcher.responseStream;

[Visual Basic]

Visual Basic Syntax

strValue = oXMLHttpDispatcher.responseStream

[C/C++]

C/C++ Syntax

HRESULT get_responseStream(VARIANT* pvarVal);

Parameters

pvarVal [out, retval]
   The response entity body as an IStream.

C/C++ Return Values

S_OK
   The value returned if successful.
E_PENDING
   The value returned if the data is unavailable.

Remarks

This property represents only one of several forms in which the HTTP response can be returned.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

responseBody Property | responseText Property | responseXML Property

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
**responseText Property**  
*(IXMLHTTPRequest)*

Represents the response entity body as a string.

**[Script]**

**Script Syntax**

```javascript
strValue = oXMLHttpRequest.responseText;
```

**Example**

```javascript
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmhttp.open("GET", "http://localhost/sample.xml", false);
xmhttp.send();
alert(xmlhttp.responseText);
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbscript
strValue = oXMLHttpRequest.responseText
```

**Example**

```vbscript
Dim xmlhttp As New Msxml2.XMLHTTP50
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.responseText
```

**[C/C++]**

**C/C++ Syntax**

```c
HRESULT get_responseText(BSTR* pbstrBody);
```
Parameters

`pbstrBody` [out, retval]

The response entity body as a string.

C/C++ Return Values

S_OK

The value returned if successful.

E_PENDING

The value returned if the data is unavailable.

Remarks

Variant. The property is read-only. This property represents only one of several forms in which the HTTP response can be returned.

`IXMLHTTP` attempts to decode the response into a Unicode string. It assumes the default encoding is UTF-8, but can decode any type of UCS-2 (big or little endian) or UCS-4 encoding as long as the server sends the appropriate Unicode byte-order mark. It does not process the `<? XML` coding declaration. If you know the response is going to be XML, use the `responseXML` property for full XML encoding support.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

`responseBody Property | responseStream Property | responseXML Property`

Applies to: `IXMLHTTPRequest`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**responseText Property**

**(ServerXMLHTTP/IServerXMLHTTP)**

Represents the response entity body as a string.

[Script]

**Script Syntax**

```javascript
strValue = oServerXMLHTTPRequest.responseText;
```

**Example**

```javascript
var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.open("GET", "http://localhost/sample.xml", false);
xmlServerHttp.send();
alert(xmlServerHttp.responseText);
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
strValue = oServerXMLHTTPRequest.responseText
```

**Example**

```vbnet
Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.responseText
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_responseText(VARIANT* pvarVal);
```
Parameters

pvarVal [out, retval]
   The response entity body as a string.

C/C++ Return Values

S_OK
   The value returned if successful.
E_PENDING
   The value returned if the data is unavailable.

Remarks

ServerXMLHTTP/IServerXMLHTTPRequest tries to decode the response into a Unicode string. It assumes the default encoding is UTF-8, but it can decode any type of UCS-2 (big or little endian) or UCS-4 encoding as long as the server sends the appropriate Unicode byte-order mark. It does not process the <? XML coding declaration. If you know the response is going to be XML, use the responseXML property for full XML encoding support.

This property represents only one of several forms in which the HTTP response can be returned.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

responseBody Property | responseStream Property | responseXML Property

Applies to: ServerXMLHTTP/IServerXMLHTTPRequest
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
responseXML Property (IXMLHTTPRequest)

Represents the parsed response entity body.

[Script]

Script Syntax

```javascript
var objDispatch = oXMLHttpRequest.responseXML;
```

Example

```javascript
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmlhttp.open("GET", "http://localhost/sample.xml", false);
xmlhttp.send();
alert(xmlhttp.responseXML.xml);
```

[Visual Basic]

Visual Basic Syntax

```vbscript
Set objDispatch = oXMLHttpRequest.responseXML
```

Example

```vbscript
Dim xmlhttp As New Msxml2.XMLHTTP50
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.responseXML.xml
```

[C/C++]

C/C++ Syntax

```c
HRESULT get_responseXML(IDispatch** ppBody);
```
Parameters

ppBody [out, retval]
   The response entity body as an XML document.

C/C++ Return Values

S_OK
   The value returned if successful.
E_PENDING
   The value returned if the data is unavailable.

Remarks

The property is read-only. For security reasons, the parser validation features are always turned off to prevent Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office from attempting to download a document type definition (DTD) or XML-Data definition. If the response entity body is not valid XML, this property returns DOMDocument that was parsed so that you can access the error. This property does not return IXMLDOMParseError itself, but it is accessible from DOMDocument.

If the response was generated by an Active Server Pages (ASP) page and the Multipurpose Internet Mail Extension (MIME) type was not correctly set to "text/xml" using the ASP method Response.ContentType, responseXML will be empty.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

DOMDocument | IXMLDOMParseError | responseBody Property | responseStream Property | responseText Property

Applies to: IXMLHTTPRequest
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
responseXML Property
(ServerXMLHTTP/IServerXMLHTTP)

Represents the response entity body as parsed by Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office.

[Script]

Script Syntax

var objDispatch = oServerXMLHttpRequest.responseXML;

Example

var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmServerHttp.open("GET", "http://localhost/sample.xml", false);
xmServerHttp.send();
alert(xmlServerHttp.responseXML.xml);

[Visual Basic]

Visual Basic Syntax

Set objDispatch = oServerXMLHttpRequest.responseXML

Example

Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.responseXML.xml

[C/C++]

C/C++ Syntax

HRESULT get_responseXML(IDispatch** ppXmlDom);
Parameters

*ppXmlDom* [out, retval]
The response entity body as an XML document.

**C/C++ Return Values**

**S_OK**
The value returned if successful.

**E_PENDING**
The value returned if the data is unavailable.

**Remarks**

For security reasons, the parser validation features are always turned off to prevent MSXML from attempting to download a document type definition (DTD) or XML-Data definition.

If the response entity body is not valid XML, this method returns the `DOMDocument` object that was parsed so that you can access the error object. This property does not return the error object `IXMLDOMParseError` itself. That object is accessible from the `DOMDocument` object.

If the response was generated by an Active Server Pages (ASP) page and the Multipurpose Internet Mail Extension (MIME) type was not correctly set to "text/xml" or "application/xml" using the ASP method `Response.ContentType`, `responseXML` will be empty.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`DOMDocument` | `IXMLDOMParseError` | `responseBody Property` | `responseStream Property` | `responseText Property`

 Applies to: `IServerXMLHTTPRequest/ServerXMLHTTP`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
schemas Property

Finds schema documents during load.

[Script]

Script Syntax

```javascript
var objXMLDOMSchemaCollection = objIXMLDOMDocument2.schemas;
objXMLDOMDocument.schemas = objXMLDOMSchemaCollection;
```

Example

The following script example attaches a schema to an XML document.

```javascript
var xmldoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var SchemaCache = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");

xmldoc.async = false;
xmldoc.validateOnParse = false;
SchemaCache.add("x-schema:books", "c:\books.xsd");
xmldoc.schemas = SchemaCache;
// The document will load only if a valid schema is attached to the
// file.
xmldoc.load("c:\books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(xmldoc.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objXMLDOMSchemaCollection = objIXMLDOMDocument2.schemas
objXMLDOMDocument.schemas = objXMLDOMSchemaCollection
```

[C/C++]
C/C++ Syntax

HRESULT get_schemas (VARIANT* otherCollection);
HRESULT putref_schemas (VARIANT otherCollection);

Parameters

otherCollection [out, retval][in]
The schema collection that is returned. This is the same object that was previously set, or Null if none has been set.

C/C++ Return Values

S_OK
The value returned if the method executes successfully and a schema collection is set.
S_FALSE (for get_schemas only)
The value returned if no schema collection is set.
E_POINTER (for get_schemas only)
The value returned if pSchemaCollection is Null.
E_FAIL (for putref_schemas only)
The value returned if an IXMLSchemaCollection interface cannot be obtained from SchemaCollection with formatted IErrorInfo.

Null if no schema collection is currently set. You will always get the same collection object you enter.

Example

#include "stdio.h"

#import <msxml5.dll>
using namespace MSXML2;

int checkParseError(IXMLDOMParseErrorPtr pError);
void dump_com_error(_com_error &e);

int main(int argc, char* argv[])
{
    CoInitialize(NULL);
HRESULT hr;

try{

    IXmLDOMParseErrorPtr pError;

    //load the XSD file
    IXmLDOMDocumentPtr pXSDDoc;
    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    pXSDDoc->async = VARIANT_FALSE;

    hr = pXSDDoc->load("c:\temp\bookschema.xdr");

    //check on the parser error
    if(hr!=VARIANT_TRUE)
    {
        return checkParseError(pXSDDoc->parseError);
    }

    //create schemachache
    IXmLDOMSchemaCollectionPtr pSchemaCache;
    hr = pSchemaCache.CreateInstance(__uuidof(XMLSchemaCache50));

    //add schema to schem cache
    hr = pSchemaCache->add("x-schema:books", pXSDDoc.GetInterfacePtr());

    // load the XML file
    // ******** you need to use IXmLDOMDocument2 interface ********
    IXmLDOMDocument2Ptr pXMLDoc;
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    pXMLDoc->async = VARIANT_FALSE;

    //associate xml doc with schemacache
    pXMLDoc->schemas = pSchemaCache.GetInterfacePtr();

    // relative path works in debugger, modify it with absolute pa
    hr = pXMLDoc->load("c:\temp\booksample.xml");

    //check on the parser error
    if(hr!=VARIANT_TRUE)
    {
        return checkParseError(pXMLDoc->parseError);
    }

    //call validate
    pError = pXMLDoc->validate();
if(pError->errorCode != S_OK) {
    _bstr_t parseError = _bstr_t("Error code:") + _bstr_t(pError->errorCode) + _bstr_t("Reason:") + pError->Getreason();
    MessageBox(NULL, (char*)parseError, "Parse Error", MB_OK);
    return -1;
} else
    MessageBox(NULL,"Validation succeeded", "Results",MB_OK);
}
catch(_com_error &e) {
    dump_com_error(e);
} return 0;
}

int checkParseError(IXMLDOMParseErrorPtr pError) {
    _bstr_t parseError = _bstr_t("At line ") + _bstr_t(pError->Getline()) + _bstr_t("Reason:") + pError->Getreason();
    MessageBox(NULL,parseError, "Parse Error", MB_OK);
    return -1;
}

void dump_com_error(_com_error &e) {
    printf("Error\n");
    printf("\a\tCode = %08lx\n", e.Error());
    printf("\a\tCode meaning = %s", e.ErrorMessage());
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
    printf("\a\tSource = %s\n", (LPCSTR)bstrSource);
    printf("\a\tDescription = %s\n", (LPCSTR)bstrDescription);
}

File Name: c:\temp\booksample.xml

<![if !compatible]><?xml version='1.0'?><![endif]>
<COLLECTION xmlns="x-schema:books"
           xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <DATE dt:dt="datetime">1998-10-13T15:56:00</DATE>
    <BOOK>
        <TITLE>Lover Birds</TITLE>
        <AUTHOR>Cynthia Randall</AUTHOR>
        <PUBLISHER>Lucerne Publishing</PUBLISHER>
    </BOOK>
    <BOOK>
        <TITLE>The Sundered Grail</TITLE>
    </BOOK>
File Name: c:\temp\bookschema.xdr

<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="TITLE" />
  <ElementType name="AUTHOR" />
  <ElementType name="PUBLISHER" />
  <ElementType name="DATE" />
  <ElementType name="BOOK" model="closed">
    <element type="TITLE" />
    <element type="AUTHOR" />
    <element type="PUBLISHER" />
  </ElementType>
  <ElementType name="COLLECTION" model="closed">
    <element type="BOOK" />
  </ElementType>
</Schema>

Remarks

The schemas property provides a way to associate preloaded schemas with any namespace. Set the schemas property before loading the document to identify which schemas are used by the document. The schemas property provides a way to override the existing schemas that are used by the document. Setting a new schema collection has no effect on the current document until the next load, loadXML, or validate call. The schemas loaded by the document during load are not added automatically to this collection.

Setting any non-Null schema collection automatically disables document type definition (DTD) processing because you cannot use both DTD and XML schema processing on the same document. This means DTDs will be ignored. Setting the schemas collection to Null reenables DTD processing.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button ☑ in the upper-left corner of the page.

See Also

IXMLDOMSchemaCollection/XMLSchemaCache | load Method | loadXML Method | validate Method

Applies to: IXMLDOMDocument2
specified Property

Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the document type definition (DTD) or schema.

[Script]

Script Syntax

boolValue = oXMLDOMNode.specified;

Example

The following script example creates an IXMLDOMNode from the specified item in an IXMLDOMNodeNamedNodeMap. It then displays whether or not the attribute was specified in the element, rather than in a DTD or schema.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
var objNamedNodeMap;
var myNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error "+ myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    objNamedNodeMap = currNode.attributes;
    myNode = objNamedNodeMap.item(0);
    alert(myNode.specified);
}
```

[Visual Basic]

Visual Basic Syntax

boolValue = oXMLDOMNode.specified
Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode from the specified item in an IXMLDOMNodeNamedNodeMap. It then displays whether or not the attribute was specified in the element, rather than in a DTD or schema.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
Dim objNamedNodeMap As IXMLDOMNamedNodeMap
Dim myNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error ", myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    Set objNamedNodeMap = currNode.Attributes
    Set myNode = objNamedNodeMap.Item(0)
    MsgBox myNode.specified
End If
```

[C/C++]

C/C++ Syntax

```c
HRESULT get_specified(
    VARIANT_BOOL *isSpecified);
```

Parameters

`isSpecified [out, retval]`

True if the attribute is explicitly specified in the element; False if the attribute value comes from the DTD or schema.

C/C++ Return Values

S_OK

The value returned if successful.

E_INVALIDARG

The value returned if the `isSpecified` parameter is Null.
Remarks

Boolean. The property is read-only. Returns True if the attribute is explicitly specified in the element. Returns False if the attribute value comes from the DTD or schema.

This value depends on the value of the `nodeType` property.

- `NODE_ATTRIBUTE` Returns True if the attribute was specified directly on the element. Returns False for default attributes.
- `NODE_CDATA_SECTION`
- `NODE_COMMENT`
- `NODE_DOCUMENT`
- `NODE_DOCUMENT_FRAGMENT`
- `NODE_DOCUMENT_TYPE`
- `NODE_ELEMENT` Returns True.
- `NODE_ENTITY`
- `NODE_ENTITY_REFERENCE`
- `NODE_NOTATION`
- `NODE_PROCESSING_INSTRUCTION`
- `NODE_TEXT`

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

**nodeType Property**

Applies to: IXMLDOMNode | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMEElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMNode
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
srcText Property

Returns the full text of the line containing the error.

[Script]

Script Syntax

strValue = oXMLDOMParseError.srcText;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it displays the text of the line containing the error.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode != 0) {
    alert(xmlDoc.parseError.srcText);
} else {
    alert(xmlDoc.documentElement.xml);
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMParseError.srcText

Example

The following Microsoft® Visual Basic® example attempts to load an XML document. If it encounters a parse error, it displays the text of the line containing the error.

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
```
xmlDoc.Load ("books.xml")
If xmlDoc.parseError.errorCode <> 0 Then
    MsgBox xmlDoc.parseError.srcText
Else
    MsgBox xmlDoc.documentElement.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_srcText(
    BSTR *sourceString);

Parameters

sourceString [out, retval]
    The full text of the line containing the error. This returns an empty string if
    the error is caused by XML that is not well-formed and cannot be assigned
to a specific line.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned if no error occurred.

Remarks

String. The property is read-only. It returns an empty string if an error is caused
by XML that is not well-formed and cannot be assigned to a specific line.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button [ ] in the upper-left corner of the page.

See Also

Applies to: IXMLDOMParseError
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
startMode Property

Returns the base name part of the start mode, a qualified name.

[Script]

Script Syntax

objXSLProcessor.startMode(mode);

Example

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
xslt.stylesheet = xslDoc;
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslProc = xslt.createProcessor();
xslProc.input = xmlDoc;
xslProc.setStartMode("view");
    alert(xslProc.startMode);
}
```

File Name: Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        Hello
    </xsl:template>
    <xsl:template match="/" mode="edit">
        In Edit Mode
    </xsl:template>
</xsl:stylesheet>
```
In View Mode
</xsl:template>
</xsl:stylesheet>

[Visual Basic]

**Visual Basic Syntax**

```vbscript
objXSLProcessor.startMode(mode)
```

**Example**

```vbscript
Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
Set xslt.stylesheet = xslDoc
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslProc = xslt.createProcessor()
xslProc.input = xmlDoc
xslProc.setStartMode "view"
    MsgBox xslProc.StartMode
End If
```

**File Name: Sample2.xsl**

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        Hello
    </xsl:template>
    <xsl:template match="/" mode="edit">
        In Edit Mode
    </xsl:template>
    <xsl:template match="/" mode="view">
        In View Mode
    </xsl:template>
</xsl:stylesheet>
```
C/C++ Syntax

HRESULT startMode(BSTR mode);

Parameters

mode [in]
The desired mode as a string. It must be the base name part of the qualified name.

C/C++ Return Values

E_FAIL
The value returned if readyState is READYSTATE_INTERACTIVE.
E_INVALIDARG
The value returned if the mode base name contains a colon character or is an invalid name.

Remarks

The default value of startMode is the empty string, "".

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor

Other Resources

Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen
**startModeURI Property**

Returns the namespace URI part of the start mode, the qualified name.

[Script]

**Script Syntax**

```javascript
strNamespace = objXSLProcessor.startModeURI;
```

**Example**

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
xslt.stylesheet = xslDoc;
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode !== 0) { 
    var myErr = xmlDoc.parseError;
    alert("You have error "+ myErr.reason);
} else {
    xslProc = xslt.createProcessor();
    xslProc.input = xmlDoc;
    xslProc.setStartMode("view");
    alert(xslProc.startModeURI);
}
```

**File Name: Sample2.xsl**

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        Hello
    </xsl:template>
    <xsl:template match="/" mode="edit">
        In Edit Mode
    </xsl:template>
</xsl:stylesheet>
```
[Visual Basic]

Visual Basic Syntax

strNamespace = objXSLProcessor.startModeURI

Example

Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
    xmlDoc.async = False
    xmlDoc.Load "books.xml"
    Set xslProc = xslt.createProcessor()
    xslProc.input = xmlDoc
    xslProc.setStartMode "view"
    MsgBox xslProc.startModeURI
End If

File Name: Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>
C/C++ Syntax

HRESULT get_startModeURI(BSTR * namespaceURI);

Parameters

namespaceURI [out, retval]
The namespace URI of the starting Extensible Stylesheet Language (XSL) mode.

Remarks

The default value of this property is the empty string, "".

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor

Other Resources

Worldwide Web Consortium XSL Transformations (XSLT) Version 1.0 Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referen...
status Property
(IXMLHTTPRequest)

 Represents the HTTP status code returned by a request.

[Script]

Script Syntax

lValue = oXMLHttpRequest.status;

Example

var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmllhttp.open("GET", "http://localhost/sample.xml", false);
xmllhttp.send();
alert(xmlhttp.status);

[Visual Basic]

Visual Basic Syntax

lValue = oXMLHttpRequest.status

Example

Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.status

[C/C++]

C/C++ Syntax

HRESULT get_status(long* plStatus);
**Parameters**

`plStatus` [out, retval]

The returned status code, as a long integer.

**Return Values**

The long integer status value is a standard HTTP status code as described in the following table:

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Continue</td>
</tr>
<tr>
<td>101</td>
<td>Switching Protocols</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
</tr>
<tr>
<td>203</td>
<td>Non-Authoritative Information</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
</tr>
<tr>
<td>205</td>
<td>Reset Content</td>
</tr>
<tr>
<td>206</td>
<td>Partial Content</td>
</tr>
<tr>
<td>300</td>
<td>Multiple Choices</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>302</td>
<td>Found</td>
</tr>
<tr>
<td>303</td>
<td>See Other</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
</tr>
<tr>
<td>305</td>
<td>Use Proxy</td>
</tr>
<tr>
<td>307</td>
<td>Temporary Redirect</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
</tr>
<tr>
<td>402</td>
<td>Payment Required</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
</tr>
<tr>
<td>406</td>
<td>Not Acceptable</td>
</tr>
</tbody>
</table>
407  Proxy Authentication Required
408  Request Timeout
409  Conflict
410  Gone
411  Length Required
412  Precondition Failed
413  Request Entity Too Large
414  Request-URI Too Long
415  Unsupported Media Type
416  Requested Range Not Suitable
417  Expectation Failed
500  Internal Server Error
501  Not Implemented
502  Bad Gateway
503  Service Unavailable
504  Gateway Timeout
505  HTTP Version Not Supported

**Remarks**

Long integer. The property is read-only. It represents the returned status code as a long integer. This value is valid only after the **send** method returns successfully.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[send Method](#) | [statusText Property](#)

Applies to: [IXMLHTTPRequest](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
status Property
(ServerXMLHTTP/IServerXMLHTTP)

Represents the HTTP status code returned by a request.

[Script]

**Script Syntax**

lValue = oServerXMLHTTPRequest.status;

**Example**

```javascript
var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.open("GET", "http://localhost/sample.xml", false);
xmlServerHttp.send();
alert(xmlServerHttp.status);
```

[Visual Basic]

**Visual Basic Syntax**

lValue = oServerXMLHTTPRequest.status

**Example**

```vbscript
Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.Status
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_status(long* p1Status);
```
Parameters

`plStatus [out, retval]`

The returned status code, as a long integer.

Return Values

The long integer status value is a standard HTTP status code as described in the following table:

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Continue</td>
</tr>
<tr>
<td>101</td>
<td>Switching Protocols</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
</tr>
<tr>
<td>203</td>
<td>Non-Authoritative Information</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
</tr>
<tr>
<td>205</td>
<td>Reset Content</td>
</tr>
<tr>
<td>206</td>
<td>Partial Content</td>
</tr>
<tr>
<td>300</td>
<td>Multiple Choices</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>302</td>
<td>Found</td>
</tr>
<tr>
<td>303</td>
<td>See Other</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
</tr>
<tr>
<td>305</td>
<td>Use Proxy</td>
</tr>
<tr>
<td>307</td>
<td>Temporary Redirect</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
</tr>
<tr>
<td>402</td>
<td>Payment Required</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
</tr>
<tr>
<td>406</td>
<td>Not Acceptable</td>
</tr>
</tbody>
</table>
407      Proxy Authentication Required
408      Request Timeout
409      Conflict
410      Gone
411      Length Required
412      Precondition Failed
413      Request Entity Too Large
414      Request-URI Too Long
415      Unsupported Media Type
416      Requested Range Not Suitable
417      Expectation Failed
500      Internal Server Error
501      Not Implemented
502      Bad Gateway
503      Service Unavailable
504      Gateway Timeout
505      HTTP Version Not Supported

**Remarks**

The results of this method are valid only after the `send` method has been successfully completed.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[send Method](#) | [statusText Property](#)

Applies to: [IServerXMLHTTPRequest/ServerXMLHTTP](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
statusText Property (IXMLHttpRequest)

Represents the HTTP response line status.

[Script]

Script Syntax

strValue = oXMLHttpRequest.statusText;

Example

var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmllhttp.open("GET", "http://localhost/sample.xml", false);
xmllhttp.send();
alert(xmlhttp.statusText);

[Visual Basic]

Visual Basic Syntax

strValue = oXMLHttpRequest.statusText

Example

Dim xmlhttp As New Msxml2.XMLHTTP50
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.Status

[C/C++]

C/C++ Syntax

HRESULT get_statusText(BSTR* pbstrStatus);
Parameters

pbstrStatus [out, retval]
The HTTP response as a "BSTR" value.

C/C++ Return Values

S_OK
The value returned if successful.

Remarks

String. The property is read-only. It represents the HTTP response as a BSTR value. This value is valid only after the send method returns successfully.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

send Method | status Property

Applies to: IXMLHTTPRequest
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
statusText Property (ServerXMLHTTP/IServerXMLHTTP)

Represents the HTTP response line status.

[Script]

Script Syntax

strValue = oXMLHttpRequest.statusText;

Example

var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.open("GET", "http://localhost/sample.xml", false);
xmlServerHttp.send();
alert(xmlServerHttp.statusText);

[Visual Basic]

Visual Basic Syntax

strValue = oXMLHttpRequest.statusText

Example

Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.statusText

[C/C++]

C/C++ Syntax

HRESULT get_statusText(BSTR* pbstrStatus);
Parameters

pbstrStatus [out, retval]
   The HTTP response as a "BSTR" value.

C/C++ Return Values

S_OK
   The value returned if successful.

Remarks

The results of this method are valid only after the send method has been successfully completed.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

send method | status Property

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
stylesheet Property (IXSLProcessor)

Provides the Extensible Stylesheet Language (XSL) style sheet that is to be compiled into the XSL template.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = objXSLProcessor.stylesheet;

Example

var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  xslt.stylesheet = xslDoc;
  xslProc = xslt.createProcessor();
  alert(xslProc.stylesheet.xml);
}

File Name: Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>
[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNode = objXSLProcessor.stylesheet

**Example**

Dim xslt As New Msxml2.XSLTemplate50  
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50  
Dim xslProc As IXSLProcessor  
xslDoc.async = False  
xslDoc.Load "sample2.xsl"  
If (xmlDoc.parseError.errorCode <> 0) Then  
    Dim myErr  
    Set myErr = xmlDoc.parseError  
    MsgBox("You have error " & myErr.reason)  
Else  
    Set xslt.stylesheet = xslDoc  
    Set xslProc = xslt.createProcessor()  
    MsgBox xslProc.stylesheet.xml  
End If

**File Name: Sample2.xsl**

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>
```

[C/C++]

**C/C++ Syntax**

HRESULT get_stylesheets (IXMLDOMNode** stylesheet);
Parameters

stylesheet [out, retval]
The node that represents an XSL style sheet. This must be a free-threaded document object.

Note In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

C/C++ Return Values

E_FAIL
The value returned if the document is not a valid style sheet or if the document is free-threaded.

Remarks

The style sheet document is made read-only while the style sheet belongs to the template or any associated processor. The read-only flag is removed when the style sheet is released from all templates and processors.

Because the processors created by this template take a snapshot of the current style sheet and always use it, they are not affected by changes to the stylesheet property. The only way to update the processor to the new style sheet is to create a new processor.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
stylesheet Property (IXSLTemplate)

Returns the Extensible Stylesheet Language (XSL) style sheet to compile into an XSL template.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = objXSLTemplate.stylesheet;
objXSLTemplate.stylesheet = objXMLDOMNode;
```

Example

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xsltProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslt.stylesheet = xslDoc;
    alert(xslt.stylesheet.xml);
}
```

File Name: Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        Hello
    </xsl:template>
    <xsl:template match="/" mode="edit">
        In Edit Mode
    </xsl:template>
    <xsl:template match="/" mode="view">
        In View Mode
    </xsl:template>
</xsl:stylesheet>
```
[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNode = objXSLTemplate.stylesheet
objXSLTemplate.stylesheet = objXMLDOMNode

**Example**

Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error ", & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
    MsgBox xslt.stylesheet.xml
End If

**File Name: Sample2.xsl**

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        Hello
    </xsl:template>
    <xsl:template match="/" mode="edit">
        In Edit Mode
    </xsl:template>
    <xsl:template match="/" mode="view">
        In View Mode
    </xsl:template>
</xsl:stylesheet>
```

[C/C++]

**C/C++ Syntax**

HRESULT get_stylesheet(IXMLDOMNode** stylesheet);
HRESULT putref_stylesheet(IXMLDOMNode* stylesheet);
Parameters

`stylesheet [out, retval][in]`

The node that represents an XSL Transformations (XSLT) style sheet. This must be a free-threaded document object.

**Note** In MSXML, "free-threaded" means `ThreadingModel='Both'`, and cross-thread marshalling is supported.

C/C++ Return Values

E_FAIL

The value returned if the document is not a valid style sheet or if the document is not free-threaded.

Remarks

The style sheet document is made read-only while the style sheet belongs to the template or any associated processor. The read-only flag is removed when the style sheet is released from all templates and processors.

Because the processors created by this template take a snapshot of the current style sheet and always use it, they are not affected by changes to this `stylesheet` property. The only way to update the processor to the new style sheet is to create a new processor.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLTemplate
systemId Property (IXMLDOMEntity)

Contains the system identifier associated with the entity.

[Script]

**Script Syntax**

strValue = oXMLDOMEntity.systemId;

[Visual Basic]

**Visual Basic Syntax**

strValue = oXMLDOMEntity.systemId

[C/C++]

**C/C++ Syntax**

HRESULT get_systemId(
    BSTR *systemID);

**Parameters**

**systemID [out, retval]**

The system identifier associated with the entity, if specified. If the system identifier is not specified, contains the empty string.

**C/C++ Return Values**

**S_OK**

The value returned if successful.

**S_FALSE**
The value returned when no system ID is specified.

**E_INVALIDARG**
The value returned if the `systemID` parameter is `NULL`.

**Example**

```c
IXMLDOMEntity *pIXMLDOMEntity = NULL;
VARIANT varName;
HRESULT hr;

try
{
    // See help on IXMLDOMEntity::get_notationName method for definition
    // the GetFirstEntity() method.
    pIXMLDOMEntity = GetFirstEntity();

    if(pIXMLDOMEntity)
        {
        hr = pIXMLDOMEntity->get_systemId(&varName);
        if(SUCCEEDED(hr))
            ::MessageBox(NULL, _bstr_t(varName), _T("Public ID"), MB_OK
        pIXMLDOMEntity->Release();
        }
}
catch(...)
{
    if(pIXMLDOMEntity)
        pIXMLDOMEntity->Release();
    DisplayErrorToUser();
}
```

**Remarks**

String. The property is read-only. If the system identifier is not specified, the property contains the empty string.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: **IXMLDOMEntity**
systemId Property  
(IXMLDOMNotation)

Contains the system identifier for the notation.

[Script]

**Script Syntax**

```javascript
strValue = oXMLDOMNotation.systemId;
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
strValue = oXMLDOMNotation.systemId
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_systemId(
    VARIANT *systemID);
```

**Parameters**

`systemID` [out, retval]

The system identifier for the notation. If the system identifier is not specified, this contains an empty string.

**C/C++ Return Values**

S_OK

The value returned if successful.

E_INVALIDARG
The value returned if the `systemID` parameter is Null.

**Remarks**

Variant. The property is read-only. If the system identifier is not specified, this contains the empty string.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: *IXMLDOMNotation*
**tagName Property**

Contains the element name.

**[Script]**

**Script Syntax**

```javascript
strValue = oXMLDOMElement.tagName;
```

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, namedNodeMap;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    alert(nodeBook.tagName);
}
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
strValue = oXMLDOMElement.tagName
```

**Example**

```vbnet
Dim xmlDoc As New Mxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim namedNodeMap As IXMLDOMNamedNodeMap
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
Dim myErr
Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    MsgBox nodeBook.tagName
End If

[C/C++]

C/C++ Syntax

HRESULT get_tagName(
    BSTR *tagName);

Parameters

tagName [out, retval]
    The string that represents the element's name.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value when returning Null.
E_INVALIDARG
    The value returned if the tagName parameter is Null.

Example

IXMLDOMElement *pIXMLDOMElement = NULL;
BSTR bstrTagName = NULL;
IXMLDOMDocument *pIXMLDOMDocument = NULL;
HRESULT hr;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocum
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;
if(pIXMLDOMElement)
{
    hr = pIXMLDOMElement->get_tagName(&bstrTagName);
    if(SUCCEEDED(hr))
    {
        ::MessageBox(NULL, bstrTagName, _T("Tag Name"), MB_OK);
    }
    ::SysFreeString(bstrTagName);
    bstrTagName = NULL;
    pIXMLDOMElement->Release();
}
}
catch(...)
{
    if(bstrTagName)
        ::SysFreeString(bstrTagName);
    if(pIXMLDOMElement)
        pIXMLDOMElement->Release();
    DisplayErrorToUser();
}

Remarks

String. The property is read-only. It contains the string that represents the element's name. For example, the tag name in the following tag is "book".

<book ISBN="1572318546">

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

See Also

Applies to: IXMLDOMElement
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
target Property

Specifies the target for the processing instruction.

[Script]

**Script Syntax**

strValue = oXMLDOMProcessingInstruction.target;

**Example**

The following script example iterates through the document's child nodes. If it finds a node of type NODE_PROCESSING_INSTRUCTION (7), it displays the node's target.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var pi;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    for (var i=0; i<xmlDoc.childNodes.length; i++) {
        if (xmlDoc.childNodes.item(i).nodeType == 7) {
            pi = xmlDoc.childNodes.item(i);
            alert(pi.target);
        }
    }
}
```

[Visual Basic]

**Visual Basic Syntax**

strValue = oXMLDOMProcessingInstruction.target

**Example**
The following Microsoft® Visual Basic® example iterates through the document's child nodes. If it finds a node of type NODE_PROCESSING_INSTRUCTION (7), it displays the node's target.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim pi As IXMLDOMProcessingInstruction
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    For i = 0 To (xmlDoc.childNodes.length - 1)
        If xmlDoc.childNodes.Item(i).nodeType = _
            NODE_PROCESSING_INSTRUCTION Then
            Set pi = xmlDoc.childNodes.Item(i)
            MsgBox pi.Target
        End If
    Next

[C/C++]

C/C++ Syntax

HRESULT get_target(
    BSTR *name);

Parameters

name [out, retval]
    The application to which this processing instruction is directed (the target).

C/C++ Return Values

S_OK
    The value returned if successful.

Remarks

String. The property is read-only. XML defines the target as the first token following the markup that begins the processing instruction. For example, the target has the value "xml" in the processing instruction <?xml version="1.0">.
The target property has the same value as the nodeName property.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

dnodeName Property

Applies to: IXMLDOMProcessingInstruction
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
text Property

Represents the text content of the node or the concatenated text representing the node and its descendants.

[Script]

Script Syntax

strValue = oXMLDOMNode.text;

Example

The following script example creates an IXMLDOMNode of type NODE_ENTITY, and then displays the object's text value, including that of any of the object's child nodes.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    alert(currNode.text);
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMNode.text

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode of
type NODE_ENTITY, and then displays the object's text value, including that of any of the object's child nodes.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    MsgBox currNode.Text
End If

[C/C++]

C/C++ Syntax

HRESULT get_text(
    BSTR *text);
HRESULT put_text(
    BSTR text);

Parameters

text [out, retval]
    A string representing the text content of this node and its descendants. This value varies depending on the value of the nodeType method.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned when there is no text.
E_INVALIDARG
    The value returned if the text parameter is Null.

Remarks
String. The property is read/write. When concatenated, the text represents the contents of text or CDATA nodes. All concatenated text nodes are normalized according to xml:space attributes and the value of the preserveWhiteSpace switch. Concatenated CDATA text is not normalized. (Child nodes that contain NODE_COMMENT and NODE_PROCESSING_INSTRUCTION nodes are not concatenated.) .text trims the whitespace on the edges of the result, and "normalizes" \r\n => \n, but otherwise just concatenates text.

Retrieves and sets the string representing the text contents of this node or the concatenated text representing this node and its descendants.

For more precise control over text manipulation in an XML document, use the lower-level nodeValue property, which returns the raw text associated with a NODE_TEXT node.

Consider the <root> element in this example:

```xml
<root att=" 123 a < ">
  <a> a a </a>
  <!-- comment b -->
  <?pi pi c ?>
  <![CDATA[ cdata d ]]> e f
</root>
```

The text property for this element returns the following concatenated text.

"a a cdata d e f"

Note that the white space within the CDATA node is preserved.

This value depends on the value of the nodeType property.

<table>
<thead>
<tr>
<th>NODE_ATTRIBUTE</th>
<th>Returns a string representing the value of the node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_DOCUMENT</td>
<td>This is the concatenated text of all subnodes with entities expanded.</td>
</tr>
<tr>
<td>NODE_ENTITY</td>
<td>Returns the text contained in the node,</td>
</tr>
<tr>
<td>NODE_CDATA_SECTION</td>
<td></td>
</tr>
<tr>
<td>NODE_COMMENT</td>
<td></td>
</tr>
</tbody>
</table>
**NODE_PROCESSING_INSTRUCTION** which is the same as the `nodeValue` property.

**NODE_TEXT**

**NODE_DOCUMENT_TYPE**

Returns the empty string ("""). These node types do not have associated text.

**NODE_NOTATION**

Returns the text comprised of the concatenation of all descendant nodes.

**NODE_DOCUMENT_FRAGMENT**

Contains a string that represents the element content. This will also include the text content from all child elements, concatenated in document order. For example, consider the following XML.

```xml
<count>
  <item>one</item>
  <item>two</item>
  <item>three</item>
  <item>four</item>
</count>
```

The text property for the `<count>` element contains the value "one two three four".

**NODE_ELEMENT**

**NODE_ENTITY_REFERENCE**

Returns the string representation of the entity reference.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[nodeType Property] | [nodeValue Property]

Applies to: [IXMLDOMAttribute] | [IXMLDOMCDATASection] | [IXMLDOMCharacterData] | [IXMLDOMComment] | [DOMDocument]
IXMLDOMDocumentFragment | IXMLDOMDocumentType  |
IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference  |
IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction  |
IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
url Property (DOMDocument)

Returns the URL for the last loaded XML document.

[Script]

Script Syntax

var objurl = objXMLDOMDocument.url;

Example

The following script example creates a DOMDocument and displays the value of its url property.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmldoc.async = false;
xmldoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(xmlDoc.url);
}

[Visual Basic]

Visual Basic Syntax

Set objurl = objXMLDOMDocument.url

Example

The following Microsoft® Visual Basic® example creates a DOMDocument and displays the value of its url property.

Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox xmlDoc.URL
End If

[C/C++]

C/C++ Syntax

HRESULT get_url(
    BSTR *urlString);

Parameters

urlString [out, retval]
    The URL from the last successful load. If the document is being built in memory, this property returns Null.

C/C++ Return Values

S_OK
    Value returned if successful.
S_FALSE
    Value returned if there is no URL.
E_INVALIDARG
    Value returned if the urlString parameter is Null.

Remarks

String. The property is read-only. It returns the URL from the last successful load. If the document is being built in memory, this property returns Null.

The url property is not updated after saving a document with the save method. To update the url property, reload the document using the load method.

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

(load Method) | (save Method)

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
url Property (IXMLDOMParseError)

Contains the URL of the XML document containing the last error.

[Script]

Script Syntax

strValue = oXMLDOMParseError.url;

Example

The following script example attempts to load an XML document. If it encounters a parse error, it displays the URL of the document containing the error.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    if (xmlDoc.parseError.errorCode != 0) {
        alert(xmlDoc.parseError.url);
    } else {
        alert(xmlDoc.documentElement.xml);
    }
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMParseError.url

Example
The following Microsoft® Visual Basic® example attempts to load an XML document. If it encounters a parse error, it displays the URL of the document containing the error.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    If xmlDoc.parseError.errorCode <> 0 Then
        MsgBox xmlDoc.parseError.URL
    Else
        MsgBox xmlDoc.documentElement.xml
    End If
End If
```

[C/C++]

C/C++ Syntax

```c
HRESULT get_url(
    BSTR *urlString);
```

Parameters

`urlString [out, retval]`

The URL of the XML file containing the error.

C/C++ Return Values

S_OK

The value returned if successful.

S_FALSE

The value returned if no error occurred.

Remarks

String. The property is read-only.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMParseError
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
validateOnLoad Property

Indicates whether the schema will be compiled and validated when it is loaded into the schema cache. The validateOnLoad Property must be set to false before any schemas that are not to be validated are loaded into the schema cache.

Example

Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")

' Load the schema.
nsTarget="http://www.example.microsoft.com/po"

' Setting validateOnLoad to true is optional; true is the default va
oSchemaCache.validateOnLoad = true
oSchemaCache.add "urn1", schema1 ' Validated on add.
oSchemaCache.add "urn2", schema2 ' Validated on add.
oSchemaCache.validateOnLoad = false
oSchemaCache.add "urn3", schema3 ' Not validated yet.
oSchemaCache.add "urn4", schema4 ' Not validated yet.
' Schema3 and schema4 are validated now. Schema1 and schema2 are not

oSchemaCache.validate

[Script]

Script Syntax

bolValidate = objXMLSchemaCache.validate;
objXMLSchemaCache.validate = boolTrueFalse;

[Visual Basic]

Visual Basic Syntax

bolValidate = objXMLSchemaCache.validate
objXMLSchemaCache.validate = boolTrueFalse

[C/C++]
C/C++ Syntax

HRESULT get_validateOnLoad(
    VARIANT_BOOL *validate);
HRESULT put_validateOnLoad(
    VARIANT_BOOL validate);

Parameters

validate [out, retval][in]
    If True, the schema is validated when added to the schema cache. If False, the schema is not validated until the validate method is called.

C/C++ Return Values

None

See Also

validate

Applies to: IXMLDOMSchemaCollection2/XMLSchemaCache
validateOnParse Property

Indicates whether the parser should validate this document.

[Script]

Script Syntax

boolValue = oXMLDOMDocument.validateOnParse;
objXMLDOMDocument.validateOnParse = boolValue;

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.validateOnParse = true;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
}

[Visual Basic]

Visual Basic Syntax

boolValue = oXMLDOMDocument.validateOnParse
objXMLDOMDocument.validateOnParse = boolValue

Example

Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.validateOnParse = True
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
End If
C/C++ Syntax

HRESULT get_validateOnParse(
    VARIANT_BOOL *isValidating);
HRESULT put_validateOnParse(
    VARIANT_BOOL isValidating);

Parameters

isValidating [out, retval][in]
    If True, validates during parsing. If False, parses only for well-formed XML.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG (for get_validateOnParse only)
    The value returned if the isValidating parameter is Null.

Remarks

Boolean. The property is read/write. If True, it validates during parsing. If False, it parses only for well-formed XML. The default is True.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
value Property

Contains the attribute value.

[Script]

Script Syntax

```
objValue = oXMLDOMAttribute.value;
objXMLDOMAttribute.value = objValue;
```

Example

The following script example gets the value of the first attribute of the document root and assigns it to the variable myVar.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var myVar;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  root = xmlDoc.documentElement;
  myVar = root.firstChild.attributes.item(0).value;
  alert(myVar);
}
```

[Visual Basic]

Visual Basic Syntax

```
objValue = oXMLDOMAttribute.value
objXMLDOMAttribute.value = objValue
```

Example

The following Microsoft® Visual Basic® example gets the value of the first
attribute of the document root and assigns it to the variable myVar.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim myVar As Variant
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    myVar = root.firstChild.Attributes.Item(0).Value
    MsgBox myVar
End If

[C/C++]

**C/C++ Syntax**

```c
HRESULT get_value(
    VARIANT *attributeValue);
HRESULT put_value(
    VARIANT *attributeValue);
```

**Parameters**

**attributeValue** [out, retval][in]
    The value of the attribute. For attributes with subnodes, the string is the
    concatenated text of all subnodes with character and general entity
    reference expanded (replaced with their values).

**C/C++ Return Values**

**S_OK**
    The value returned if successful.
**E_INVALIDARG** (for get_value only)
    The value returned if the attributeValue parameter is Null.
**E_FAIL** (for put_value only)
    The value returned if an error occurs.

**Remarks**
Variant. The property is read/write. It returns the value of the attribute. For attributes with subnodes, the string is the concatenated text of all subnodes with character and general entity references expanded (replaced with their values).

The new attribute value is added as the value of a single text node that is a child of the attribute node. The string contents are unparsed.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMAttribute
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
xml Property

Contains the XML representation of the node and all its descendants.

[Script]

Script Syntax

strValue = oXMLDOMNode.xml;

Example

The following script example creates an IXMLDOMNode of type NODE_ENTITY, and then displays the object's XML value, including that of any of the object's child nodes.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.childNodes.item(0);
    alert(currNode.xml);
}
```

[Visual Basic]

Visual Basic Syntax

strValue = oXMLDOMNode.xml

Example

The following Microsoft Visual Basic® example creates an IXMLDOMNode of type NODE_ENTITY, and then displays the object's XML value, including that of
any of the object's child nodes.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.childNodes.Item(0)
    MsgBox currNode.xml
End If

[C/C++]

C/C++ Syntax

HRESULT get_xml(
    BSTR *xmlString);

Parameters

xmlString [out, retval]
    An XML representation of this node and all its descendants.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the xmlString property is Null.

Remarks

The property is read-only.

The xml property always returns a Unicode string. That is, the xml property for DOMDocument converts the document from its original encoding to Unicode. As a result, the original encoding attribute is removed. For example, <?xml
version="1.0" encoding="UTF-8"?" appears in the xml property as follows.

<?xml version="1.0"?>

If the original encoding was not removed by the xml property, the following line returns an error indicating the parser cannot switch from Unicode to UTF-8 encoding.

xmlDoc.loadxml(xmlDoc.xml);

If you use the save method, the original encoding is preserved. For example, if you load a document in Microsoft® Windows® 1252 encoding, it will be saved in that format.

This value depends on the value of the nodeType property.

NODE_ATTRIBUTE

NODE_CDATA_SECTION

NODE_COMMENT

NODE_DOCUMENT

NODE_ELEMENT

Returns the string representation of the node and all its descendants.

NODE_ENTITY

NODE_NOTATION

NODE_PROCESSING_INSTRUCTION

NODE_TEXT

NODE_DOCUMENT_FRAGMENT

Returns the string representation of all descendant nodes of the document fragment.

NODE_DOCUMENT_TYPE

Returns the string representation of the <!DOCTYPE ...> declaration, including the internal subset if specified.
Returns the string representation of the entity reference. Although the entity reference has children, they are not returned.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

See Also

nodeType Property | save method | Persistence and the DOM

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Second-level DOM Properties

Second-level DOM properties are properties that are manipulated through the `setProperty` or `getProperty` method on the DOM. The following is a complete list of second-level DOM properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Variant Type</th>
<th>Default Value</th>
<th>MXSML Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowDocumentFunction</td>
<td>Boolean</td>
<td>True in MSXML 4.0 and earlier. False in MSXML 5.0 and later.</td>
<td>3.0 SP4, 4.0 SP2 and later**</td>
</tr>
<tr>
<td>ForcedResync</td>
<td>Boolean</td>
<td>True</td>
<td>3.0 SP3, 4.0 SP1, 5.0 and later</td>
</tr>
<tr>
<td>MaxXMLSize</td>
<td>Integer</td>
<td>0 for MSXML 4.0 and earlier. 10MB in later versions.</td>
<td>3.0 SP4, 4.0 SP2 and later***</td>
</tr>
<tr>
<td>MultipleErrorMessages</td>
<td>Boolean</td>
<td>False</td>
<td>5.0 and later</td>
</tr>
<tr>
<td>NewParser</td>
<td>Boolean</td>
<td>False</td>
<td>4.0 and later</td>
</tr>
<tr>
<td>ResolveExternals</td>
<td>Boolean</td>
<td>True</td>
<td>5.0 and later*</td>
</tr>
<tr>
<td>SelectionLanguage</td>
<td>String</td>
<td>&quot;XPath&quot;</td>
<td>3.0 and later</td>
</tr>
<tr>
<td>SelectionNamespaces</td>
<td>String</td>
<td>&quot;&quot;</td>
<td>3.0 and later</td>
</tr>
<tr>
<td>ServerHTTPRequest</td>
<td>Boolean</td>
<td>False</td>
<td>3.0 and later</td>
</tr>
<tr>
<td>UseInlineSchema</td>
<td>Boolean</td>
<td>True</td>
<td>5.0 and later</td>
</tr>
<tr>
<td>ValidateOnParse</td>
<td>Boolean</td>
<td>True</td>
<td>5.0 and later*</td>
</tr>
</tbody>
</table>

(*) The feature of this property is also available as a first-level property prior to MSXML 5.0.

(**) Implementation changes have been made in MSXML 5.0 and later. The new implementation of the AllowDocumentFunction property in MSXML 5.0 is a work in progress.
Remarks

The preceding table shows the properties that you can set using the `setProperty` method. White space is not stripped or normalized in property names or values.

To set a second-level property on a DOM object, use the `setProperty` method as shown in the following JScript snippet:

```javascript
domObj.setProperty('ValidateOnParse', false);
```

In contrast, first-level properties are exposed directly on the DOM. To set a first-level property, assign a value directly, as shown in the following JScript snippet:

```javascript
domObj.validateOnParse = false;
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**AllowDocumentFunction Property**

[This property was not implemented in the early versions of MSXML 5.0 used to produce this documentation. Its availability is subject to change in later versions as MSXML 5.0 is released or updated in the future.]

Enables (true) or disables (false) the document function in XLST. Attempts to call the disabled document function will result in an "Access Denied" error. This property is supported in MSXML 3.0 SP4, MSXML 4.0 SP2, and later.

**[JScript]**

**JScript Syntax**

```javascript
domObj.setProperty(strProp, strVal);
strVal = domObj.getProperty(strProp);
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbscript
domObj.setProperty(strProp, strVal)
strVal = domObj.getProperty(strProp)
```

**[C/C++]**

**C/C++ Syntax**

```c++
HRESULT setProperty(BSTR strProp, VARIANT strVal);
HRESULT getProperty(BSTR strProp, VARIANT* strVal);
```

**Value**

`strProp`

A BSTR string whose value is "AllowDocumentFunction".

`strVal`

A VARIANT_BOOL value of true or false. The default value is true in
MSXML 3.0 SP4 and MSXML 4.0 SP2. The default value is `false` in MSXML 5.0 and later.

**Remarks**

Because the document function is used in an XSLT style sheet, the AllowDocumentFunction property is effective only when it is set on a DOM object that represents an XSLT style sheet. When set, it applies to all the ensuing transformations initiated either by the `transformNode` or `transformNodeToObject` method or through the `XSLProcessor` object.

When the style sheet DOM object is cloned, the value of this property is not propagated to the cloned object. You need to reset this property on the clone if desired.

When you use `XSLTemplate`, use the value for the `AllowDocumentFunction` property that is set on the DOM object when the corresponding XSLT style sheet is added (for example, at compile time). Do not use the value that is set when the style sheet is applied.

When this property is set to `false`, the presence of the document function in an XSLT style sheet does not necessarily generate errors, as long as the function is not invoked in a transformation.

**Applies To**

**Component:** MSXML 3.0 SP4, MSXML 4.0 SP2 and later

**Interface:** [IXMLDOMDocument2](#)

**Method:** `setProperty` | `getProperty`
ForcedResync Property

Enables (true) or disables (false) resynchronization during loading through URLMON. This property is useful for optimizing cache utility in high-load server applications. To force resynchronization, set this property to true. Otherwise, set it to false. The default value of this property is true.

[JScript]

JScript Syntax

domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);

[Visual Basic]

Visual Basic Syntax

domObj.setProperty(strProp, vBool)  
vBool = domObj.getProperty(strProp)

[C/C++]

C/C++ Syntax

HRESULT setProperty(BSTR strProp, VARIANT vBool);  
HRESULT getProperty(BSTR strProp, VARIANT* vBool);

Value

strProp

A BSTR string whose value is "ForcedResync".

vBool

A VARIANT_BOOL value of true or false. The default value is true.

Applies To
Component: MSXML 3.0 SP3, MSXML 4.0 SP1, MSXML 5.0 and later

Interface: IXMLDOMDocument2

Method: setProperty | getProperty
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
MaxXMLSize Property

[This property was not implemented in the early versions of MSXML 5.0 used to produce this documentation. Its availability is subject to change in later versions as MSXML 5.0 is released or updated in the future.]

Limits the size of an XML document to be loaded into a DOM object. A zero (0) value means no limits on the size of an XML document. A non-zero value specifies the maximum size, in multiples of 1024 characters. For ASCII, the size unit is KiloBytes (KB). This property is supported in MSXML 3.0 SP4 and MSXML 4.0 SP2.

[JScript]

JScript Syntax

domObj.setProperty(strProp, strVal);
strVal= domObj.getProperty(strProp);

[Visual Basic]

Visual Basic Syntax

domObj.setProperty(strProp, strVal)
strVal= domObj.getProperty(strProp)

[C/C++]

C/C++ Syntax

HRESULT setProperty(BSTR strProp, VARIANT strVal);
HRESULT getProperty(BSTR strProp, VARIANT* strVal);

Value

strProp
A BSTR string whose value is "MaxXMLSize".
strVal

An integer of the VARIANT type. In MSXML 3.0 SP4 and MSXML 4.0 SP2, the default value of this property is 0, meaning that there is no limit in size. In later versions, the default value is $10\times2^{10}$ characters. For the ASCII character set, this is 10MB.

**Remarks**

When the DOM object is cloned, the value of this property is not propagated to the cloned object. You need to reset this property on the clone, if desired.

When the size of an XML document exceeds the limit set by this property, a load error will result when the document is parsed. Check the `parseError` property on the DOM for a more detailed error message. These error messages can differ depending on when the document was parsed (for example, in the `load` method on DOM as opposed to in the `document` function in an XSLT style sheet, and so on).

The maximum value of this property is 4194303 (i.e., $2^{22}-1$) characters. Setting this property to a value greater than this maximum value or to a negative value will result in error.

The size refers to the size of the loaded XML, so entities contribute to this size only once.

**Applies To**

**Component**: MSXML 3.0 SP4, MSXML 4.0 SP2 and later

**Interface**: [XMLDOMDocument2](#)

**Method**: [setProperty](#) | [getProperty](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
MultipleErrorMessages Property

Specifies whether or not a ParseError object can return multiple validation errors through the IXMLDOMParseError2 interface. To get all validation errors, set this property to true. Otherwise, only a single validation error is returned. The default value is false.

[JScript]

**JScript Syntax**

```javascript
domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);
```

[Visual Basic]

**Visual Basic Syntax**

```vbscript
domObj.setProperty(strProp, vBool)
vBool = domObj.getProperty(strProp)
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT setProperty(BSTR strProp, VARIANT vBool);
HRESULT getProperty(BSTR strProp, VARIANT* vBool);
```

**Value**

- **strProp**
  - A BSTR string whose value is "MultipleErrorMessages".
- **vBool**
  - A VARIANT_BOOL value of true/false. The default value is true.

**Remarks**
Multiple error messages result when non-fatal errors are encountered in a process (for example, validation). Non-fatal errors will not cause the process to stop immediately. If you set this property to false, the process stops on the first error encountered.

The MultipleErrorMessage property is supported by IXMLDOMDocument3, together with IXMLDOMParseError2 and IXMLDOMParseErrorCollection interfaces introduced in MSXML 5.0.

**Applies to**

**Component:** MSXML 5.0 and later

**Interface:** [IXMLDOMDocument2](#)

**Method:** [setProperty](#) | [getProperty](#)

**See Also**

[IXMLDOMParseError2](#) | [IXMLDOMDocument3](#) | [IXMLDOMParseErrorCollection](#)
NewParser Property

Specifies whether to enable (true) or disable (false) the use of the new parser, which was introduced in MSXML 4.0, to load a DOM document. Setting the NewParser property to false causes subsequent DOM documents to be loaded using the old parser. Setting this property to true causes DOM documents to be loaded using the new parser.

[JScript]

JScript Syntax

domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);

[Visual Basic]

Visual Basic Syntax

domObj.setProperty(strProp, vBool)
vBool = domObj.getProperty(strProp)

[C/C++]

C/C++ Syntax

HRESULT setProperty(BSTR strProp, VARIANT vBool);
HRESULT getProperty(BSTR strProp, VARIANT* vBool);

Value

strProp
A BSTR string whose value is "NewParser".

vBool
A VARIANT_BOOL value of true or false. The default value is false.

Remarks
The new parser is faster and more reliable than the old one, but it lacks support for asynchronous loading or DTD validation. The new parser will ignore the async property and throw an exception when validation against a DTD is requested. Therefore, the default value of this property is false.

For example, the following Visual Basic code fragment will fail because validation against a DTD is attempted while using the new parser. The dom object will not be loaded.

```vba
Set dom = CreateObject("MSXML2.DOMDocument.5.0")
dom.setProperty "NewParser", true
dom.ValidateOnParse = true
dom.loadXML "<?xml version='1.0' ?><!DOCTYPE root [<!ELEMENT root (#

However, the above code snippet will work if the ValidateOnParse property is turned off:

dom.ValidateOnParse = false

The following JScript code fragment also works:

dom.setProperty("NewParser", true);
dom.async = true;
dom.load("mytest.xml");

The second line above, however, will be ignored.

**Applies To**

**Component**: MSXML 4.0 and later

**Interface**: [IXMLDOMDocument2](#)

**Method**: [setProperty](#) | [getProperty](#)
ResolveExternals Property

[This feature was first implemented for MSXML 5.0.]

Specifies whether external definitions, resolvable namespaces, external subsets of document type definitions (DTDs), and external entity references are to be resolved at the time the document is parsed, and independent of validation.

[JScript]

JScript Syntax

domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);

[Visual Basic]

Visual Basic Syntax

domObj.setProperty(strProp, vBool)
vBool = domObj.getProperty(strProp)

[C/C++]

C/C++ Syntax

HRESULT setProperty(BSTR strProp, VARIANT vBool);
HRESULT getProperty(BSTR strProp", VARIANT* vBool);

Value

strProp
A BSTR string whose value is "ResolveExternals".

vBool
A VARIANT_BOOL value of true/false. The default value is true.

Remarks
ResolveExternals has a corresponding first-level property, resolveExternals, on an XML DOM. Both properties, as shown in the following JScript code fragments, have the same effect.

dom.setProperty("ResolveExternals", true);

and

dom.resolveExternals = true;

The second-level properties ValidateOnParse, ResolveExternals, UseInlineSchema, and NonValidatingSchemaUse all involve the validation of an XML document. Collectively, they influence the behavior of the parser in different ways. For more information see the Remarks section of the UseInlineSchema property.

Applies To

Component: MSXML 5.0 and later

Interface: IXMLDOMDocument2

Method: setProperty | getProperty

See Also

ValidateOnParse | UseInlineSchema
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
SelectionLanguage Property

Used in MSXML 3.0 to specify whether the DOM object should use XPath language ("XPath") or the old XSLPattern language (default) as the query language.

[JScript]

**JScript Syntax**

```javascript
domObj.setProperty(strProp, strVal);
strVal = domObj.getProperty(strProp);
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
domObj.setProperty(strProp, strVal)
strVal = domObj.getProperty(strProp)
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT setProperty(BSTR strProp, VARIANT strVal);
HRESULT getProperty(BSTR strProp, VARIANT* strVal);
```

**Value**

- `strProp`
  - A BSTR string whose value is "SelectionLanguage".
- `strVal`
  - A VARIANT string of "XPath".

**Remarks**

In MSXML 3.0 you can set this property to "XPath" to switch the query
language from XSLPattern to XPath. Once the property is set, there is no way to
switch back without creating an instance of a new DOM object.

MSXML 4.0 and later supports XPath as the only query language. This property
is always set to "XPath" in MSXML 4.0 and later. You can ignore this property
completely.

Example

```javascript
xmlDoc.setProperty("SelectionLanguage", "XPath"); // JScript
```
```vbscript
xmlDoc.setProperty "SelectionLanguage", "XPath"  ' VBScript
```

Applies To

**Component:** MSXML 3.0 and later

**Interface:** [IXMLDOMDocument2](https://msdn.microsoft.com/en-us/library/ms536588.aspx)

**Method:** [setProperty](https://msdn.microsoft.com/en-us/library/ms536588.aspx) | [getProperty](https://msdn.microsoft.com/en-us/library/ms536181.aspx)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
SelectionNamespaces Property

Specifies namespaces for use in XPath expressions when it is necessary to define new namespaces externally. Namespaces are defined in the XML style, as a space-separated list of namespace declaration attributes. You can use this property to set the default namespace as well.

[JScript]

**JScript Syntax**

```javascript
domObj.setProperty(strProp, strVal);
strVal = domObj.getProperty(strProp);
```

[Visual Basic]

**Visual Basic Syntax**

```visualbasic
domObj.setProperty(strProp, strVal)
strVal = domObj.getProperty(strProp)
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT setProperty(BSTR strProp, VARIANT strVal);
HRESULT getProperty(BSTR strProp, VARIANT* strVal);
```

**Value**

*strProp*

A BSTR string whose value is "SelectionNamespaces".

*strVal*

A VARIANT string containing a space-separated list of namespace declaration attributes. The default value is an empty string (""").

**Remarks**
When an XML document contains elements defined in an external namespace, you must use this property to specify that namespace in order to use DOM methods such as `selectNodes` or `selectSingleNode` to navigate the document.

**Example**

The following JScript example shows how to use the `SelectionNamespaces` property to specify namespaces in order to query elements that belong to different namespaces from an XML document.

**XML File (example.xml)**

The following example contains elements that belong to "a" (in **bold**) and "b" (in *italic*), in addition to elements that do not belong to any namespace.

```xml
<?xml version="1.0"?>
<root>
   <branch>branch</branch>
   <a:root xmlns:a="http://myserver.com">
      <a:branch>a-branch</a:branch>
      <b:branch xmlns:b="http://yourserver.com">
         b-branch
      </b:branch>
   </a:root>
</root>
```

**JScript Code**

Notice that the following code uses "na" and "nb" as the namespace aliases for the "http://myserver.com" and "http://yourserver.com" namespaces. The corresponding namespace aliases are "a" and "b" in the input document (example.xml).

```javascript
var dom = new ActiveXObject("MSXML2.DOMDocument.5.0");
dom.async = false;
dom.ValidateOnParse = false;
dom.load("example.xml");
if (dom.parseError.errorCode!=0)
{
   alert("can't load dom" + dom.parseError.reason);
   exit;
}

ns = "xmlns:na='http://myserver.com' xmlns:nb='http://yourserver.com"
alert("ns:(before setProperty())
  "+dom.getProperty("SelectionNamespaces", ns);
alert("ns:(after setProperty())
  "+dom.getProperty("SelectionNamespaces", ns));

node = dom.selectSingleNode("//root");
alert("root: 
  "+node.xml);

node = dom.selectSingleNode("//na:root");
alert("a:root: 
  "+node.xml);

node = dom.selectSingleNode("//branch");
alert("branch: 
  "+node.xml);

node = dom.selectSingleNode("//na:branch");
alert("a:branch: 
  "+node.xml);

node = dom.selectSingleNode("//nb:branch");
alert("b:branch: 
  "+node.xml);

function alert(str)
{
  WScript.echo(str+"\n");
}

Try It!

1. Copy example.xml and example.js onto your local drive.
2. Open a command prompt and navigate to the directory where example.xml and example.js are located.
3. Type the following command from the command prompt:
   
   Cscript example.js

You should get the following output.

Output

ns:(before setProperty())

ns:(after setProperty())
  

root:
  <root>
    <branch>branch</branch>
<a:root xmlns:a="http://myserver.com">
  <a:branch>a-branch</a:branch>
  <b:branch xmlns:b="http://yourserver.com">
    b-branch
  </b:branch>
</a:root>

<a:root xmlns:a="http://myserver.com">
  <a:branch>a-branch</a:branch>
  <b:branch xmlns:b="http://yourserver.com">
    b-branch
  </b:branch>
</a:root>

<branch>branch</branch>

<a:branch xmlns:a="http://myserver.com">a-branch</a:branch>

<b:branch xmlns:b="http://yourserver.com">
  b-branch
</b:branch>

Applies To

Component: MSXML 3.0 and later

Interface: IXMLDOMDocument2

Method: setProperty | getProperty
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
**ServerHTTPRequest Property**

Specifies whether to enable (true) or disable (false) the use of the ServerHTTPRequest object in a server application. Setting the property to false causes the DOM object not to use the HttpRequest object. Setting this property to true causes DOM documents to use ServerHTTPRequest. The default value is false.

**[JScript]**

**JScript Syntax**

```javascript
domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
domObj.setProperty(strProp, vBool)
vBool = domObj.getProperty(strProp)
```

**[C/C++]**

**C/C++ Syntax**

```c
HRESULT setProperty(BSTR strProp, VARIANT vBool);
HRESULT getProperty(BSTR strProp, VARIANT* vBool);
```

**Value**

- `strProp`  
  A BSTR string whose value is "ServerHTTPRequest".

- `vBool`  
  A VARIANT_BOOL value of true or false. The default value is false.

**Remarks**
Setting this property to true means that WinHTTP.dll is used. Setting this property to false means that the WinInet component is used instead. WinHTTP is server-safe, but it lacks support for asynchronous mode and is unavailable in earlier versions of Microsoft® Windows®, such as Windows 98 and Windows Me. WinInet is widely available but unsafe in multi-thread and server environments.

In general, you must use WinHTTP with MSXML in any context where the URLMON system class (part of WinInet) is not supported or is not well supported. For example, URLMON is not supported for file download in an ASP page. Therefore, you need to set the ServerHTTPRequest property to true if you must handle XML file downloads from within an ASP page. See the document function of XSLT for more information. Also, WinHTTP is recommended in highly multi-threaded scenarios because WinHTTP handles the situation better than URLMON.

Because WinHTTP only supports synchronous loading, the async property must be set to false when the ServerHTTPRequest property is set to true.

For example:

```<%@language=JScript%>
<% var xmlDoc;
doc = Server.CreateObject("Msxml2.DOMDocument.5.0");
doc.async = false;
doc.setProperty("ServerHTTPRequest", true);
doc.load("http://myserver.com");%>
```

**Important** For servers running on an Intranet, the ServerHTTPRequest property requires that you run a proxycfg.exe utility to configure WinHTTP's proxy settings. They cannot be configured through the Microsoft Windows Control Panel. To download the proxy configuration utility, go to the XML Development Center's [WinHTTP Proxy Configuration Utility](#) page. For instructions for running the proxycfg.exe utility, see [Using the WinHTTP Proxy Configuration Utility](#). After you have run the proxycfg.exe tool and updated the registry, the previous registry settings cannot be restored.

**Note** The ServerHTTPRequest functionality is provided by the WinHTTP
APIs in the winhttp5.dll file. To function properly, the winhttp5.dll file must be in the system path, typically located in %windir%\system32.

Applies To

**Component:** MSXML 3.0 and later

**Interface:** [IXMLDOMDocument2](#)

**Method:** [setProperty](#) | [getProperty](#)

UseInlineSchema Property

[This feature was first implemented for MSXML 5.0.]

Specifies whether inline schemas should be processed (true) or not (false). When this property is set to true, inline schemas are used for validation. When this property is set to false, inline schemas are treated like any other XML fragments.

[JScript]

**JScript Syntax**

```javascript
domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
domObj.setProperty(strProp, vBool)
vBool = domObj.getProperty(strProp)
```

[C/C++]

**C/C++ Syntax**

```c/c++
HRESULT setProperty(BSTR strProp, VARIANT vBool);
HRESULT getProperty(BSTR strProp, VARIANT* vBool);
```

**Value**

- **strProp**
  - A BSTR string whose value is "UseInlineSchema".

- **vBool**
  - A VARIANT_BOOL value of true or false. The default value is true.
Remarks

The second-level properties ValidateOnParse, ResolveExternals, UseInlineSchema, and NonValidatingSchemaUse all involve the validation of an XML document. Collectively, they influence the behavior of the parser in the following way.

<table>
<thead>
<tr>
<th>Validate Property Settings</th>
<th>Behavior of the parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidateOnParse = true</td>
<td>Validates against a schema resolved from a schema-cache, an inline schema, or schemaLocation attributes.</td>
</tr>
<tr>
<td>ResolveExternals = true</td>
<td>This is the default behavior.</td>
</tr>
<tr>
<td>UseInlineSchema = true</td>
<td>Validates the document against a schema resolved from a schema-cache or the schemaLocation attributes. An inline schema is ignored.</td>
</tr>
<tr>
<td></td>
<td>Validates the document against a schema resolved from a schema-cache or an inline schema. The schemaLocation attributes are ignored in resolving the schema.</td>
</tr>
<tr>
<td></td>
<td>Validates the document against a schema resolved from a schema-cache. An inline schema and the schemaLocation attributes are ignored.</td>
</tr>
<tr>
<td>ValidateOnParse = false</td>
<td>Process ID/IDREF/datatype using schema resolved from schema-cache, inline schema or the schemaLocation attributes.</td>
</tr>
<tr>
<td>ResolveExternals = true</td>
<td>Process ID/IDREF/datatype using schema resolved from schema-cache or the schemaLocation attributes. Inline schema is ignored.</td>
</tr>
<tr>
<td>UseInlineSchema = true</td>
<td>Process ID/IDREF/datatype using schema resolved from schema-cache or inline schema. The schemaLocation attributes are ignored.</td>
</tr>
</tbody>
</table>
ValidateOnParse = false  
ResolveExternals = false  
UseInlineSchema = false

ValidateOnParse = false  
ResolveExternals = true or false  
UseInlineSchema = true or false

attributes are ignored.  
Process ID/IDREF/datatype using schema resolved from schema-cache.  
Inline schema and the schemaLocation attributes are ignored.  
Schema validation is not performed and ID/IDREF/datatype are not resolved.

Applies To

Component: MSXML 5.0 and later

Interface: IXMLDOMDocument2

Method: setProperty | getProperty

See Also

Using Inline Schemas (XSD) | ResolveExternals | ValidateOnParse
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
ValidateOnParse Property

This feature was first implemented for MSXML 5.0.

This property is used to specify whether the XML parser should validate (true) this document against document type definition (DTD), schema, or schema cache on loading.

[JScript]

**JScript Syntax**

```javascript
domObj.setProperty(strProp, vBool);
vBool = domObj.getProperty(strProp);
```

[Visual Basic]

**Visual Basic Syntax**

```vbscript
domObj.setProperty(strProp, vBool)
vBool = domObj.getProperty(strProp)
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT setProperty(BSTR strProp, VARIANT vBool);
HRESULT getProperty(BSTR strProp, VARIANT* vBool);
```

**Value**

- **strProp**
  A BSTR string whose value is "ValidateOnParse".

- **vBool**
  A VARIANT_BOOL value of true/false. The default value is true.

**Remarks**
The following rules apply for the validation:

1. If an external schema cache is set, the DTD will not be used to validate the document.
2. If an external schema cache is not set, and the document has a DTD, XSD, or XDR schema file, will not be used to validate the document.
3. Mixing XSD and XDR validation outside external cache is not permitted.

ValidateOnParse is also a first-level property on an XML DOM with the same effect. Thus, the following JScript code fragments accomplish the same:

```javascript
dom.setProperty("ValidateOnParse", true);
```

and

```javascript
dom.validateOnParse = true;
```

The second-level properties of ValidateOnParse, ResolveExternals, UseInlineSchema, and NonValidatingSchemaUse all deal with validation of an XML document. Collectively, they influence the behavior of the parser in different ways. For more information see the Remarks section of the UseInlineSchema property.

**Apply To**

**Component:** MSXML 5.0 and later

**Interface:** IXMLDOMDocument2

**Method:** setProperty | getProperty

**See Also**

ResolveExternals | UseInlineSchema
XML DOM Methods

This section describes DOM method syntax.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abort (DOMDocument)</code></td>
<td>Aborts an asynchronous download in progress.</td>
</tr>
<tr>
<td><code>abort (IXMLHTTPRequest)</code></td>
<td>Cancels the current HTTP request.</td>
</tr>
<tr>
<td><code>abort (ServerXMLHTTP/IServerXMLHTTPRequest)</code></td>
<td>Cancels the current HTTP request.</td>
</tr>
<tr>
<td><code>add</code></td>
<td>Adds a new schema to the schema collection and associates the given namespace URI with the specified schema.</td>
</tr>
<tr>
<td><code>addCollection</code></td>
<td>Adds schemas from another collection into the current collection. Replaces any schema on the same namespace URI.</td>
</tr>
<tr>
<td><code>addObject</code></td>
<td>Adds objects to a style sheet.</td>
</tr>
<tr>
<td><code>addParameter</code></td>
<td>Adds parameters to a style sheet.</td>
</tr>
<tr>
<td><code>appendChild</code></td>
<td>Appends a new child node as the last child of the node.</td>
</tr>
<tr>
<td><code>appendData</code></td>
<td>Appends the supplied string to the existing string data.</td>
</tr>
<tr>
<td><code>clone</code></td>
<td>Clones a copy of the current <code>IXMLDOMS</code>...</td>
</tr>
</tbody>
</table>
cloneNode

createAttribute

createCDATASection

createComment

createDocumentFragment

createElement

createEntityReference

createNode

createProcessingInstruction

createProcessor

createTextNode

errorParameters

deleteData

get

with the same position and context.
Clones a new node.
Creates a new attribute with the specified name.
Creates a CDATA section node that contains the supplied data.
Creates a comment node that contains the supplied data.
Creates an empty IXMLDOMDocumentFragment object.
Creates an element node using the specified name.
Creates a new EntityReference object.
Creates a node using the supplied type, name, and namespace.
Creates a processing instruction node that contains the supplied target and data.
Creates a rental-IXSLProcessor object that will use this template.
Creates a text node that contains the supplied data.
Returns the parameter value for a given index.
Deletes specified data.
Returns a read-only XML Document Object Model (DOM) node.
getAllResponseHeaders (IXMLHTTPRequest)
Retrieves the values of all the HTTP headers.

getAllResponseHeaders (ServerXMLHTTP/IServerXMLHTTPRequest)
Retrieves the values of all the HTTP headers.

getAttribute
Gets the value of the attribute.

getAttributeNode
Gets the attribute node.

getDeclaration
Returns the declaration of the DOM node that is sent to the function.

getElementsByTagName (DOMDocument)
Returns a collection of elements that have the specified name.

getElementsByTagName (IXMLDOMElement)
Returns a list of all descendant elements that match the supplied name.

getNamedItem
Retrieves the attribute with the specified name.

getOption
Returns the value of the specified option.

getProperty (IXMLDOMDocument2)
Returns the default properties.

getProperty (IXMLDOMSelection)
Returns a property.

getQualifiedItem Method
Returns the attribute with the specified namespace and attribute name.

getResponseHeader (IXMLHTTPRequest)
Retrieves the value of an HTTP header from the response body.

getResponseHeader (ServerXMLHTTP/IServerXMLHTTPRequest)
Retrieves the value of an HTTP header from the response body.

getSchema Method
Returns an ISchema.

hasChildNodes
Provides a fast way to determine whether
hasFeature

importNode

insertBefore

insertData

item (IXMLDOMNodeList)

item (IXMLDOMNamedNodeMap)

load

loadXML

matches

nextNode (IXMLDOMNodeList)

nextNode (IXMLDOMNamedNodeMap)

nodeFromID

normalize

has children.

Indicates support for specified feature.

Clones a node from a different DOM object.

Inserts a child node left of the specified node or at the end of the list.

Inserts a string at a specified offset.

Allows random access to individual nodes within the collection.

Allows random access to individual nodes within the collection.

Loads an XML document from the specified location.

Loads an XML document using the supplied string.

Checks if the node that is passed is contained in the current collection.

Returns the next node in the collection.

Returns the next node in the collection.

Returns the node that matches the ID at.

Normalizes all descendant elements by combining two or more adjacent text nodes into one unified text node.

Initializes an MSXML2.XMLHTTP...
open (IXMLHTTPRequest)

request and specifies the method, URL, an authentication information for the request.

open (ServerXMLHTTP/IServerXMLHTTPRequest)

Initializes a request and specifies the method, URL, and authentication information for the request.

peekNode

Gets the next node that the nextNode method will return without advancing the list position.

remove

Removes the specified namespace from a collection.

removeAll

Removes all the nodes from the collection described by the IXMLDOMSelection.

removeAttribute

Removes or replaces the named attribute.

removeAttributeNode

Removes the specified attribute from this element.

removeChild

Removes the specified child node from the list of children and returns it.

removeNamedItem

Removes an attribute from the collection.

removeNext

Removes the next node.

removeQualifiedItem

Removes the attribute with the specified namespace and attribute name.

replaceChild

Replaces the specified child node with the supplied new child.

replaceData

Replaces the specified number of characters with
reset (IXMLDOMNamedNodeMap)  
Resets the iterator.

reset (IXMLDOMNodeList)  
Resets the iterator.

reset (IXMLDOMParseErrorCollection)  
Resets the internal position to start, so that the method will return the first error in the list.

reset (IXSLProcessor)  
Resets the state of the processor to the state it was in prior to calling the transform method.

save  
Saves an XML document to the specified location.

selectNodes  
Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

selectSingleNode  
Applies the specified pattern-matching operation to this node's context and returns the first matching node.

send (IXMLHTTPRequest)  
Sends an HTTP request to the server and receives a response.

send (ServerXMLHTTP/IServerXMLHTTPRequest)  
Sends an HTTP request to the server and receives a response.

setAttribute  
Sets the value of the named attribute.

setAttributeNode  
Sets or updates the supplied attribute node on this element.

setNamedItem  
Adds the supplied node to the collection.
**setOption**

Sets the specified option.

**setProperty**

Sets the SelectionLanguage, ServerHTTPRequest, SelectionNamespaces, NewParser internal properties (flags).

**setProxy**

Sets the proxy configuration.

**setProxyCredentials**

Sets the proxy authentication credentials.

**setRequestHeader (IXMLHTTPRequest)**

Specifies the name of an HTTP header.

**setRequestHeader (ServerXMLHTTP/IServerXMLHTTPRequest)**

Specifies the name of an HTTP header.

**setStartMode**

Performs a subset of a larger XSLT transformation by selecting the XSLT mode with which to start.

**setTimeouts**

Specifies timeout settings for resolving the domain name, establishing the connection to the server, sending the data, and receiving the response.

**splitText**

Splits this text node into two text nodes at the specified offset and inserts the new text node into the tree as a sibling that immediately follows this node.

**substringData**

Retrieves a substring of the full string from the specified range.

Starts the transformation.
transform

process or resume a previously failed transformation.

Processes this node and its children using the XSL Transformations (XSLT) style sheet and returns the resulting transformation.

transformNode

Processes this node and its children using the XSLT style sheet, and returns the resulting transformation.

transformNodeToObject

Processes this node and its children using the XSLT style sheet, and returns the resulting transformation in the supplied object.

validate

Performs run-time validation on the currently loaded document using the currently loaded DTD, schema, or schema collection.

validate (IXMLDOMSchemaCollection2/IXMLDOMSchemaCollection)

Performs run-time validation on the documents in the cache that have not been compiled and validated.

validateNode

Validates a specified DOM fragment.

waitForResponse

Allows the requesting server to suspend execution while waiting for an asynchronous operation to complete.
**abort Method (DOMDocument)**

Aborts an asynchronous download in progress.

**[Script]**

**Script Syntax**

```javascript
oXMLDOMDocument.abort();
```

**Example**

```javascript
var xmlDOM;
xmlDOM = new ActiveXObject("Msxml2.DOMDocument50");
xmlDOM.async = true;
xmlDOM.onreadystatechange = doOnReadyStateChange;
xmlDOM.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
}

function doOnReadyStateChange () {
    if (xmlDOM.readyState == 1) {
        xmlDOM.abort();
    }
}
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
Dim WithEvents xmlDOM As Msxml2.DOMDocument50
Private Sub cmdDoAbortDOMDocument_Click()
    Set xmlDOM = New Msxml2.DOMDocument50
    xmlDOM.async = True
```

**Example**

```vbnet
Dim WithEvents xmlDOM As Msxml2.DOMDocument50
Private Sub cmdDoAbortDOMDocument_Click()
    Set xmlDOM = New Msxml2.DOMDocument50
    xmlDOM.async = True
```
xmlDOM.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
End If
End Sub
Private Sub xmlDOM_onreadystatechange()
    If xmlDOM.readyState = 1 Then
        xmlDOM.abort
    End If
End Sub

[C/C++]

C/C++ Syntax

HRESULT abort(
    void
);

C/C++ Return Values

S_OK
    The value returned if successful.

Remarks

This method stops download and parsing and discards any portion of the XML tree already built. IXMLDOMParseError indicates that the download was stopped.

If the readyState property has the value COMPLETED, no action is taken and the current document is unchanged.

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

See Also
IXMLDOMParseError | readyState Property (DOMDocument)

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
abort Method (IXMLHTTPRequest)

Cancels the current HTTP request.

[Script]

Script Syntax

`oXMLHttpRequest.abort();`

Example

```javascript
var xmlhttp;
xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmlhttp.onreadystatechange = doHttpReadyStateChange;
xmlhttp.open("GET", "http://localhost/sample.xml", true);
xmlhttp.send();

function doHttpReadyStateChange() {
    if (xmlhttp.readyState == 2) {
        xmlhttp.abort();
    }
}
```

[Visual Basic]

Visual Basic Syntax

`oXMLHttpRequest.abort`

[C/C++]

C/C++ Syntax

`HRESULT abort(void);`

C/C++ Return Values
S_OK
   The value returned if successful.

Remarks

The request will be returned to the UNINITIALIZED state, and open method must be called next.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

readyState Property (IXMLHTTPRequest) | open Method (IXMLHTTPRequest)

Applies to: IXMLHTTPRequest
abort Method
(ServerXMLHTTP/IServerXMLHTTP)

 Cancels the current HTTP request.

[Script]

Script Syntax

oServerXMLHttpRequest.abort();

Example

```javascript
var xmlServerHttp;
xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.onreadystatechange = doServerHttpReadyStateChange;
xmlServerHttp.open("GET", "http://localhost/sample.xml", true);
xmlServerHttp.send();

function doServerHttpReadyStateChange()
{
  if (xmlServerHttp.readyState == 2)
  {
    xmlServerHttp.abort();
  }
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
oServerXMLHttpRequest.abort
```

[C/C++]

C/C++ Syntax

```c
HRESULT abort(void);
```

C/C++ Return Values
S_OK
   The value returned if successful.

Remarks

The object will be returned to the UNINITIALIZED state, and open method must be called next.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

readyState Property (ServerXMLHTTP/IServerXMLHTTPRequest) | open Method (ServerXMLHTTP/IServerXMLHTTPRequest)

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
add Method

Adds a new schema to the schema collection and associates the given namespace URI with the specified schema.

[Script]

Script Syntax

`objXMLDOMSchemaCol.add(namespaceURI, var);`

Parameters

`namespaceURI`

The namespace to associate with the specified schema. The empty string, """, will associate the schema with the empty namespace, xmlns="". This may be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute also occurs on namespaceURI (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

`var`

This specifies the schema to load. It will load it synchronously and with resolveExternals=false and validateOnParse=false. This parameter can also take any DOMDocument as an argument. This argument can be Null, which results in the removal of any schema for the specified namespaces. If the schema is an IXMLElementNode, the entire document the node belongs to will be preserved.

Example

The following script example attaches a schema to an XML document.
var xmldoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0")
var SchemaCache = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");

xmldoc.async = false;
xmldoc.validateOnParse = false;
SchemaCache.add("x-schema:books", "c:\\books.xsd");
xmldoc.schemas = SchemaCache;
// The document will load only if a valid schema is attached to the
// file.
xmldoc.load("c:\\books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(xmldoc.xml) ;
}

[Visual Basic]

**Visual Basic Syntax**

objXMLDOMSchemaCol.add(namespaceURI, var)

**Parameters**

*namespaceURI*

This is the namespace to associate with the specified schema. The empty
string, "", will associate the schema with the empty namespace, xmlns="".

This may be any string that can be used in an xmlns attribute, but it cannot
contain entity references. The same white space normalization that occurs
on the xmlns attribute also occurs on namespaceURI (that is, leading and
trailing white space is trimmed, new lines are converted to spaces, and
multiple adjacent white space characters are collapsed into one space).

*var*

Specifies the schema to load. It will load it synchronously and with
resolveExternals=false and validateOnParse=false. The schema
parameter can also be any DOMDocument.

This argument can be Null, which results in the removal of any schema for
the specified namespaces. If the schema is an IXMLDOMNode, the entire
document the node belongs to will be preserved.

**Example**

The following Visual Basic example attaches a schema to an XML document.

```vbnet
Dim xmldoc As New Msxml2.FreeThreadedDOMDocument50
Dim SchemaCache As New Msxml2.XMLSchemaCache50

xmldoc.async = False
xmldoc.validateOnParse = True
SchemaCache.Add "x-schema:books", "c:\books.xsd"
Set xmldoc.schemas = SchemaCache
' The document will load only if a valid schema is attached to the x
' file.
xmldoc.Load "c:\books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox xmldoc.xml
End If
```

[C/C++]

**C/C++ Syntax**

```
HRESULT add(BSTR namespaceURI, VARIANT var);
```

**Parameters**

`namespaceURI [in]`

The namespace to associate with the specified schema.

The empty string, "", will associate the schema with the empty namespace, `xmlns=""`. This may be any string that can be used in an `xmlns` attribute, but it cannot contain entity references. The same white space normalization that occurs on the `xmlns` attribute also occurs on this parameter (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).
var [in]

This specifies the schema. It can be a BSTR, in which case it points to the URL to load. It will load it synchronously and with resolveExternals=false and validateOnParse=false. The var parameter can also be any DOMDocument.

This argument can be Null, which results in the removal of any schema for the specified namespaces. If the schema is an IXMLDOMNode, the entire document the node belongs to will be preserved.

C/C++ Return Values

If this call fails, the collection remains unchanged. E_FAIL is returned if:

- The collection is read-only.
- The document is not a recognized schema.
- An error occurs when compiling the schema.
- The ready state of the document is not 4.

If it was loading a schema and encountered a parse error, then the parse error reason is returned in the IErrorInfo. If the VARIANT argument contains an invalid value, E_INVALIDARG is returned.

Example

```
#include "stdafx.h"
#include "tchar.h"
#import "c:\winnt\system32\msxml5.dll"
using namespace MSXML2;

void AddCollectionSample();

int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow)
{
    ::CoInitialize(NULL);
    AddCollectionSample();
    ::CoUninitialize();
    return 0;
}
```
void AddCollectionSample()
{
    IXMLDOMDocument2Ptr pIXMLDOMDocument2;
    IXMLDOMSchemaCollection2Ptr pIXMLDOMSchemaCollection2Ptr;
    int nResult;

    try
    {
        // Create the DOM
        nResult = pIXMLDOMDocument2.CreateInstance(__uuidof(MSXML2::DOMDocument50));
        (nResult == 0) ? 0 : throw nResult;

        // Create the Schema Collections
        nResult = pIXMLDOMSchemaCollection2Ptr.CreateInstance(__uuidof(MSXML2::XMLSchemaCache50));
        (nResult == 0) ? 0 : throw nResult;

        // Add the schema to the collection
        nResult = pIXMLDOMSchemaCollection2Ptr->add(_T("x-schema:books"), _T("c:\books.xsd"));
        (nResult == 0) ? 0 : throw nResult;

        // Attach schemas
        pIXMLDOMDocument2->schemas = pIXMLDOMSchemaCollection2Ptr.GetInterfacePtr();
        pIXMLDOMDocument2->async = false;
        pIXMLDOMDocument2->validateOnParse = true;

        // Load the document into the DOM
        nResult = pIXMLDOMDocument2->load(_T("c::\books.xml"));
        (nResult == -1) ? 0 : throw nResult;

        ::MessageBox(NULL, pIXMLDOMDocument2->xml, _T("Loaded Document"), MB_OK);
    }
    catch(...)
    {
        ::MessageBox(NULL, _T("Sample Failed"), _T("Error"), MB_OK);
    }
}

File Name: c:\books.xml

<?xml version='1.0'?><Collection xmlns="x-schema:books">
    <Book>
        <Title>Lover Birds</Title>
        <Author>Cynthia Randall</Author>
        <Publisher>Lucerne Publishing</Publisher>
    </Book>
</Collection>
File Name: c:\books.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Collection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OtherBook">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Title" type="xs:string"/>
              <xs:element name="Author" type="xs:string"/>
              <xs:element name="Publisher" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Output (in a message box)

```xml
<?xml version='1.0'?>
<Collection xmlns="x-schema:books">
  <Book>
    <Title>Lover Birds</Title>
    <Author>Cynthia Randall</Author>
    <Publisher>Lucerne Publishing</Publisher>
  </Book>
</Collection>
```

Remarks

Schemas referenced by this schema are not added to the collection. The contents of the schema are added to the internal collection. (The schema can be copied.) If a schema is already in the collection with this namespace, the new one will replace it.

The XmlSchemaCache object does not retrieve imported or included schemas during validation if `validateOnLoad` is set to False. In this situation, if a.xsd imports or includes another schema file, b.xsd, and you need to use the `add` method to add b.xsd separately to the schema cache as shown in the following sample JScript code for validation to run successfully as expected:
var schemas = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
schemas.validateOnLoad = false
schemas.add("http://www.example.com/a/", "a.xsd");
/* Add all imported/included schemas by hand */
schemas.add("http://www.example.com/b/", "b.xsd");
schemas.validate()

This only applies if you are not using the default setting of True for validateOnLoad. For example, the following code below will import and include b.xsd if it is referenced in a.xsd.

var schemas = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
schemas.validateOnLoad = true; /* THIS IS THE DEFAULT */
schemas.add("http://www.example.com/a/", "a.xsd");
schemas.validate()

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSchemaCollection/XMLSchemaCache
addCollection Method

Adds schemas from another collection into the current collection and replaces any schemas that collide on the same namespace URI.

[Script]

Script Syntax

objXMLDOMSchemaCol.addCollection(objXMLDOMSchemaCollection);

Parameters

objXMLDOMSchemaCollection
  The collection containing the schemas to add.

Example

```javascript
var xmldoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var SchemaCache = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");
var SchemaCache2 = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");

xmldoc.async = false;
xmldoc.validateOnParse = false;
SchemaCache.add("x-schema:books", "c:\books.xsd");
SchemaCache2.addCollection(SchemaCache);
SchemaCache2.add("x-schema:books", "c:\NewBooks.xsd");
xmldoc.schemas = SchemaCache2;
// The document will load only if a valid schema is attached to the
// file. The new schema will override the old one
xmldoc.load("c:\books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  alert(xmldoc.xml);
}
```

[Visual Basic]
Visual Basic Syntax

objXMLDOMSchemaCol.addCollection(objXMLDOMSchemaCollection)

Parameters

objXMLDOMSchemaCollection
   The collection containing the schemas to add.

Example

Dim xmldoc As New Msxml2.FreeThreadedDOMDocument50
Dim SchemaCache As New Msxml2.XMLSchemaCache50
Dim SchemaCache2 As New Msxml2.XMLSchemaCache50

xmldoc.async = False
xmldoc.validateOnParse = True
SchemaCache.Add "x-schema:books", "c:\books.xsd"
SchemaCache2.addCollection SchemaCache
SchemaCache2.Add "x-schema:books", "c:\newbooks.xsd"
Set xmldoc.schemas = SchemaCache2
' The document will load only if a valid schema is attached to the x
' file. The new schema will override the old one.
xmldoc.Load "c:\books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
   Dim myErr
   Set myErr = xmlDoc.parseError
   MsgBox("You have error " & myErr.reason)
Else
   MsgBox xmldoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT addCollection(IXMLDOMSchemaCollection * otherCollection);

Parameters

otherCollection [in]
   The collection containing the schemas to add.
C/C++ Return Values

S_OK
The value returned if the collection is added successfully.

E_FAIL is returned if:
- The collection is read-only.
- The document is not a recognized schema.
- An error occurs when compiling the schema.
- The ready state of the document is not 4.

E_POINTER
The value returned if input pointer is invalid.

Example

```c++
#include "stdafx.h"
#include "tchar.h"
#import "c:\winnt\system32\msxml5.dll"
using namespace MSXML2;

void AddCollectionSample();

int APIENTRY WinMain(HINSTANCE hInstance,
            HINSTANCE hPrevInstance,
            LPSTR lpCmdLine,
            int nCmdShow)
{
    ::CoInitialize(NULL);
    AddCollectionSample();
    ::CoUninitialize();
    return 0;
}

void AddCollectionSample()
{
    IXMLDOMDocument2Ptr pIXMLDOMDocument2;
    IXMLDOMSchemaCollection2Ptr pIXMLDOMSchemaCollection2Ptr;
    IXMLDOMSchemaCollection2Ptr pIXMLDOMSchemaCollection2Ptr2;
    int nResult;

    try
    {
        // Create the DOM
        nResult = pIXMLDOMDocument2.CreateInstance(__uuidof(MSXML2::DOMDocument50));
        (nResult == 0) ? 0 : throw nResult;
    }
```
Create the Schema Collections
nResult = pIXMLDOMSchemaCollection2Ptr.CreateInstance(__uuidof(MSXML2::XMLSchemaCache50))
(nResult == 0) ? 0 : throw nResult;
nResult = pIXMLDOMSchemaCollection2Ptr2.CreateInstance(__uuidof(MSXML2::XMLSchemaCache50))
(nResult == 0) ? 0 : throw nResult;

Add the schema to the collection
nResult = pIXMLDOMSchemaCollection2Ptr->add(_T("x-schema:books"),
_T("c:\books.xsd"));
(nResult == 0) ? 0 : throw nResult;

attach all schemas from "pIXMLDOMSchemaCollection2Ptr"
(nResult == 0) ? 0 : throw nResult;

override the old schema
(nResult == 0) ? 0 : throw nResult;

Attach schemas
pIXMLDOMDocument2->schemas = pIXMLDOMSchemaCollection2Ptr2.Get
pIXMLDOMDocument2->async = false;
pIXMLDOMDocument2->validateOnParse = true;

Load the document into the DOM
nResult = pIXMLDOMDocument2->load(_T("c:\books.xml"));
(nResult == -1) ? 0 : throw nResult;

MessageBox(NULL, pIXMLDOMDocument2->xml, _T("Loaded Document")
} catch(...) {
    MessageBox(NULL, _T("Sample Failed"), _T("Error"), MB_OK);
}

File Name: c:\books.xml

<?xml version='1.0'?><Collection xmlns="x-schema:books">
    <Book>
        <Title>Lover Birds</Title>
        <Author>Cynthia Randall</Author>
        <Publisher>Lucerne Publishing</Publisher>
    </Book>
</Collection>

File Name: c:\books.xsd
<xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Collection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OtherBook">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Title" type="xs:string"/>
              <xs:element name="Author" type="xs:string"/>
              <xs:element name="Publisher" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

File Name: c:\NewBooks.xsd

<xml version='1.0'?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Collection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Title" type="xs:string"/>
              <xs:element name="Author" type="xs:string"/>
              <xs:element name="Publisher" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

Output (in a message box)

<xml version='1.0'?><Collection xmlns="x-schema:books">
  <Book>
    <Title>Lover Birds</Title>
    <Author>Cynthia Randall</Author>
    <Publisher>Lucerne Publishing</Publisher>
  </Book>
</Collection>
Remarks

There is no guarantee that the two collections will return exactly the same schema. The collection is free to clone them, if necessary.

Adding a collection to itself has no effect.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSchemaCollection/XMLSchemaCache
addObject Method

Adds objects into a style sheet.

[Script]

Script Syntax

objXSLProcessor.addObject(obj, namespaceURI);

Parameters

obj

The object to pass in. Optionally, you can pass a null value here to signal to the processor that an object previously added should be released.

namespaceURI

The namespace to use inside the style sheet to identify the object.

Example

The following script example passes an object to the style sheet.

```javascript
var xsldoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xmldoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xsltemp = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslproc;

xsldoc.load("d:\\inetpub\\wwwroot\\sampleXSLWithObject.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  xsltemp.stylesheet = xsldoc.documentElement;
  xslproc = xsltemp.createProcessor();
  xmldoc.loadXML("<level>Twelve</level>");
  xslproc.input = xmldoc;

  xslproc.addObject(xmldoc, "urn:my-object");
  xslproc.transform();
  alert(xslproc.output);
```
File Name: d:\inetpub\wwwroot\sampleXSLWithObject.xml

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
xmlns:myObj="urn:my-object">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <xsl:element name="stage">
      <xsl:value-of select="myObj:get-text()"/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Output

```xml
<?xml version="1.0" encoding="UTF-16"?>
<stage>
  Twelve
</stage>
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
objXSLProcessor.addObject(obj, namespaceURI)
```

**Parameters**

**obj**

The object to pass in. Optionally, you can pass a null object value here such as Nothing to signal to the processor that an object previously added should be released.

**namespaceURI**

The namespace to use inside the style sheet to identify the object.

**Example**

The following Visual Basic example passes an object to the style sheet.

```vbnet
Dim xsldoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlldoc As New Msxml2.FreeThreadedDOMDocument50
Dim xslttemp As New Msxml2.XSLTemplate50
```
Dim xslproc As Msxml2.IXSLProcessor

xsldoc.Load "d:\inetpub\wwwroot\sampleXSLWithObject.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xsltemp.stylesheet = xsldoc.documentElement
    Set xslproc = xsltemp.createProcessor
    xmlDoc.loadXML "<level>Twelve</level>"
    xslproc.input = xmlDoc
    xslproc.addObject xmlDoc, "urn:my-object"
    xslproc.Transform
    MsgBox xslproc.output
End If

File Name: d:\inetpub\wwwroot\sampleXSLWithObject.xml

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
xmlns:myObj="urn:my-object">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
        <xsl:element name="stage">
            <xsl:value-of select="myObj:get-text()"/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>

Output

<?xml version="1.0" encoding="UTF-16"?>
<stage>
    Twelve
</stage>

[C/C++]

C++ Syntax

HRESULT addObject (IDispatch* obj, BSTR namespaceURI);

Parameters

obj [in]
The object to pass in. Optionally, you can pass a NULL IDispatch here to signal to the processor that an object previously added should be released.

**namespaceURI [in]**
The namespace that will be used inside the style sheet to identify the object.

### C/C++ Return Values

E_FAIL if the value of the readyState property is READYSTATE_INTERACTIVE.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

### Remarks

Numbers are coerced to double, everything is coerced into a string, and objects return an error.

The syntax `get-` is used to retrieve the property value exposed by the object that was passed into the style sheet. In the preceding example, the value for the property 'text' was retrieved as follows:

```xml
<xsl:value-of select="myObj:get-text()"/>
```

For example, if the object that was passed into the style sheet included a property called `sheepcount`, you would use the following syntax to retrieve the value of that property.

```xml
<xsl:value-of select="myObj:get-sheepcount()"/>
```

### See Also

Applies to: IXSLProcessor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
addParameter Method

Adds parameters into an XSL Transformations (XSLT) style sheet.

[Script]

Script Syntax

objXSLProcessor.addParameter(baseName, parameter, namespaceURI);

Parameters

baseName

The name that will be used inside the style sheet to identify the parameter context.

parameter

In most cases, a number, Boolean, string, IXMLDOMNodeList, or IXMLDOMNode. Passing in a single node will produce a node list that contains one node (shortcut). To remove a parameter previously added to the processor, provide a value of Empty or Null instead. This acts as a signal to the processor to remove any previously added parameter of the same name.

namespaceURI (optional)

An optional namespace.

Example

var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslt.stylesheet = xslDoc;
    var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmlDoc.async = false;
```xml
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslProc = xslt.createProcessor();
    xslProc.input = xmlDoc;
    xslProc.addParameter("param1", "Hello");
    xslProc.transform();
    alert(xslProc.output);
}
```

**File Name: Sample.xsl**

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    The parameter value was: <xsl:value-of select="$param1"/>
  </xsl:template>
</xsl:stylesheet>
```

**Visual Basic Syntax**

```
objXSLProcessor.addParameter(baseName, parameter, namespaceURI)
```

**Parameters**

- **baseName**
  The name that will be used inside the style sheet to identify the parameter context.

- **parameter**
  A number, Boolean, string, IXMLDOMNodeList, or IXMLDOMNode. Passing in a single node will produce a node list that contains one node (shortcut). To remove a parameter previously added to the processor, provide a value of Empty or Null instead. This acts as a signal to the processor to remove any previously added parameter of the same name.

- **namespaceURI (optional)**
  An optional namespace.
Example

Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
    xmlDoc.async = False
    xmlDoc.Load "books.xml"
    If (xmlDoc.parseError.errorCode <> 0) Then
        Dim myErr
        Set myErr = xmlDoc.parseError
        MsgBox("You have error " & myErr.reason)
    Else
        Set xslProc = xslt.createProcessor()
        xslProc.input = xmlDoc
        xslProc.addParameter "param1", "Hello"
        xslProc.Transform
        MsgBox xslProc.output
    End If
End If

File Name: Sample.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        The parameter value was: <xsl:value-of select="$param1"/>
    </xsl:template>
</xsl:stylesheet>

[C/C++]

C/C++ Syntax

HRESULT addParameter (BSTR baseName, VARIANT parameter, BSTR namespaceURI);
Parameters

**baseName** [in]
The name that will be used inside the style sheet to identify the parameter context.

**parameter** [in]
A number, Boolean, string, node list, or node. Passing in a single node will produce a node list that contains one node (shortcut). To remove a parameter previously added to the processor, you can pass a value of VT_EMPTY, VT_NULL, or a NULL IDispatch or IUnknown instead. This acts as a signal to the processor to remove any previously added parameter of the same name.

**namespaceURI** [in, optional]
An optional namespace.

C/C++ Return Values

E_FAIL if readyState is READYSTATE_INTERACTIVE.

Example

```c
#include "stdio.h"

#import <msxml5.dll>
using namespace MSXML2;

int checkParseError(IXMLDOMParseErrorPtr pError);
void dump_com_error(_com_error &e);

int main(int argc, char* argv[])
{
    CoInitialize(NULL);
    HRESULT hr;

    try{
        BOOL bResult = FALSE;
        short sResult = FALSE;

        IXMLDOMDocument2Ptr pStyleSheet=NULL;
```
IXSLTemplatePtr pIXSLTemplate=NULL;
IXSLProcessorPtr pXSLProcessor=NULL;

hr = pIXSLTemplate.CreateInstance(__uuidof(XSLTemplate50));

hr=pStyleSheet.CreateInstance(__uuidof(FreeThreadedDOMDocument);
StyleSheet->async = VARIANT_FALSE;
hr=pStyleSheet->load("c:\\samplexsl.xml");
//check on the parser error
if(hr!=VARIANT_TRUE)
{
    return checkParseError(pStyleSheet->parseError);
}

pIXSLTemplate->stylesheet = pStyleSheet.GetInterfacePtr();
pXSLProcessor = pIXSLTemplate->createProcessor();

IXMLDOMDocumentPtr pInputDoc;

hr = pInputDoc.CreateInstance(__uuidof(DOMDocument50));
pInputDoc->async = VARIANT_FALSE;
hr = pInputDoc->load("c:\\sampleXSLWithParam.xml");
//check on the parser error
if(hr!=VARIANT_TRUE)
{
    return checkParseError(pInputDoc->parseError);
}

pInputDoc->async = VARIANT_FALSE;
pXSLProcessor->input = pInputDoc.GetInterfacePtr();

hr=pXSLProcessor->addParameter("param1", "Hello", "");

VARIANT_BOOL vtRet = pXSLProcessor->transform();
if(vtRet != VARIANT_TRUE)
{
    MessageBox(NULL, "transformation failed","Error", MB_OK);
    return -1;
}

_bstr_t bstrOutput = pXSLProcessor->Getoutput();

MessageBox(NULL, bstrOutput,"Transformed Output", MB_OK);

try{
}
catch(_com_error &e)
{
    dump_com_error(e);
}
return 0;
int checkParseError(IXMLDOMParseErrorPtr pError)
{
    _bstr_t parseError = _bstr_t("At line ") + _bstr_t(pError->Getline());
    MessageBox(NULL, parseError, "Parse Error", MB_OK);
    return -1;
}

void dump_com_error(_com_error &e)
{
    printf("Error\n");
    printf("\a\tCode = %08lx\n", e.Error());
    printf("\a\tCode meaning = %s", e.ErrorMessage());
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
    printf("\a\tSource = %s\n", (LPCSTR) bstrSource);
    printf("\a\tDescription = %s\n", (LPCSTR) bstrDescription);
}

Style Sheet: "d:\inetpub\wwwroot\sampleXSLWithParam.xml"

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        The parameter value was: <xsl:value-of select="$param1"/>
    </xsl:template>
</xsl:stylesheet>

Output (in a message box)

<?xml version="1.0" encoding="UTF-16"?>
<bar>
Add Parameter Test
</bar>

Remarks

The addParameter method can be called on transformNode handlers and between transform calls (in asynchronous processing), and further processing will use the updated parameter. Added parameters are referenced by <xsl:param> within the style sheet.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: IXSLProcessor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
appendChild Method

Appends a new child node as the last child of the node.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNode.appendChild(newChild);
```

Parameters

**newChild**

An object. Address of the new child node to be appended at the end of the list of children belonging to this node.

Return Value

An object. Returns the new child node successfully appended to the list.

Example

The following script example creates a new IXMLDOMNode object, and then uses the appendChild method to append it to the document's list of children.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var newNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    alert(root.xml);
    newNode = xmlDoc.createNode(1,"VIDEOS",""(Parser)));
    root.appendChild(newNode);
```
alert(root.xml);
}

Remarks

This is equivalent to calling insertBefore(newChild, null). For more information, see insertBefore Method.

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNode.appendChild(newChild)

Parameters

newChild

An object. The address of the new child node to be appended at the end of the list of children belonging to this node.

Return Value

An object. Returns the new child node successfully appended to the list.

Example

The following Microsoft® Visual Basic® example creates a new IXMLDOMNode object, and then uses the appendChild method to append it to the document's list of children.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim newNode As IXMLDOMNode

xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Dim myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  MsgBox root.xml
  Set newNode = xmlDoc.createElement(NODE_ELEMENT, "VIDEOS", "")
  root.appendChild newNode
  MsgBox root.xml
End If

Remarks

This is equivalent to calling insertBefore(newChild, null). For more information, see insertBefore Method.

[C/C++]

C/C++ Syntax

HRESULT appendChild(
    IXMLDOMNode *newChild,
    IXMLDOMNode **outNewChild);

Parameters

newChild [in]
    The address of the new child node to be appended to the end of the list of children of this node.
outNewChild [out, retval]
    The new child node successfully appended to the list. If Null, no object is created.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the newChild parameter is Null.
E_FAIL
    The value returned if an error occurs.

To view reference information for Visual Basic, C/C++, or Script only, click the
Remarks

If the newChild parameter has an existing parent, the node is automatically removed from that parent before being inserted into its new location.

A node retains its namespace information even when moved. Moving a node does not create a namespace declaration; declarations are added when retrieving the XML source (through the save method or the xml property) to ensure that all namespaces are properly declared.

When inserting a node tree under another node that has a different owner document, the ownerDocument property for each inserted node is changed to match the owner document of its new parent.

When moving a node tree to another document, the content of all entity reference nodes contained therein is updated to conform to the new document. If the new document does not declare an entity that was moved into it, the entity reference will have no children, and the old content is removed. Existing references to nodes under the entity reference are still valid, but the node whose parent previously was the entity reference now has a null parent.

See Also

save Method | xml Property | ownerDocument Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
appendData Method

Appends the supplied string to the existing string data.

[Script]

Script Syntax

```javascript
oXMLDOMCharacterData.appendData(data);
```

Parameters

data

A string containing the data that is to be appended to the existing string.

Example

The following script example creates an IXMLDOMComment object and uses the appendData method to add text to the string.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    comment = xmlDoc.createComment("Hello World!");
    comment.appendData("Ellohay Orldway!");
    alert(comment.data);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
oXMLDOMCharacterData.appendData(data)
```
Parameters

\textit{data}

A string containing the data that is to be appended to the existing string.

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMComment object and uses the appendData method to add text to the string.

\begin{verbatim}
Dim xmlDoc As New Msxml2.DOMDocument50
Dim comment As IXMLDOMComment
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set comment = xmlDoc.createComment("Hello World!")
  comment.appendData("Ellohay Orldway!")
  MsgBox (comment.Data)
End If
\end{verbatim}

[C/C++]

C/C++ Syntax

\begin{verbatim}
HRESULT appendData(
    BSTR data);
\end{verbatim}

Parameters

\textit{data} [in]

The string data to be appended to the existing string.

C/C++ Return Values

\begin{verbatim}
S_OK
    The value returned if successful.
S_FALSE
\end{verbatim}
The value returned if the string is not appended.

E_FAIL
The value returned if an error occurs.

**Remarks**

The `length` property is also updated by this operation.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[length Property (IXMLDOMCharacterData)]

Applies to: [IXMLDOMCDATASection] | [IXMLDOMCharacterData] | [IXMLDOMComment] | [IXMLDOMText]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
clone Method

Clones a copy of the current IXMLDOMSelection, with the same position and context.

[Script]

Script Syntax

var NewObjXMLDOMSelection = objXMLDOMSelection.clone();

Example

//Create a DOMDocument2 object and load some XML:
xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.loadXML("<root><elem1>foo</elem1><elem2>bar</elem2></root>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    //Create an XMLDOMSelection object from selected nodes:
    xpath = xmlDoc.selectNodes("root/elem1");
    //Cache the XPath expression and context:
    xpath.expr = "root/elem1";
    xpath.context = xmlDoc;

    //Clone the XMLDOMSelection object
    xpath2 = xpath.clone();
    temp1 = xpath.peekNode(); // temp1 == <elem1/>
    temp2 = xpath2.peekNode(); // temp2 == <elem1/> Note the position and context are maintained.
}

[Visual Basic]

Visual Basic Syntax

Set NewObjXMLDOMSelection = objXMLDOMSelection.clone

Example
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xpath As IXMLDOMSelection
Dim xpath2 As IXMLDOMSelection
Dim temp1 As IXMLDOMNode
Dim temp2 As IXMLDOMNode
xmlDoc.loadXML("<root><elem1>foo</elem1><elem2>bar</elem2></root>")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    ' Create an XIXMLDOMSelection object from selected nodes:
    Set xpath = xmlDoc.selectNodes("root/elem1")
    ' Cache the XPath expression and context:
    xpath.expr = "root/elem1"
    Set xpath.context = xmlDoc
    ' Clone the XIXMLDOMSelection object.
    Set xpath2 = xpath.Clone()
    Set temp1 = xpath.peekNode() ' temp1 == <elem1/>
    Set temp2 = xpath2.peekNode() ' temp2 == <elem1/> Note the
    ' context are maintained.
End If

[C/C++]

C/C++ Syntax

HRESULT clone(IXMLDOMSelection ** ppNode);

Parameters

ppNode [out, retval]
    The returned copy of the IXMLDOMSelection object.

C/C++ Return Values

S_OK
    The value returned if the cloning is successful.
E_INVALIDARG
    The value returned if the input argument is Null.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [IXMLDOMSelection](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
cloneNode Method

Clones a new node.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNode.cloneNode(deep);
```

Parameters

`deep`

Boolean. A flag that indicates whether to recursively clone all nodes that are descendants of this node. If True, creates a clone of the complete tree below this node. If False, clones this node and its attributes only.

Return Value

Object. Returns the newly created clone node.

Example

The following script example clones a node, and then appends it as a child of the top-level node.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var currNode;
var MyNewNode;
xmlDoc.async = false;
xmlDoc.load("books.xml")
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    currNode = root.childNodes.item(1);
```
MyNewNode = currNode.cloneNode(true);
root.appendChild(MyNewNode);
alert(xmlDoc.xml);
}

[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNode = oXMLDOMNode.cloneNode(deep)

**Parameters**

*deep*  
Boolean. A flag that indicates whether to recursively clone all nodes that are descendants of this node. If True, creates a clone of the complete tree below this node. If False, clones this node and its attributes only.

**Return Value**

An object. Returns the newly created clone node.

**Example**

The following Microsoft® Visual Basic® example clones a node, and then appends it as a child of the top-level node.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim currNode As IXMLDOMNode
Dim MyNewNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set currNode = root.childNodes.Item(1)
    Set MyNewNode = currNode.cloneNode(True)
    root.appendChild MyNewNode
```
(MsgBox xmlDoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT cloneNode(
    VARIANT_BOOL deep,
    IXMLDOMNode **cloneRoot);

Parameters

deep [in]
    A flag that indicates whether to recursively clone all nodes that are
descendants of this node. If True, creates a clone of the complete tree below
this node. If False, clones this node and its attributes only.
cloneRoot [out, retval]
    A newly created clone node.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the cloneRoot parameter is Null.

Remarks

The cloned node has the same property values as this node for the following
properties: nodeName property, nodeValue property, nodeType property,
parentNode property, ownerDocument property, and, if it is an element,
attributes property. The value of the clone's childNodes property depends on
the setting of the deep flag parameter.

Note  If the node is the DOMDocument node, it is safer to clone the document
using the save method, as follows.

doc.save(doc2);
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

See Also

nodeName Property | nodeValue Property | nodeType Property | parentNode Property | ownerDocument Property | attributes Property | childNodes Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
createAttribute Method

Creates a new attribute with the specified name.

[Script]

Script Syntax

var objXMLDOMAttribute = oXMLDOMDocument.createAttribute(name);

Parameters

name

A string specifying the name of the new attribute object. This name is subsequently available as the new node’s nodeName property.

Return Value

An object. Returns the new IXMLDOMNode Attribute object.

Example

The following script example creates a new attribute called ID and adds it to the attributes of the DOMDocument object.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var newAtt;
var namedNodeMap;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    newAtt = xmlDoc.createAttribute("ID");
    namedNodeMap = root.attributes;
```
(namedNodeMap.setNamedItem(newAtt);
    for (var i=0; i<namedNodeMap.length; i++) {
        alert(namedNodeMap.item(i).xml);
    }
}

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMAttribute = oXMLDOMDocument.createAttribute(name)

Parameters

name

A string specifying the name of the new attribute object. This name is subsequently available as the new node's nodeName property.

Return Value

An object. Returns the new IXMLDOMAttribute object.

Example

The following Visual Basic example creates a new attribute called ID and adds it to the attributes of the DOMDocument object.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim newAtt As IXMLDOMAttribute
Dim namedNodeMap As IXMLDOMNamedNodeMap
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set newAtt = xmlDoc.createAttribute("ID")
    Set namedNodeMap = root.Attributes
    namedNodeMap.setNamedItem newAtt
    For i = 0 To (namedNodeMap.length - 1)
MsgBox (namedNodeMap.Item(i).xml)
Next
End If

[C/C++]

C/C++ Syntax

HRESULT createAttribute(
    BSTR name,
    IXMLDOMAttribute **attribute);

Parameters

name [in]
The name of the new attribute object. This name is subsequently available as the new node's nodeName property.

attribute [out, retval]
The address of the new IXMLDOMAttribute object.

C/C++ Return Values

S_OK
The value returned if successful.

E_INVALIDARG
The value returned if the attribute parameter is Null.

E_FAIL
The value returned if an error occurs.

Remarks

Creating an attribute with this method is the same as using createNode where the type parameter value is NODE_ATTRIBUTE and no namespace is specified.

You cannot create a namespace-qualified attribute using the createAttribute method. Regardless of whether a namespace prefix is included in the name parameter, the namespaceURI property for the new attribute is set to an empty string, "". An attribute constructed as part of an XML document load operation will never have both a prefix and an empty namespace Uniform Resource
Identifier (URI). You can only create a namespace-qualified attribute using the createNode method of the DOMDocument.

No data value is set for the attribute during the create operation. You can set the value by calling the setAttribute method of the element object.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the ownerDocument property of the new node points to this document object, the parentNode property remains null. To associate the attribute with an element, call the setAttributeNode method of the IXMLDOMElement object.

Because the parentNode property of an attribute always returns a Null value, this property will not change after associating the new attribute with an element using the setAttribute method.

The nodeType property has the value NODE_ATTRIBUTE.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

createNode Method | namespaceURI Property (IXMLDOMNode) | setAttribute Method | ownerDocument Property | parentNode Property | nodeType Property | nodeName Property | IXMLDOMElement | IXMLDOMAttribute

Applies to: DOMDocument
createCDATASection Method

Creates a CDATA section node that contains the supplied data.

[Script]

Script Syntax

var objXMLDOMCDATASection = oXMLDOMDocument.createCDATASection(data)

Parameters

data
   A string specifying the value to be supplied to the new IXMLDOMCDATASection object's nodeValue property.

Return Value

An object. Returns the new IXMLDOMCDATASection object.

Example

The following script example creates a new CDATA section node and appends it as the last child node of the DOMDocument object.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var CDATASection;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
   var myErr = xmlDoc.parseError;
   alert("You have error " + myErr.reason);
} else {
   root = xmlDoc.documentElement;
   CDATASection = xmlDoc.createCDATASection("Hello World!");
   root.appendChild(CDATASection);
   alert(root.xml);
Visual Basic Syntax

Set objXMLDOMCDATASEction = oXMLDOMDocument.createCDATASEction(data)

Parameters

data
A string specifying the value to be supplied to the new CDATASEction object's nodeValue property.

Return Value

An object. Returns the new IXMLDOMCDATASEction object.

Example

The following Visual Basic example creates a new CDATA section node and appends it as the last child node of the DOMDocument object.

```
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim CDATASEction As IXMLDOMCDATASEction
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  Set CDATASEction = xmlDoc.createCDATASEction("Hello World!")
  root.appendChild CDATASEction
  MsgBox (root.xml)
End If
```

[C/C++]

C/C++ Syntax
HRESULT createCDATASection(
    BSTR data,
    IXMLDOMCDATASection **cdata);

Parameters

data [in]
    The value to be supplied to the new CDATA section object's nodeValue property.
cdata [out, retval]
    The address of the new IXMLDOMCDATASection.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the cdata parameter is Null.
E_FAIL
    The value returned if an error occurs.

Remarks

Creating a CDATA section with this method is the same as using createNode where the type parameter value is NODE_CDATA_SECTION and no namespace is specified. You cannot specify a namespace with the createCDATASection method.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the ownerDocument property of the new node points to this document object, the parentNode property is set to Null. To add the new object, you must explicitly call one of the node insert methods, insertBefore method, replaceChild method, or appendChild method.

The new object's nodeType property has the value NODE_CDATA_SECTION.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

createNode Method | ownerDocument Property | parentNode Property | insertBefore Method | nodeType Property | nodeValue Property | replaceChild Method | appendChild Method | IXMLDOMCDATASection

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
createComment Method

Creates a comment node that contains the supplied data.

[Script]

Script Syntax

var objXMLDOMComment = oXMLDOMDocument.createComment(data);

Parameters

data
    The string specifying the value to be supplied to the new Comment object's
    nodeValue property.

Return Value

An object. Returns the new IXMLDOMComment.

Example

The following script example creates a new comment node and appends it as the
last child node of the DOMDocument object.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var comment;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    comment = xmlDoc.createComment("Hello World!");
    root.appendChild(comment);
    alert(root.xml);
Visual Basic Syntax

Set objXMLDOMComment = oXMLDOMDocument.createComment(data)

Parameters

data
A string specifying the value to be supplied to the new Comment object's nodeValue property.

Return Value

An object. Returns the new IXMLDOMComment.

Example

The following Visual Basic example creates a new comment node and appends it as the last child node of the DOMDocument object.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim comment As IXMLDOMComment
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  Set comment = xmlDoc.createComment("Hello World!")
  root.appendChild comment
  MsgBox (root.xml)
End If

[C/C++]

C/C++ Syntax
HRESULT createComment(
    BSTR data,
    IXMLDOMComment **comment);

Parameters

data [in]
    A value to be supplied to the new comment object's nodeValue property.

comment [out, retval]
    The address of the new IXMLDOMComment object.

C/C++ Return Values

S_OK
    The value returned if successful.

E_INVALIDARG
    The value returned if the comment parameter is Null.

E_FAIL
    The value returned if an error occurs.

Remarks

Creating a comment with this method is the same as using createNode where the type parameter value is NODE_COMMENT and no namespace is specified. You cannot specify a namespace with the createComment method.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the ownerDocument property of the new node points to this document object, the parentNode property is set to Null. To add the new object, you must explicitly call one of the node insert methods, insertBefore method, replaceChild method, or appendChild method.

The new object's nodeType property has the value NODECOMMENT.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
**createNode Method** | **ownerDocument Property** | **parentNode Property** | **insertBefore Method** | **replaceChild Method** | **appendChild Method** | **nodeType Property** | **nodeValue Property** | **IXMLDOMComment**

Applies to: **DOMDocument**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
createDocumentFragment Method

Creates an empty IXMLDOMDocumentFragment object.

[Script]

Script Syntax

```javascript
var objXMLDOMDocumentFragment = oXMLDOMDocument.createDocumentFragment();
```

Return Value

An object. Returns the empty IXMLDOMDocumentFragment object.

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var docFragment;
xmlDoc.async = false;
xmlDoc.loadXML("<root/>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    docFragment = xmlDoc.createDocumentFragment();
    docFragment.appendChild(xmlDoc.createElement("node1"));
    docFragment.appendChild(xmlDoc.createElement("node2"));
    docFragment.appendChild(xmlDoc.createElement("node3"));
    alert(docFragment.xml);
    xmlDoc.documentElement.appendChild(docFragment);
    alert(xmlDoc.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objXMLDOMDocumentFragment = oXMLDOMDocument.createDocumentFragment;
```
Return Value

An object. Returns the empty IXMLDOMDocumentFragment object.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim docFragment As IXMLDOMDocumentFragment
xmlDoc.async = False
xmlDoc.loadXML "<root/>
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set docFragment = xmlDoc.createDocumentFragment()
    docFragment.appendChild xmlDoc.createElement("node1")
    docFragment.appendChild xmlDoc.createElement("node2")
    docFragment.appendChild xmlDoc.createElement("node3")
    MsgBox docFragment.xml
    xmlDoc.documentElement.appendChild docFragment
    MsgBox xmlDoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT createDocumentFragment(
    IXMLDOMDocumentFragment **docFrag);

Parameters

docFrag [out, retval]
    The address of the empty IXMLDOMDocumentFragment.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the docFrag parameter is Null.
E_FAIL
The value returned if an error occurs.

Remarks

Creating a document fragment with this method is the same as using createNode where the type parameter value is NODE_DOCUMENT_FRAGMENT and no namespace is specified. You cannot specify a namespace with the createDocumentFragment method.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the ownerDocument property of the new node points to this document object, the parentNode property is set to Null. To add the new object, you must explicitly call one of the node insert methods, insertBefore method, replaceChild method, or appendChild method.

The nodeType property has the value NODE_DOCUMENT_FRAGMENT.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

createNode Method | ownerDocument Property | parentNode Property |
nodeType property | insertBefore Method | replaceChild Method |
appendChild Method | IXMLDOMDocumentFragment

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
createElement Method

Creates an element node using the specified name.

[Script]

Script Syntax

```javascript
var objXMLDOMElement = oXMLDOMDocument.createElement(tagName);
```

Parameters

`tagName`
A string specifying the name for the new element node. The name is case-sensitive. This name is subsequently available as the element node's `nodeName` property.

Return Value

An object. Returns the `IXMLDOMElement` object for the new element.

Example

The following script example creates an element called PAGES and appends it to an `IXMLDOMNode` object. It then sets the text value of the element to 400.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var newElem;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    newElem = xmlDoc.createElement("PAGES");
    root.childNodes.item(0).appendChild(newElem);
    newElem.textContent = 400;
}```
Set objXMLDOMElement = oXMLDOMDocument.createElement(tagName)

Parameters

tagName
A string specifying the name for the new element node. The name is case-sensitive. This name is subsequently available as the element node's nodeName property.

Return Value

An object. Returns the IXMLDOMElement object for the new element.

Example

The following Microsoft® Visual Basic® example creates an element called PAGES and appends it to an IXMLDOMNode object. It then sets the text value of the element to 400.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim newElem As IXMLDOMElement
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
   Dim myErr
   Set myErr = xmlDoc.parseError
   MsgBox("You have error " & myErr.reason)
Else
   Set root = xmlDoc.documentElement
   Set newElem = xmlDoc.createElement("PAGES")
   root.childNodes.Item(0).appendChild newElem
   root.childNodes.Item(0).lastChild.Text = "400"
   MsgBox root.childNodes.Item(0).xml
End If

[C/C++]

**C/C++ Syntax**

```cpp
HRESULT createElement(
    BSTR tagName,
    IXMLDOMElement **element);
```

**Parameters**

- `tagName` **[in]**
  The name for the new element node. It is case-sensitive. This name is subsequently available as the element node's `nodeName` property.

- `element` **[out,retval]**
  The address of the `IXMLDOMElement` interface for the new element.

**C/C++ Return Values**

- **S_OK**
  The value returned if successful.

- **E_INVALIDARG**
  The value returned if the `element` parameter is Null.

- **E_FAIL**
  The value returned if an error occurs.

**Remarks**

Creating an element with this method is the same as using `createElement` where the type parameter value is `NODE_ELEMENT` and no namespace is specified.

You cannot create a namespace-qualified element using the `createElement` method. Regardless of whether a namespace prefix is included in the `tagName` parameter, the `namespaceURI` property for the new element node is set to an empty string, "". An element node constructed as part of an XML document load operation will never have both a prefix and an empty namespace Uniform Resource Identifier (URI). You can only create a namespace-qualified element using the `createTextNode` method of the `DOMDocument` object.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the `ownerDocument` property of the new node points to this document object, the `parentNode` property is set to Null. To add the new object, you must explicitly call one of the node insert methods, `insertBefore` method, `replaceChild` method, or `appendChild` method.

The `nodeType` property has the value `NODE_ELEMENT`.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- `createNode Method` | `namespaceURI Property (IXMLDOMNode)` | `ownerDocument Property` | `parentNode Property` | `nodeName Property` | `nodeType Property` | `insertBefore Method` | `replaceChild Method` | `appendChild Method` | `IXMLDOMElement`

Applies to: `DOMDocument`
createEntityReference Method

Creates a new EntityReference object.

[Script]

Script Syntax

var objXMLDOMEntityReference = oXMLDOMDocument.createEntityReference

Parameters

name
A string specifying the name of the entity referenced. This name is subsequently available as the new object's nodeName property.

Return Value

An object. Returns the new IXMLDOMEntityReference object.

Example

The following script example creates a new IXMLDOMEntityReference object and appends it to an IXMLDOMNode object.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var MyEntity;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    MyEntity = xmlDoc.createEntityReference("newRef");
    root.childNodes.item(0).appendChild(MyEntity);
    alert(root.xml);
```
Visual Basic Syntax

Set objXMLDOMEntityReference = oXMLDOMDocument.createEntityReference

Parameters

name
A string specifying the name of the entity referenced. This name is subsequently available as the new object's nodeName property.

Return Value

An object. Returns the new IXMLDOMEntityReference object.

Example

The following Visual Basic example creates a new IXMLDOMEntityReference object and appends it to an IXMLDOMNode object.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim MyEntity As IXMLDOMEntityReference
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  Set MyEntity = xmlDoc.createEntityReference("newRef")
  root.childNodes.Item(0).appendChild MyEntity
  MsgBox root.xml
End If

[C/C++]

C/C++ Syntax
HRESULT createEntityReference(
    BSTR name,
    IXMLDOMEntityReference **entityRef);

Parameters

name [in]
    The name of the entity referenced. This name is subsequently available as the new object's nodeName property.

entityRef [out, retval]
    The address of the new IXMLDOMEntityReference object.

C/C++ Return Values

S_OK
    The value returned if successful.

E_INVALIDARG
    The value returned if the entityRef parameter is Null.

E_FAIL
    The value returned if an error occurs.

Remarks

Creating an entity reference with this method is the same as using createNode where the type parameter value is NODE_ENTITY_REFERENCE and no namespace is specified. You cannot specify a namespace within the name parameter.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the ownerDocument property of the new node points to this document object, the parentNode property is set to Null. To add the new object, you must explicitly call one of the node insert methods, insertBefore method, replaceChild method, or appendChild method.

The nodeType property has the value NODE_ENTITY_REFERENCE.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

nodeName Property | nodeType Property | createNode Method | ownerDocument Property | parentNode Property | insertBefore Method | replaceChild Method | appendChild Method | IXMLDOMEntityReference

Applies to: DOMDocument
**createNode Method**

Creates a node using the supplied type, name, and namespace.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMDocument.createNode(Type, name, namespaceURI);
```

**Parameters**

*Type*
A variant. A value that uniquely identifies the node type. This can be specified using either the integer value or the string value. For the complete list of values, see [XML DOM Enumerated Constants](#) or the description for the `name` parameter.

*name*
A string containing the value for the new node's `nodeName` property. The relationship between the `name` and `Type` parameters is summarized in the Remarks section of this topic.

*namespaceURI*
A string defining the namespace URI. If specified, the node is created in the context of the `namespaceURI` parameter with the prefix specified on the node name. If the `name` parameter does not have a prefix, this is treated as the default namespace.

**Return Value**

Object. Returns the newly created node.

**Example**

The following script example creates an attribute node called Sci-Fi and adds it to the attributes for the DOMDocument object.
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var MyNode;
var namedNodeMap;
xmlDoc.async = false;
xmlDoc.load("books.xml");
MyNode = xmlDoc.createNode(2, "Sci-Fi", "");
namedNodeMap = xmlDoc.documentElement.childNodes.item(0).attributes;
namedNodeMap.setNamedItem(MyNode);
alert(xmlDoc.documentElement.xml);

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMDocument.createNode(Type, name, namespace

Parameters

Type
A variant. A value that uniquely identifies the node type. This can be
specified using either the integer value or the string value. For the complete
list of values, see the XML DOM Enumerated Constants or the description
for the name parameter.

name
A string containing the value for the new node's nodeName property. The
relationship between the name and Type parameters is summarized in the
Remarks section of this topic.

namespaceURI
A string defining the namespace URI. If specified, the node is created in the
context of the namespaceURI parameter with the prefix specified on the
node name. If the name parameter does not have a prefix, this is treated as
the default namespace.

Return Value

An object. Returns the newly created node.

Example

The following Microsoft® Visual Basic® example creates an attribute node
called Sci-Fi and adds it to the attributes for the DOMDocument object.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim MyNode As IXMLDOMNode
Dim namedNodeMap As IXMLDOMNamedNodeMap
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set MyNode = xmlDoc.createNode(NODE_ATTRIBUTE, "Sci-Fi", ">
    Set namedNodeMap = xmlDoc.documentElement.childNodes.Item(0).Attributes
    namedNodeMap.setNamedItem MyNode
    MsgBox (xmlDoc.documentElement.xml)
End If

[C/C++]

C/C++ Syntax

HRESULT createNode(
    VARIANT Type,
    BSTR name,
    BSTR namespaceURI,
    IXMLDOMNode **node);

Parameters

Type [in]
The value that uniquely identifies the node type. This can be specified using either the integer value or the string value. For the complete list of values, see the XML DOM Enumerated Constants or the description given for the name parameter.

name [in]
The value for the new node's nodeName property. The relationship between the name and Type parameters is summarized in the Remarks section of this topic.

namespaceURI [in]
The namespace URI. If specified, the node is created in the context of the namespaceURI parameter with the prefix specified on the node name. If the name parameter does not have a prefix, this is treated as the default
namespace.

node [out, retval]
   The newly created node.

**C/C++ Return Values**

S_OK
   The value returned if successful.
E_INVALIDARG
   The value returned if the node parameter is Null.
E_FAIL
   The value returned if an error occurs.

**Remarks**

The **name** parameter depends on the value of the **Type** parameter:

- **NODE_ATTRIBUTE**
  The name of the attribute.

- **NODECDATA_SECTION, NODE_COMMENT, NODE_DOCUMENT, NODE_DOCUMENT_FRAGMENT, NODE_TEXT**
  The **nodeName** property for these node types is a constant value; the name parameter is ignored.

- **NODE_DOCUMENT_TYPE**
  The name of the document type; for example, the **xxx** in `<!DOCTYPE xxx ...>`.

- **NODE_ELEMENT**
  The name of the XML tag, with any namespace prefix included if present.

- **NODE_ENTITY**
  The name of the entity.

- **NODE_ENTITY_REFERENCE**
  The name of the entity referenced. The name does not include the leading ampersand or the trailing semicolon. The name includes the namespace, if one is present.

- **NODE_NOTATION**
  The name of the notation.

- **NODE_PROCESSING_INSTRUCTION**
  The target, the first token following the `<?` characters.

You cannot create a node of type **NODE_DOCUMENT, NODE_DOCUMENT_TYPE**,
When a node is created, it is created in the context of a namespace if one is supplied (if the namespaceURI parameter is supplied). If one is not supplied, the node is created in the namespace of the document. If the namespaceURI parameter is specified, the node is created in the context of the namespaceURI parameter with the prefix specified on the node name.

For node types that do not have names, the empty string, "", should be passed as the name parameter.

For elements and entity references, when the value of the namespaceURI parameter is anything other than "", and the value of the name parameter does not contain a prefix (xxx in xxx:yyy), the value of the namespaceURI parameter is treated as the default namespace.

Attributes cannot be scoped to a default namespace, and other elements are not qualified to a particular namespace; they are treated as being from the namespace defined by the document itself.

When the value of namespaceURI parameter is the empty string, "", the node is created without a namespace. Creating a qualified node without specifying a nonempty namespaceURI returns an error. This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeName Property

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
createProcessingInstruction Method

Creates a processing instruction node that contains the supplied target and data.

[Script]

Script Syntax

var objXMLDOMProcessingInstruction =
oXMLDOMDocument.createProcessingInstruction(target, data);

Parameters

target
A string specifying the target part of the processing instruction. This supplies the nodeName property of the new object.

data
A string specifying the rest of the processing instruction preceding the closing ?> characters. This supplies the nodeValue property for the new object.

Return Value

An object. Returns the new IXMLDOMProcessingInstruction object.

Example

The following script example specifies the target string "xml" and the data string "version="1.0"" to generate the processing instruction <<?XML version="1.0"?>.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var pi;
xmldoc.async = false;
xmldoc.load("books.xml");
if (xmldoc.parseError.errorCode <> 0) {
   var myErr = xmldoc.parseError;
alert("You have error " + myErr.reason);
} else {
    pi = xmlDoc.createProcessingInstruction("xml", "version="1.0"");
    xmlDoc.insertBefore(pi, xmlDoc.childNodes.item(0));
    alert(xmlDoc.xml);
}

[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMProcessingInstruction = oXMLDOMDocument.createProcessingInstruction(target, data)

**Parameters**

- **target**
  A string specifying the target part of the processing instruction. This supplies the `nodeName` property of the new object.

- **data**
  A string specifying the rest of the processing instruction preceding the closing `?>` characters. This supplies the `nodeValue` property for the new object.

**Return Value**

An object. Returns the new `IXMLDOMProcessingInstruction` object.

**Example**

The following Microsoft® Visual Basic® example specifies the target string "xml" and data string "version = "1.0"" to generate the processing instruction `<?XML version="1.0">`.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim pi As IXMLDOMProcessingInstruction
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
```
MsgBox("You have error " & myErr.reason)
Else
    Set pi = xmlDoc.createProcessingInstruction("xml", "version=""1.0"
    xmlDoc.insertBefore pi, xmlDoc.childNodes.Item(0)
    MsgBox (xmlDoc.xml)
End If

[C/C++]

C/C++ Syntax

HRESULT createProcessingInstruction(
    BSTR target,
    BSTR data,
    IXMLDOMProcessingInstruction **pi);

Parameters

target [in]
    The target part of the processing instruction. It supplies the nodeName
    property of the new object.

data [in]
    The remainder of the processing instruction preceding the closing ?>
    characters. It supplies the nodeValue property for the new object.

pi [out, retval]
    The address of the new IXMLDOMProcessingInstruction object.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the pi parameter is Null.
E_FAIL
    The value returned if an error occurs.

Remarks

Creating a processing instruction node with this method is the same as using
createNode where the Type parameter value is NODE_PROCESSING_INSTRUCTION
and no namespace is specified. You cannot specify a namespace with the createProcessingInstruction method.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. In other words, although the ownerDocument property of the new node points to this document object, the parentNode property is set to Null. To add the new object, you must explicitly call one of the node insert methods, insertBefore method, replaceChild method, or appendChild method.

The new object's nodeType property has the value NODE_PROCESSING_INSTRUCTION.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

nodeName Property | nodeType Property | nodeValue Property | IXMLDOMProcessingInstruction | createNode Method | ownerDocument Property | insertBefore Method | replaceChild Method | appendChild Method

Applies to: DOMDocument
createProcessor Method

Creates a rental-model IXSLProcessor object that will use this template.

[Script]

Script Syntax

```javascript
var objXSLProcessor = objXSLTemplate.createProcessor();
```

Example

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslt.stylesheet = xslDoc;
    var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmlDoc.async = false;
    xmlDoc.load("books.xml");
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    } else {
        xslProc = xslt.createProcessor();
        xslProc.input = xmlDoc;
        xslProc.addParameter("param1", "Hello");
        xslProc.transform();
        alert(xslProc.output);
    }
}
```

File Name: Sample.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
```
The parameter value was: 

Example

Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
    xmlDoc.async = False
    xmlDoc.Load "books.xml"
    If (xmlDoc.parseError.errorCode <> 0) Then
        Dim myErr
        Set myErr = xmlDoc.parseError
        MsgBox("You have error " & myErr.reason)
    Else
        Set xslProc = xslt.createProcessor()
        xslProc.input = xmlDoc
        xslProc.addParameter "param1", "Hello"
        xslProc.Transform
        MsgBox xslProc.output
    End If
End If

File Name: Sample.xsl
The parameter value was: <xsl:value-of select="$param1"/>
</xsl:template>
</xsl:stylesheet>

[C/C++]

C/C++ Syntax

HRESULT createProcessor(IXSLProcessor** ppProcessor);

Parameters

ppProcessor [out, retval]
   The returned processor associated with this template.

C/C++ Return Values

E_OUTOFMEMORY

E_FAIL
   The value returned if the template has no style sheet.

Remarks

Multiple processors can be created from the same IXSLTemplate object.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLTemplate
createTextNode Method

Creates a text node that contains the supplied data.

[Script]

Script Syntax

var objXMLDOMText = oXMLDOMDocument.createTextNode(data);

Parameters

data
A string specifying the value to be supplied to the new text object's
nodeValue property.

Return Value

An object. Returns the new IXMLDOMText object.

Example

The following script example creates an IXMLDOMText object, and then uses the
insertBefore method to place it before the first child node of the document.

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var MyText;
var MyNewNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    root = xmlDoc.documentElement;
    MyText = xmlDoc.createTextNode("Hello World");
    MyNewNode = root.insertBefore(MyText, root.childNodes.item(0));
alert(xmlDoc.xml);
}

[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMText = oXMLDOMDocument.createTextNode(data)

**Parameters**

data
A string specifying the value to be supplied to the new text object's nodeValue property.

**Return Value**

An object. Returns the new IXMLDOMText object.

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMText object, and then uses the insertBefore method to place it before the first child node of the document.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim MyText As IXMLDOMText
Dim MyNewNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set root = xmlDoc.documentElement
    Set MyText = xmlDoc.createTextNode("Hello World")
    Set MyNewNode = root.insertBefore(MyText, root.childNodes.Item(0))
    MsgBox(xmlDoc.xml)
End If
```
[C/C++]

C/C++ Syntax

HRESULT createTextNode(
    BSTR data,
    IXMLDOMText **text);

Parameters

data [in]
    The value to be supplied to the new text's nodeValue.
text [out, retval]
    The address of new IXMLDOMText.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the text parameter is Null.
E_FAIL
    The value returned if an error occurs.

Remarks

Creating a text node with this method is the same as using createNode where the Type parameter value is NODE_TEXT and no namespace is specified. You cannot specify a namespace with the createTextNode method.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. (In other words, although the ownerDocument property of the new node points to this document object, the parentNode property is set to Null.) To add the new object, you must explicitly call one of the node insert methods, insertBefore method, replaceChild method, or appendChild method.

The new nodeType property has the value NODE_TEXT.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeValue Property | IXMLDOMText | createNode Method | ownerDocument Property | parentNode Property | nodeType Property | insertBefore Method | replaceChild Method | appendChild Method

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Referenc
**deleteData Method**

Deletes specified data.

[Script]

**Script Syntax**

```
oXMLDOMCharacterData.deleteData(offset, count);
```

**Parameters**

- **offset**
  A long integer value specifying the offset, in characters, at which to start deleting string data.

- **count**
  A long integer value specifying the number of characters to delete.

**Example**

The following script example creates an IXMLDOMComment object and uses the `deleteData` method to delete the last six characters.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    comment = xmlDoc.createComment("Hello World!");
    comment.deleteData(6, 6);
    alert(comment.xml);
}
```

[Visual Basic]

**Visual Basic Syntax**
oXMLDOMCharacterData.deleteData(offset, count)

Parameters

offset
A long integer value specifying the offset, in characters, at which to start deleting string data.

count
A long integer value specifying the number of characters to delete.

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMComment object and uses the deleteData method to delete the last six characters.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim comment As IXMLDOMComment
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set comment = xmlDoc.createComment("Hello World!")
    comment.deleteData 6, 6
    MsgBox(comment.xml)
End If

[C/C++]

C/C++ Syntax

HRESULT deleteData(
    long offset,
    long count);

Parameters

offset [in]
The offset, in characters, at which to start deleting string data.

count [in]
The number of characters to delete.

**C/C++ Return Values**

- **S_OK**
  - The value returned if successful.
- **S_FALSE**
  - The value when returning Null.
- **E_FAIL**
  - The value returned if an error occurs.

**Remarks**

If the offset and count of characters specify a range beyond the end of the string, this method deletes only to the end of the string.

The `length` property is updated by this operation.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[length Property (IXMLDOMCharacterData)]

Appplies to: [IXMLDOMNodeCDATASection] | [IXMLDOMCharacterData] | [IXMLDOMComment] | [IXMLDOMText]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**errorParameters Method**

Returns the parameter value for a given index. This parameter value is a string that is used in the construction of the error message. For example, when a validation error is reported on a missing element, the first error parameter, errorParameters(0), shows where the missing element is located and the second error parameter, errorParameters(1), shows which element is missing.

**Note** The error parameters returned from this method are specific to each error message. An error message might not have any error parameters.

[Script]

**Script Syntax**

```
strParam = objXMLDOMParseError2.errorParameters(index);
```

[Visual Basic]

**Visual Basic Syntax**

```
strParam = objXMLDOMParseError2.errorParameters(index);
```

[C/C++]

**C/C++ Syntax Using Smart Pointers**

```
_bstr_t strParam = objXMLDOMParseError2->errorParameters (index);
```

**C/C++ Syntax**

```
HRESULT errorParameters (  
    LONG index,
    BSTR *strParam);
```

**Parameters**
index [in]
A long integer representing the index of an error parameter.

strParam [out,retval]
A BSTR string containing the error parameter of the given index.

Return Values

S_OK
The error parameter has been retrieved successfully.

E_FAIL
The index is out of bounds and the resultant strParam parameter is NULL.

Example

This example uses the same two resource files used in the allErrors example, books.xml and books.xsd. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (books.xml and books.xsd)
- JScript Code (errorParams.js)
- Visual Basic Code (errorParams.frm)
- C/C++ Code (errorParams.cpp)
- Output for the errorParams Example

Applies To

IXMLDOMParseError2 Interface

Versioning

MSXML 5.0 and later
JScript Code (errorParams.js)

```javascript
var xs, xd;

errorParameters();

function errorParameters()
{
    try {
        xs = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
        xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
    }
    catch (e) {
        alert("Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office is not installed. Download and install MSXML 5.0 from http://msdn.microsoft.com before continuing.");
        return;
    }

    try {
        xs.add("urn:books", "books.xsd");
        xd.schemas = xs;
        xd.async = false;
        xd.validateOnParse = false;
    }
    catch (e) {
        alert("Failed to add to schema cache: " + e.description);
        return;
    }

    try {
        xd.load("books.xml");
    }
    catch (e) {
        alert("can't load books.xml : " + e.description);
        return;
    }

    msg = "Validating DOM...
    var err = xd.validate();
    if (err.errorCode != 0) {
        msg += "invalid dom:\n        + " + err.reason + "\terrorXPath:\n        + " + err.errorXPath + "\nParameters Count: " + err.errorParametersCount + "\n        for (var i=0; i<err.errorParametersCount; i++)
```
function alert(msg) {
    WScript.echo(msg);
}

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as
**errorParams.js**, in the same directory where you saved **books.xml** and **books.xsd**.

4. Double click the **errorParams.js** file from Windows Explorer to launch the application. Alternatively, you can type "errorParams.js" from a command prompt.

   **Note**  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (**wscript.exe**), if it is not already installed.

5. Verify that your output is the same as that listed in **Output for the errorParams Example**.
Private Sub Form_Resize()
    ' Resize the text box to the size of the form
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Private Sub Form_Load()
    ' Resize the text box to the size of the form
    Text1.Top = 100
    Text1.Left = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750

    ' Check "Microsoft XML, v50" from Project->References...
    ' in Visual Studio.
    Dim er, e As IXMLDOMParseError2
    Dim xd As New DOMDocument50
    Dim xs As New XMLSchemaCache50

    ' Add xs to xd.
    xs.Add "urn:books", App.Path & "\books.xsd"
    Set xd.schemas = xs
    xd.async = False
    xd.validateOnParse = False
    xd.setProperty "MultipleErrorMessages", True

    msg = "Validating DOM..." & vbCrLf

    ' Load XML to xd.
    xd.Load App.Path & "\books.xml"
    Set er = xd.Validate
    If er.errorCode <> 0 Then
        msg = msg & "invalid dom:" & vbCrLf
        msg = msg & " reason:" & vbCrLf & vbCrLf
        msg = msg & " errorXPath:" & vbCrLf & vbCrLf
        msg = msg & " errorParametersCount:	" & vbCrLf
        msg = msg & vbCrLf
        For i = 0 To er.errorParametersCount - 1
            msg = msg & " errorParameters(" & vbCrLf
            msg = msg & " & vbCrLf & vbCrLf
            msg = msg & vbCrLf
            msg = msg & vbCrLf
            Next
        Else
            msg = msg & " DOM is valid:" & vbCrLf
    End If
msg = msg & xd.xml
End If

msg = msg & vbCrLf
msg = msg & "Validating nodes..." & vbCrLf
Set pNodes = xd.selectNodes("//book")
For Each pNode In pNodes
    Set er = xd.validateNode(pNode)
    If er.errorCode <> 0 Then
        msg = msg & vbCrLf
        msg = msg & "Node is invalid:" & vbCrLf
        msg = msg & " reason:" & vbCrLf & er.reason
        msg = msg & " errorXPath:" & vbCrLf & er.errorXPath
        msg = msg & vbCrLf
        msg = msg & "Parameters Count: " & vbCrLf & er.errorParametersCount
        msg = msg & vbCrLf
        For i = 0 To er.errorParametersCount - 1
            msg = msg & " errorParameter(" & vbCrLf & er.errorParameters(i) & vbCrLf & vbCrLf
            Next
        Else
            msg = msg & vbCrLf
            msg = msg & "Node is valid: " & vbCrLf
            msg = msg & pNode.xml & vbCrLf
        End If
    Next
    Text1 = msg
End Sub

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as errorParams.vbp to the same directory where you saved books.xml and books.xsd. Name the form file errorParams.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v5.0.
5. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named "Text1". Select the Text1 control and and in the Properties window, modify the following settings: select "2 -
Vertical" as the value for the **ScrollBars** property and set the value of the **MultiLine** property to True.

6. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor to replace whatever code is already there.

7. Execute the code by selecting **Start** from the **Run** menu.

8. Verify that your output is the same as that listed in **Output for the errorParams Example**.
C/C++ Code (errorParams.cpp)

```c
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    HRESULT hr;
    CoInitialize(NULL);

    IXMLDOMDocument3Ptr pXMLDoc;
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XML DOM"\n);
        return -1;
    }
    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
    if (FAILED(hr)) {
        printf("Failed to enable mulitple validation errors\n"\n);
        return -1;
    }
    if (pXMLDoc->load("books.xml")!=VARIANT_TRUE) {
        printf("Failed to load DOM from books.xml. %s\n", (LPCSTR)pXMLDoc->parseError->reason);
        return -1;
    }

    IXMLDOMDocument3Ptr pXSDDoc;
    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XSD Doc"\n);
        return -1;
    }
    pXSDDoc->async = VARIANT_FALSE;
    pXSDDoc->validateOnParse = VARIANT_FALSE;
    if (pXSDDoc->load("books.xsd")!=VARIANT_TRUE) {
```

```c
```
printf("Failed to load XSD from books.xsd. %s\n", 
    (LPCSTR)pXSSDDoc->parseError->reason);
return -1;
}

IXMLDOMSchemaCollectionPtr pSCache;
hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr)) {
    printf("Cannot instantiate XMLSchemaCache50\n");
    return -1;
}
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr)) {
    printf("Cannot add 'urn:books' to schema cache.\n");
    return -1;
}
printf("Validating DOM...\n");
// Validate the entire DOM object.
IXMLDOMParseError2Ptr pError = pXMLDoc->validate();
if (pError->errorCode != 0) {
    printf("invalid dom:\n\treason:\n%s",
        (LPCSTR)pError->reason);
    printf("errorXPath:\n%s",
        (LPCSTR)pError->errorXPath);
    printf("nParameters Count: %d\n",
        pError->errorParametersCount);
    for (int i=0; i<pError->errorParametersCount; i++)
    {
        printf("errorParameters(%d): %s\n",
                i, (LPCSTR)pError->errorParameters(i));
    }
} else
    printf("tDOM is valid:\n\ns\n",
        (LPCSTR)pXMLDoc->xml);

printf("\n\nValidating nodes...\n");
IXMLDOMNodeListPtr pNodeList;
pNodeList = pXMLDoc->selectNodes("//book");
for (int i=0; i<pNodeList->length; i++)
{
    pError = pXMLDoc->validateNode(pNodeList->item[i]);
    if (pError->errorCode != 0)
{  
  printf("\nNode is invalid:\n\treason: %s",
         (LPCSTR)pError->reason);
  printf("\terrorXPath: %s",
         (LPCSTR)pError->errorXPath);
  printf("\nParameters Count: %d\n",
         pError->errorParametersCount);
  for (int j=0; j<pError->errorParametersCount; j++)
  {
    printf("\terrorParameters(%d): %s\n",
           j, (LPCSTR)pError->errorParameters(j));
  }
}
else
{
  IXMLDOMNodePtr pNode=pNodeList->item[i];
  printf("\nNode is valid:\n\n%s\n",
         (LPCSTR)pNode->xml);
}
}

pError.Release();
pXMLDoc.Release();
pNodeList.Release();
pSCache.Release();
pXSDDoc.Release();

CoUninitialize();
return 0;
}

Try It!

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "errorParamsProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
4. Select FileView on the project browser, and highlight errorParamsProj files. From the File menu, select New.
5. On the Files tab of the New dialog box, highlight C++ Source File. Then type "books.xml" in the File name text box.
6. Click **OK**.
7. Copy the XML data file (**books.xml**), and paste it into the text file you just created.
8. Repeat steps 4-7 for the XSD listing (**books.xsd**).

   **Note**  You can also copy these files into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ source code above (**errorParams.cpp**).
10. Build the sample by selecting **Build errorParamsProj.exe** from the **Build** menu.
11. Execute the sample application by selecting **!Execute errorParamsProj.exe** from the **Build** menu.
12. Verify that your output is the same as that listed in **Output for the errorParams Example**.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Output for the errorParams Example

When you build and run the errorParams code, you should get the following output in a message box.

Validating DOM...
invalid dom:
  reason:
Element 'review' is unexpected according to content model of parent
Expecting: price.
  errorXPath:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/review
Parameters Count: 3
  errorParameters(0): book
  errorParameters(1): review
  errorParameters(2): price

Validating nodes...

Node is valid:
<book id="bk001">
  <author>Hightower, Kim</author>
  <title>The First Book</title>
  <genre>Fiction</genre>
  <price>44.95</price>
  <pub_date>2000-10-01</pub_date>
  <review>An amazing story of nothing.</review>
</book>

Node is invalid:
  reason: Element 'review' is unexpected according to content
Expecting: price.
  errorXPath:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[2]/review
Parameters Count: 3
  errorParameters(0): book
  errorParameters(1): review
  errorParameters(2): price

Node is invalid:
  reason: Element 'review' is unexpected according to content
Expecting: pub_date.
  errorXPath:
/*[local-name()="books" and namespace-uri()="urn:books"]/book[3]/rev
Parameters Count: 3
  errorParameters(0): book
  errorParameters(1): review
  errorParameters(2): pub_date
**get Method**

Returns a read-only XML Document Object Model (DOM) node that contains the `<Schema>` element.

**[Script]**

**Script Syntax**

```javascript
var objXMLDOMNode = objXMLDOMSchemaCol.get(namespaceURI);
```

**Example**

```javascript
)
var oSchemaCache = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");
var nsTarget = "";
oSchemaCache.add(nsTarget, "po.xsd");
var oDOMNode = oSchemaCache.get(nsTarget);
```

**Parameters**

`namespaceURI`

The namespace URI associated with the schema to return.

This can be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute also occurs on this argument (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

**[Visual Basic]**

**Visual Basic Syntax**

```visualbasic
Set objXMLDOMNode = objXMLDOMSchemaCol.get(namespaceURI)
```

**Example**
Dim oSchemaCache As New XMLSchemaCache
Dim oDOMNode As DOMDocument
Dim nsTarget As String

nsTarget = ""
oSchemaCache.Add nsTarget, "po.xsd"
Set oDOMNode = oSchemaCache.get(nsTarget)

**Parameters**

*namespaceURI*

The namespace URI associated with the schema to return.

This can be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute also occurs on this argument (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

[C/C++]

**C/C++ Syntax**

HRESULT get(BSTR namespaceURI, IXMLDOMNode ** schemaNode);

**Parameters**

*namespaceURI* [in]

The namespace URI associated with the schema to return.

This may be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute also occurs on this argument (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

*schemaNode* [out, retval]

A read-only IXMLDOMNode that represents the schema that is returned.

**C/C++ Return Values**
S_OK
   The value returned if successful.

Remarks

This will not necessarily be the same document object provided by the add method, because the add method might have copied the schema. For inline schemas, this will apply directly to the <Schema> node embedded within the document.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

add Method

Applies to: IXMLDOMSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getAllResponseHeaders Method (IXMLHttpRequest)

Retrieves the values of all the HTTP headers.

[Script]

Script Syntax

strValue = oXMLHttpRequest.getAllResponseHeaders();

Return Value

A string. Contains the resulting header information.

Example

var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmlhttp.open("GET", "http://localhost/sample.xml", false);
xmlhttp.send();
alert(xmlhttp.getAllResponseHeaders());

[Visual Basic]

Visual Basic Syntax

strValue = oXMLHttpRequest.getAllResponseHeaders

Return Value

A string. Contains the resulting header information.

Example

Dim xmlhttp As New Msxml2.xmlhttp
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.getAllResponseHeaders()

[C/C++]

C/C++ Syntax

HRESULT getAllResponseHeaders(BSTR *pbstrHeaders);

Parameters

pbstrHeaders [out, retval]
   The resulting header information.

C/C++ Return Values

S_OK
   The value returned if successful.

C/C++ Example

HRESULT hr;
BSTR bstrValue = NULL;
IXMLHttpRequest *pIXMLHttpRequest = NULL;

try
{
   // Create XMLHttpRequest object and initialize pIXMLHttpRequest.
   hr = pIXMLHttpRequest->getAllResponseHeaders(&bstrValue);
   if(SUCCEEDED(hr))
      ::MessageBox(NULL, bstrValue, _T("All Response Headers"), MB_OK)
}
catch(...)
{
   DisplayErrorToUser();
}
// Release pIXMLHttpRequest when finished with it.

Remarks

Each header name/value pair is separated by a combination carriage return-line
feed character (vbCrLf in Microsoft® Visual Basic®).

The results of this method are valid only after the send method has been successfully completed.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☐ in the upper-left corner of the page.

See Also

send Method (IXMLHTTPRequest) | getResponseHeader Method (IXMLHTTPRequest) | setRequestHeader Method (IXMLHTTPRequest)

Applies to: IXMLHTTPRequest
**getAllResponseHeaders Method**

*(ServerXMLHTTP/IServerXMLHTTP)*

Retrieves the values of all the HTTP headers.

**[Script]**

**Script Syntax**

```javascript
strValue = oServerXMLHTTPRequest.getAllResponseHeaders();
```

**Example**

```javascript
var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.open("GET", "http://localhost/sample.xml", false);
xmlServerHttp.send();
alert(xmlServerHttp.getAllResponseHeaders());
```

**[Visual Basic]**

**Visual Basic Syntax**

```vbnet
strValue = oServerXMLHTTPRequest.getAllResponseHeaders()
```

**Example**

```vbnet
Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.getAllResponseHeaders()
```

**[C/C++]**

**C/C++ Syntax**

```c
HRESULT getAllResponseHeaders(BSTR *pbstrHeaders);
```
Parameters

`pbstrHeaders [out, retval]`
The resulting header information.

C/C++ Return Values

S_OK
The value returned if successful.

Remarks

Each header name/value pair is separated by a combination carriage return–line feed character (vbCrLf in Microsoft® Visual Basic®).

The results of this method are valid only after the `send` method has been successfully completed.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

`send Method (ServerXMLHTTP/IServerXMLHTTPRequest)` | `getResponseHeader Method (ServerXMLHTTP/IServerXMLHTTPRequest)` | `setRequestHeader Method (ServerXMLHTTP/IServerXMLHTTPRequest)`

Applies to: `IServerXMLHTTPRequest/ServerXMLHTTP`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**getAttribute Method**

Gets the value of the attribute.

[Script]

**Script Syntax**

```javascript
objValue = oXMLDOMElement.getAttribute(name);
```

**Parameters**

*name*

A string specifying the name of the attribute to return.

**Return Value**

A variant. Returns the value as a string if the attribute value is a non-empty string. Returns Null if the named attribute does not have a specified value, or an implied default value, such as one taken from a DTD or schema.

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, sIdValue;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    sIdValue = nodeBook.getAttribute("id")
    alert(sIdValue);
}
```

[Visual Basic]
**Visual Basic Syntax**

```
objValue = oXMLDOMElement.getAttribute(name)
```

**Parameters**

*name*

A string specifying the name of the attribute to return.

**Return Value**

A variant. Returns the value as a string if the attribute value is a non-empty string. Returns Null if the named attribute does not have a specified value, or an implied default value, such as one taken from a DTD or schema.

**Example**

```
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim sIdValue As String
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    sIdValue = nodeBook.getAttribute("id")
    MsgBox sIdValue
End If
```

**[C/C++]**

**C/C++ Syntax**

```
HRESULT getAttribute(
    BSTR name,
    VARIANT *value);
```

**Parameters**
**name [in]**
The name of the attribute to return.

**value [out, retval]**
The string that contains the attribute value. The empty string is returned if the named attribute does not have a default value specified.

### C/C++ Return Values

**S_OK**
The value returned if successful.

**S_FALSE**
The value when no attribute with the given name is found.

**E_INVALIDARG**
The value returned if the name parameter is Null.

### C/C++ Example

```cpp
BOOL DOMElementAttribute() 
{
    BOOL bResult = FALSE;
    _variant_t varValue;
    BSTR bstrAttributeName = ::SysAllocString(_T("dateCreated"));
    IXMLDOMDocument *pIXMLDOMDocument = NULL;
    IXMLDOMElement *pIXMLDOMElement = NULL;
    HRESULT hr;

    try
    {
        // Create an instance of DOMDocument and initialize
        // pIXMLDOMDocument.
        // Load/create an XML fragment.
        hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXMLDOMElement)
        {
            varValue = _T("year 2000");
            hr = pIXMLDOMElement->setAttribute(bstrAttributeName, varValue);
            SUCCEEDED(hr) ? 0 : throw hr;

            hr = pIXMLDOMElement->getAttribute(bstrAttributeName, &varValue);
            SUCCEEDED(hr) ? 0 : throw hr;
            if(varValue.vt != VT_NULL)
            {
```
::MessageBox(NULL, _bstr_t(varValue), bstrAttributeName, MB_OK);

}  
bResult = TRUE;
::SysFreeString(bstrAttributeName);
bstrAttributeName = NULL;
pIXMLDOMElement->Release();

}
catch(...)
{
    if(bstrAttributeName)
        ::SysFreeString(bstrAttributeName);
    if(pIXMLDOMElement)
        pIXMLDOMElement->Release();
    DisplayErrorToUser();

    return bResult;
}

Remarks

You can also retrieve attributes by using the getNamedItem method of the IXMLDOMNamedNodeMap.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ▼ in the upper-left corner of the page.

See Also

getNamedItem Method | IXMLDOMNamedNodeMap

Applies to: IXMLDOMElement
getAttributeNode Method

Gets the attribute node.

[Script]

Script Syntax

var objXMLDOMAttribute = oXMLDOMElement.getAttributeNode(name);

Parameters

name
   The string specifying the name of the attribute to be retrieved.

Return Value

An object. Returns IXMLDOMAttribute with the supplied name, or Null if the named attribute cannot be found on this element.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodeId, sIdValue;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
   var myErr = xmlDoc.parseError;
   alert("You have error " + myErr.reason);
} else {
   nodeBook = xmlDoc.selectSingleNode("//book");
   nodeId = nodeBook.getAttributeNode("id");
   sIdValue = nodeId.value;
   alert(sIdValue);
}

[Visual Basic]
Visual Basic Syntax

Set objXMLDOMAttribute = oXMLDOMElement.getAttributeNode(name)

Parameters

name
   The string specifying the name of the attribute to be retrieved.

Return Value

An object. Returns IXMLDOMAttribute with the supplied name, or Null if the
named attribute cannot be found on this element.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim nodeId As IXMLDOMAttribute
Dim sIdValue As String
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
   Dim myErr
   Set myErr = xmlDoc.parseError
   MsgBox("You have error " & myErr.reason)
Else
   Set nodeBook = xmlDoc.selectSingleNode("//book")
   Set nodeId = nodeBook.getAttributeNode("id")
   sIdValue = nodeId.Value
   MsgBox sIdValue
End If

[C/C++]

C/C++ Syntax

HRESULT getAttributeNode(
   BSTR name,
   IXMLDOMAttribute **attributeNode);

Parameters
name [in]
The name of the attribute to be retrieved.

attributeNode [out, retval]
An IXMLDOMAttribute object that is returned with the supplied name, or Null if the named attribute cannot be found on this element.

C/C++ Return Values

S_OK
The value returned if successful.

S_FALSE
The value returned when no attribute with the given name is found.

E_INVALIDARG
The value returned if name is Null.

C/C++ Example

BOOL DOMElementAttributeNode()
{
    BOOL bResult = FALSE;
    BSTR bstrAttributeName = ::SysAllocString(_T("dateCreated"));
    IXMLDOMAttribute* pIXMLDOMAttribute = NULL;
    IXMLDOMElement* pIXMLDOMElement = NULL;
    IXMLDOMDocument* pIXMLDOMDocument = NULL;
    HRESULT hr;

    try
    {
        // Create an instance of DOMDocument and initialize
        // pIXMLDOMDocument.
        // Load/create an XML fragment.
        hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
        SUCCEEDED(hr) ? 0 : throw hr;

        If(pIXMLDOMElement)
        {
            hr = pIXMLDOMElement->getAttributeNode(bstrAttributeName, &
            if(SUCCEEDED(hr) && pIXMLDOMAttribute)
            {
                // Read node value and display . . .
                bResult = TRUE;
                pIXMLDOMAttribute->Release();
                pIXMLDOMAttribute = NULL;
            }
        }
    }
}
::SysFreeString(bstrAttributeName);
bstrAttributeName = NULL;
pIXMLDOMElement->Release();
pIXMLDOMElement = NULL;
}
}
catch(...)
{
    if(bstrAttributeName)
        ::SysFreeString(bstrAttributeName);
    if(pIXMLDOMAttribute)
        pIXMLDOMAttribute->Release();
    if(pIXMLDOMElement)
        pIXMLDOMElement->Release();
    DisplayErrorToUser();
}
return bResult;
}

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMAttribute

Applies to: IXMLDOMElement
getDeclaration Method

Returns an ISchemaItem object. This object is the declaration of the DOM node that is sent to the function. The declaration specifies the schema information that is used to validate the document item in the DOM node. The ISchemaItem interface can be used to obtain further information about the schema declaration stored in the ISchemaItem object.

[Script]

Implementation Syntax

var oSchemaItem = oDom.namespaces.getDeclaration(oDOMNode);

Parameters

oDOMNode
   An object. The DOM node for which a declaration object will be returned.

Return Values

oSchemaItem
   An object. The schema item object that describes the specified DOM node, passed in the node parameter.

Code Example

The following VBScript example shows the getDeclaration method in a schema.

' Load the doc.
Set oDoc = CreateObject("Msxml2.DOMDocument.5.0")
oDoc.async = false
oDoc.load "po2.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
   Dim myErr
   Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
    ' Retrieve the namespace URI for schema used in
    ' purchase order document.
    Set oDecl = oDoc.namespaces.getDeclaration(oDoc.documentElement)
    WScript.Echo oDecl.namespaceURI
End If

[Visual Basic]

Implementation Syntax

Set oSchemaItem = oDom.namespaces.getDeclaration(oDOMNode)

Parameters

oDOMNode
    An object. The DOM node for which a declaration object will be returned.

Return Values

oSchemaItem
    An object. The schema item object for the specified DOM node, passed in
    the node parameter.

[C/C++]

Syntax

HRESULT getDeclaration(IXMLDOMNode* node, ISchemaItem** item);

Parameters

node [in]
    An object. The DOM node.
item [out,retval]
    An object. The schema object for the specified DOM.

Return Values
S_OK
The value returned if successful.

E_INVALIDARG
The value returned if the DOM node points to some object other than a Node or NULL.

E_POINTER
The value returned if the item object is NULL

E_FAIL
The value returned if the node does not have a corresponding declaration in the schema.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note**  The syntax oSchemaCollection.getDeclaration(oDOMNODE) is no longer supported and will return E_NOTIMPL.

**See Also**

**DOM Node Interface**

Applies to: **IXMLDOMSchemaCollection2**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getElementsByTagName Method (DOMDocument)

Returns a collection of elements that have the specified name.

[Script]

Script Syntax

```
var objXMLDOMNodeList = 
    oXMLDOMDocument.getElementsByTagName(tagName);
```

Parameters

tagName

A string specifying the element name to find. The tagName value "*" returns all elements in the document.

Return Value

An object. Points to a collection of elements that match the specified name.

Example

The following script example creates an IXMLDOMNodeList object using the DOMDocument object's getElementsByTagName method, and then displays all of the elements with the desired tag name.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    objNodeList = xmlDoc.getElementsByTagName("author");
```
for (var i=0; i<objNodeList.length; i++) {
    alert(objNodeList.item(i).xml);
}

[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNodeList =
oXMLDOMDocument.getElementsByTagName(tagName)

**Parameters**

*tagName*

A string specifying the element name to find. The `tagName` value "*" returns all elements in the document.

**Return Value**

An object. Points to a collection of elements that match the specified name.

**Example**

The following Microsoft® Visual Basic® example creates an `IXMLDOMNodeList` object using the `DOMDocument` object's `getElementsByTagName` method, and then displays all of the elements with the desired tag name.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLDOMNodeList
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set objNodeList = xmlDoc.getElementsByTagName("author")
    For i = 0 To (objNodeList.length - 1)
        MsgBox (objNodeList.Item(i).xml)
    Next
End If
```
[C/C++]

C/C++ Syntax

HRESULT getElementsByTagName(
    BSTR tagName,
    IXMLDOMNodeList **resultList);

Parameters

tagName [in]
    The element name to find. The tagName value"*" returns all elements in the
document.
resultList [out, retval]
    The address of a collection of elements that match the specified name.

C/C++ Return Values

S_OK
    The value returned if successful.

Example

IXMLDOMDocument *pIXMLDOMDocument = NULL;
wstring strFindText (_T("author"));
IXMLDOMNodeList *pIDOMNodeList = NULL;
IXMLDOMNode *pIDOMNode = NULL;
long value;
BSTR bstrItemText;
HRESULT hr;

try
{
    // Initialize pIXMLDOMDocument (create a DOMDocument).
    // Load document.
    hr = pIXMLDOMDocument->getElementsByTagName(
        (TCHAR*)strFindText.data(), &pIDOMNodeList);
    SUCCEEDED(hr) ? 0 : throw hr;

    hr = pIDOMNodeList->get_length(&value);
    if(SUCCEEDED(hr))
    {

pIDOMNodeList->reset();
for(int ii = 0; ii < value; ii++)
{
    pIDOMNodeList->get_item(ii, &pIDOMNode);
    if(pIDOMNode )
    {
        pIDOMNode->get_text(&bstrItemText);
        ::MessageBox(NULL, bstrItemText, strFindText.data(), MB_OK);
        pIDOMNode->Release();
        pIDOMNode = NULL;
    }
}
pIDOMNodeList->Release();
pIDOMNodeList = NULL;
}
catch(...)
{
    if(pIDOMNodeList)
        pIDOMNodeList->Release();
    if(pIDOMNode)
        pIDOMNode->Release();
    DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.

**Remarks**

The elements in the collection are returned in the order in which they would be encountered in a preorder traversal of the document tree. In a preorder traversal, the parent root node is visited first and each child node from left to right is then traversed.

The returned IXMLDOMNodeList object is live and immediately reflects changes to the nodes that appear in the list.

The `getElementsByTagName` method simulates the matching of the provided argument against the result of the `tagName` property of `IXMLDOMElement`. When executed, it does not recognize or support namespaces. Instead you should use the `selectNodes` method, which is faster in some cases and can support more complex searches.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

IXMLDOMNodeList | selectNodes Method

Applies to: DOMDocument
**getElementsByTagName Method (IXMLDOMElement)**

Returns a list of all descendant elements that match the supplied name.

**[Script]**

**Script Syntax**

```javascript
var objXMLDOMNodeList = oXMLDOMElement.getElementsByTagName(tagName);
```

**Parameters**

*tagName*

A string specifying the name of the element to find. The `tagName` value "*" matches all descendant elements of this element.

**Return Value**

An object. Returns `IXMLDOMNodeList` object containing all elements that match the supplied name.

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodelistAuthor;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    nodelistAuthor = nodeBook.getElementsByTagName("author");
    alert(nodelistAuthor.length);
}
```
[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNodeList = oXMLDOMElement.getElementsByTagName(tagName)

Parameters

tagName
A string specifying the name of the element to find. The tagName value "*" matches all descendant elements of this element.

Return Value

An object. Returns IXMLDOMNodeList object containing all elements that match the supplied name.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim nodeListAuthor As IXMLDOMNodeList
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    Set nodeListAuthor = nodeBook.getElementsByTagName("author")
    MsgBox nodeListAuthor.length
End If

[C/C++]

C/C++ Syntax

HRESULT getElementsByTagName(
    BSTR tagName,
    IXMLDOMNodeList **resultList);
Parameters

tagName [in]
The name of the element to find. The tagName value "*" matches all
descendant elements of this element.

resultList [out, retval]
An IXMLDOMNodeList object containing all elements that match the supplied
name.

C/C++ Return Values

S_OK
The value returned if successful.

C/C++ Example

// Find all Nodes with a particular tag.
BOOL DOMDocFindNodesWithTag()
{
    BOOL bResult = FALSE;
    CString strFindText(_T("author"));
    IXMLDOMNodeList *pIDOMNodeList = NULL;
    IXMLDOMNode *pIDOMNode = NULL;
    long value = 0;
    BSTR bstrItemText = NULL;
    HRESULT hr;

    try
    {
        // Create the DOMDocument and initialise m_pIXMLDOMDocument2.
        // Find all "author" elements.
        hr = m_pIXMLDOMDocument2->getElementsByTagName(0,
            (TCHAR *)strFindText.GetBuffer(0),
            &pIDOMNodeList);
        SUCCEEDED(hr) ? 0 : throw hr;

        // Get the length of the list returned by the previous find.
        hr = pIDOMNodeList->get_length(&value);
        if(SUCCEEDED(hr))
        {
            pIDOMNodeList->reset();
            // Loop through the elements found and display the contents.
            for(int ii = 0; ii < value; ii++)
            {
            
        
        
}
hr = pIDOMNodeList->get_item(ii, &pIDOMNode);
SUCCEEDED(hr) ? 0 : throw hr;
if(pIDOMNode)
{
    hr = pIDOMNode->get_text(&bstrItemText);
    SUCCEEDED(hr) ? 0 : throw hr;
    if(bstrItemText)
    {
        bResult = TRUE;
        ::MessageBox(NULL, bstrItemText, strFindText, MB_OK);
        ::SysFreeString(bstrItemText);
        bstrItemText = NULL;
    }
    pIDOMNode->Release();
    pIDOMNode = NULL;
}
if(pIDOMNodeList)
pIDOMNodeList->Release();
pIDOMNodeList = NULL;
}

catch(...)
{
    if(pIDOMNodeList)
pIDOMNodeList->Release();
    if(pIDOMNode)
pIDOMNode->Release();
    if(bstrItemText)
        ::SysFreeString(bstrItemText);
    bResult = FALSE;
    DisplayErrorToUser();
}
return bResult;

Remarks

Elements appear in the order encountered in a preorder traversal of this element’s tree.

The IXMLDOMNodeList object is returned even if there are no matches. In this case, the length of the list will be set to zero.

The IXMLDOMNodeList is live and immediately reflects changes to the nodes that appear in the list.
The `getElementsByTagName` method simulates the matching of the provided argument against the result of the `tagName` property of `IXMLDOMElement`. When executed, it does not recognize or support namespaces. Instead you should use the `selectNodes` method, which is faster in some cases and can support more complex searches.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`IXMLDOMNodeList`

Applies to: `IXMLDOMElement`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getNamedItem Method

Retrieves the attribute with the specified name.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNamedNodeMap.getNamedItem(name);
```

Parameters

name
A string specifying the name of the attribute.

Return Value

An object. Returns IXMLDOMNode object for the specified attribute. Returns Nothing if the attribute node is not in this collection.

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodeId;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    nodeId = nodeBook.attributes.getNamedItem("id");
    alert(nodeId.value);
}
```

[Visual Basic]

Visual Basic Syntax
Set objXMLDOMNode = oXMLDOMNamedNodeMap.getNamedItem(name)

Parameters

name
A string specifying the name of the attribute.

Return Value

An object. Returns an IXMLDOMNode object for the specified attribute. Returns Nothing if the attribute node is not in this collection.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim nodeId As IXMLDOMAttribute
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    If TypeName(nodeBook.Attributes.getNamedItem("id")) <> "Nothing"
        Set nodeId = nodeBook.Attributes.getNamedItem("id")
        MsgBox(nodeId.Value)
    Else
        'No @id attribute in attributes collection.
        MsgBox "Could not find the @id attribute as " & _
            in a <book id=''> element."
    End If
End If

[C/C++]

C/C++ Syntax

HRESULT getNamedItem(  
    BSTR name,  
    IXMLDOMNode **namedItem);
Parameters

*name* [in]
  The name of the attribute.

*namedItem* [out, retval]
  An IXMLDOMNode object for the specified attribute. Returns Null if the attribute node is not in this collection.

C/C++ Return Values

S_OK
  The value returned if successful.

S_FALSE
  The value when returning Null.

E_INVALIDARG
  The value returned if the namedItem parameter is Null.

C/C++ Example

```c
IXMLDOMNode *pIXMLDOMNode = NULL;
IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMap = NULL;
BSTR bstrAttributeName = ::SysAllocString(_T("dateModified"));
IXMLDOMElement *pIXMLDOMElement = NULL;
IXMLDOMDocument *pIXMLDOMDocument = NULL;
VARIANT varValue;
try
{
  // Create an instance of DOMDocument and initialize pIXMLDOMDocum
  // Load/create an XML fragment.
  hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
  SUCCEEDED(hr) ? 0 : throw hr;

  if(pIXMLDOMElement)
  {
    hr = pIXMLDOMElement->get_attributes(&pIXMLDOMNamedNodeMap);
    if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMap)
    {
      hr = pIXMLDOMNamedNodeMap->getNamedItem(bstrAttribute
```
pIXMLDOMNode->Release();
pIXMLDOMNode = NULL;
}
pIXMLDOMNamedNodeMap->Release();
pIXMLDOMNamedNodeMap = NULL;
}
pIXMLDOMElement->Release();
pIXMLDOMElement = NULL;
}
::SysFreeString(bstrAttributeName);
bstrAttributeName = NULL;
}
catch(...)
{
  if(bstrAttributeName)
    ::SysFreeString(bstrAttributeName);
  if(pIXMLDOMElement)
    pIXMLDOMElement->Release();
  if(pIXMLDOMNamedNodeMap)
    pIXMLDOMNamedNodeMap->Release();
  if(pIXMLDOMNode)
    pIXMLDOMNode->Release();
  DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode

Applies to: IXMLDOMNamedNodeMap
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getOption Method

Returns the value of one of the following options:

- SXH_OPTION_URL (Read-only.)
- SXH_OPTION_URL_CODEPAGE
- SXH_OPTION_ESCAPE_PERCENT_IN_URL
- SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS
- SXH_OPTION_SELECT_CLIENT_SSL_CERT

For more information about these options, see Remarks.

[Script]

Script Syntax

```javascript
varValue = oServerXMLHTTPRequest.getOption(option);
```

Parameters

option
   The option whose value is to be returned.

[Visual Basic]

Visual Basic Syntax

```vbnet
varValue = oServerXMLHTTPRequest.getOption(option)
```

option
   The option whose value is to be returned.

[C/C++]

C/C++ Syntax
HRESULT getOption(SERVERXMLHTTP_OPTION option, VARIANT * value);

**Parameters**

*option* [in]

The option whose value is to be returned.

*value* [out, retval]

The return value of the option.

**C/C++ Return Values**

S_OK

Value returned if successful.

**Remarks**

The following table lists the options available for the `getOption` method.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXH_OPTION_URL</td>
<td>By default, returns a variant of type string that contains the URL of the resource. Read-only.</td>
</tr>
<tr>
<td>-1</td>
<td>The <code>SXH_OPTION_URL</code> option allows the client to learn the final URL last used. For example, after a call made using the <code>send</code> method has complete you can use this option to check the URL and see if any redirection occurred. The URL, however, cannot be read until the <code>open</code> method is ne: called.</td>
</tr>
<tr>
<td>SXH_OPTION_URL_CODEPAGE</td>
<td>By default, CP_UTF8 is the code page used to convert the Unicode URL string (specified in the <code>open</code> method) to a single-byte representation.</td>
</tr>
<tr>
<td>0</td>
<td>The <code>SXH_OPTION_URL_CODEPAGE</code> option allows the client to override this default with a different code page value. The client should specify an unsigned integer value for the code page.</td>
</tr>
</tbody>
</table>
SXH_OPTION_ESCAPE_PERCENT_IN_URL

By default, escaping unsafe ANSI characters in the URL (for example, " > "%20") does not escape the % character itself.

The SXH_OPTION_ESCAPE_PERCENT_IN_URL option allows the client to change this behavior. The client should specify a Boolean True/False value for this option.

SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS

The SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS option is a DWORD mask of various flags that can be set to change this default behavior. The default value is to ignore all problems. You must set this option before calling the send method. The flags are as follows:

 SXH_SERVER_CERT_IGNORE_UNKNOWN_CA = 256

Unknown certificate authority

 SXH_SERVER_CERT_IGNORE_WRONG_USAGE = 512

Malformed certificate such as a certificate with no subject name.

 SXH_SERVER_CERT_IGNORE_CERT_CN_INVALID = 4096

Mismatch between the visited hostname and the certificate name being used on the server.

 SXH_SERVER_CERT_IGNORE_CERT_DATE_INVALID = 8192

The date in the certificate is invalid or has expired.

 SXH_SERVER_CERT_IGNORE_ALL_SERVER_ERRORS = 13056

All certificate errors.

To turn off a flag, you subtract it from the default value, which is the sum of all flags. For example, to catch an invalid date in a certificate, you turn off the SXH_SERVER_CERT_IGNORE_CERT_DATE_INVALID flag as follows:

 shx.setOption(2) = (shx.getOption(2) _
 - SXH_SERVER_CERT_IGNORE_CERT_DATE_INVALID)
**SXH_OPTION_SELECT_CLIENT_SSL_CERT**

By default, the value of this option is an empty string (""), which means pick the first certificate in the local store to send if the server requests a client certificate.

The **SXH_OPTION_SELECT_CLIENT_SSL_CERT** option is a string that lets you select which client certificate from the local store should be sent. You must set this option before calling the send method. The following example sets the client certificate option to request the client certificate named "MSXML":

```plaintext
certName = shx.getOption(3) = "MSXML"
```

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- [open Method (ServerXMLHTTP/IServerXMLHTTPRequest)](open)
- [setOption Method](setOption)

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getProperty Method (IXMLDOMDocument2)

Retrieves the value of one of the second-level properties that are set either by default or using the setProperty method.

[Script]

Script Syntax

strPropValue = objXMLDOMDocument2.getProperty(name);

Parameters

name

The string name of the property. This name is case-sensitive.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.setProperty("SelectionLanguage", "XPath");
alert(xmlDoc.getProperty("SelectionLanguage"));  

[Visual Basic]

Visual Basic Syntax

strPropValue = objXMLDOMDocument2.getProperty(name)

Parameters

name

The string name of the property. This name is case-sensitive.

Example
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.setProperty "SelectionLanguage", "XPath"
MsgBox xmlDoc.getProperty("SelectionLanguage")

[C/C++]

C/C++ Syntax

HRESULT getProperty(BSTR name, VARIANT* value);

Parameters

name [in]
The string name of the property. This name is case-sensitive.
value [out, retval]
The variant return value of the requested flag.

C/C++ Return Values

S_OK
The value returned if successful.
E_FAIL
The value returned if property name is invalid.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Second-level properties | setProperty Method

Applies to: IXMLDOMDocument2
getProperty Method
(IXMLDOMSelection)

Returns a property.

[Script]

Script Syntax

strValue = objXMLDOMSelection.getProperty(name);

Parameters

name

The string name of the property. This name is case-sensitive.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oSelection;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    oSelection = xmlDoc.selectNodes("//book");
    alert(oSelection.getProperty("SelectionLanguage"));
}

[Visual Basic]

Visual Basic Syntax

strValue = objXMLDOMSelection.getProperty(name)
Parameters

name
The string name of the property. This name is case-sensitive.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim oSelection As IXMLDOMSelection
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set oSelection = xmlDoc.selectNodes("//book")
    MsgBox oSelection.getProperty("SelectionLanguage")
End If

[C/C++]

C/C++ Syntax

HRESULT getProperty(BSTR name, VARIANT* value);

Parameters

name [in]
The string name of the property. This name is case-sensitive.
value [out, retval]
The variant return value of the requested property.

C/C++ Return Values

S_OK
The value returned if method successful.
E_INVALIDARG
The value returned if named property does not exist.
Remarks

The `getProperty` method returns the value for the internal `SelectionLanguage` property (flag) that was set by calling the `setProperty` method on the document or the default value.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

`setProperty Method`

Applies to: `IXMLDOMSelection`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getQualifiedItem Method

Returns the attribute with the specified namespace and attribute name.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNamedNodeMap.getQualifiedItem(baseName, namespaceURI);
```

Parameters

- **baseName**
  The string specifying the base name of the attribute, without namespace qualification.
- **namespaceURI**
  The string specifying the namespace prefix that qualifies the attribute name.

Return Value

An object. Returns the attribute node specified by the `baseName` and `namespaceURI` parameters. Returns Null if the attribute is not in the collection or if the item is not an attribute.

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oNamedNodeMap, nodeBook;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
oNamedNodeMap = nodeBook.attributes;
alert(oNamedNodeMap.getQualifiedItem("id", ").value);
}
```
[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNode = oXMLDOMNamedNodeMap.getQualifiedItem(baseName, n

**Parameters**

*baseName*

The string specifying the base name of the attribute, without namespace qualification.

*namespaceURI*

The string specifying the namespace prefix that qualifies the attribute name.

**Return Value**

An object. Returns the attribute node specified by the baseName and namespaceURI parameters. Returns Null if the attribute is not in the collection or if the item is not an attribute.

**Example**

Dim xmlDoc As New Msxml2.DOMDocument50
Dim oNamedNodeMap As IXMLDOMNamedNodeMap
Dim nodeBook As IXMLDOMElement
Dim nodeId As IXMLDOMAttribute
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    Set oNamedNodeMap = nodeBook.Attributes
    Set nodeId = oNamedNodeMap.getQualifiedItem("id", ")
    MsgBox nodeId.Value
End If

[C/C++]
C/C++ Syntax

HRESULT getQualifiedItem(
    BSTR baseName,
    BSTR namespaceURI,
    IXMLDOMNode **qualifiedItem);

Parameters

baseName [in]
    The base name of the attribute, without namespace qualification.

namespaceURI [in]
    The namespace prefix that qualifies the attribute name.

qualifiedItem [out, retval]
    The attribute node specified by the baseName and namespaceURI parameters. Returns Null if the attribute is not in the collection or if the item is not an attribute.

C/C++ Return Values

S_OK
    The value returned if successful.

S_FALSE
    The value when returning Null.

E_INVALIDARG
    The value returned if qualifiedItem is Null.

C/C++ Example

IXMLDOMNode *pIXMLDOMNode = NULL;
IXMLDOMNodeNamedNodeMap *pIXMLDOMNodeNamedNodeMap = NULL;
BSTR bstrAttributeName = ::SysAllocString(_T("dateModified"));
IXMLDOMElement *pIXMLDOMElement = NULL;
VARIANT varValue;
IXMLDOMDocument *pIXMLDOMDocument = NULL;
HRESULT hr;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocum
    // Load/create an XML fragment.
hr = pXMLDOMDocument->get_documentElement(&pIXMLDOMElem);  
SUCCEEDED(hr) ? 0 : throw hr;

if(pIXMLDOMElem)  
{
    hr = pIXMLDOMElem->get_attributes(&pIXMLDOMNamedNodeMap);  
    if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMap)
    {
        hr = pIXMLDOMNamedNodeMap->getQualifiedItem(bstrAttributeName, NULL, &pIXMLDOMNode);  
        if(SUCCEEDED(hr) && pIXMLDOMNode)
        {
            pIXMLDOMNode->get_nodeValue(&varValue);  
            ::MessageBox(NULL, _bstr_t(varValue), _T("Item Value"),  
            pIXMLDOMNode->Release();  
            pIXMLDOMNode = NULL;  
        }
        pIXMLDOMNamedNodeMap->Release();  
        pIXMLDOMNamedNodeMap = NULL;
    }
    pIXMLDOMElem->Release();  
    pIXMLDOMElem = NULL;
}
::SysFreeString(bstrAttributeName);  
bstrAttributeName = NULL;
}
catch(...)
{
    if(bstrAttributeName)
        ::SysFreeString(bstrAttributeName);  
    if(pIXMLDOMElem)
        pIXMLDOMElem->Release();  
    if(pIXMLDOMNamedNodeMap)
        pIXMLDOMNamedNodeMap->Release();  
    if(pIXMLDOMNode)
        pIXMLDOMNode->Release();  
    DisplayErrorToUser();
}

// Release pXMLDOMDocument when finished with it.

Remarks

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

Applies to: IXMLDOMNamedNodeMap
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
getResponseHeader Method (IXMLHTTPRequest)

Retrieves the value of an HTTP header from the response body.

[Script]

Script Syntax

strValue = oXMLHttpRequest.getResponseHeader(bstrHeader);

Parameters

bstrHeader
A string containing the case-insensitive header name.

Return Value

A string. Contains the resulting header information.

Example

var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmhttp.open("GET", "http://localhost/sample.xml", false);
xmhttp.send();
alert(xmlhttp.getResponseHeader("Content-Length"));

[Visual Basic]

Visual Basic Syntax

strValue = oXMLHttpRequest.getResponseHeader(bstrHeader)

Parameters
**bstrHeader**

A string containing the case-insensitive header name.

**Return Value**

A string. Contains the resulting header information.

**Example**

```vbs
Dim xmlhttp As New Msxml2.XMLHTTP50
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.getResponseHeader("Content-Length")
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT getResponseHeader(BSTR bstrHeader, BSTR* pbstrValue);
```

**Parameters**

- **bstrHeader [in]**
  A case-insensitive header name.
- **pbstrValue [out, retval]**
  The resulting header information.

**C/C++ Return Values**

- **S_OK**
  The value returned if successful.

**C/C++ Example**

```c
HRESULT hr;
BSTR bstrValue = NULL;
IXMLHttpRequest *pIXMLHttpRequest = NULL;
try
```
{
    // Create XMLHttpRequest object and initialize pIXMLHttpRequest.
    hr = pIXMLHttpRequest->getResponseHeader(_T("Server"), &bstrValue);
    if(SUCCEEDED(hr))
    {
        ::MessageBox(NULL, m_bstrValue, _T("Response Header-Server"),
                    ::SysFreeString(bstrValue);
        bstrValue = NULL;
    }
} catch(...)
{
    if(bstrValue)
        ::SysFreeString(bstrValue);
    DisplayErrorToUser();
}
// Release pIXMLHttpRequest when finished with it.

Remarks

The results of this method are valid only after the send method has been
successfully completed. The line, xmlhttp.getResponseHeader("Content-
Type"), returns the string "text/xml", assuming the server set "text/xml" as
the content type. The full list of header variables you can query can be accessed
from the getAllResponseHeaders method.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button ☐ in the upper-left corner of the page.

See Also

getAllResponseHeaders Method (IXMLHTTPRequest) | send Method
(IXMLHTTPRequest) | setRequestHeader Method (IXMLHTTPRequest)

Applies to: IXMLHTTPRequest
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getResponseHeader Method
(ServerXMLHTTP/IServerXMLHTTP)

Retrieves the value of an HTTP header from the response body.

[Script]

Script Syntax

strValue = oServerXMLHTTPRequest.getResponseHeader(bstrHeader);

Parameters

bstrHeader
A case-insensitive header name.

Example

var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.open("GET", "http://localhost/sample.xml", false);
xmlServerHttp.send();
alert(xmlServerHttp.getResponseHeader("Content-Length"));

[Visual Basic]

Visual Basic Syntax

strValue = oServerXMLHTTPRequest.getResponseHeader(bstrHeader)

Parameters

bstrHeader
A case-insensitive header name.

Example
Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.getResponseHeader("Content-Length")

[C/C++]

**C/C++ Syntax**

HRESULT getResponseHeader(BSTR bstrHeader, BSTR* pbstrValue);

**Parameters**

*bstrHeader* [in]
A case-insensitive header name.

*pbstrValue* [out, retval]
The resulting header information.

**C/C++ Return Values**

S_OK
The value returned if successful.

**Remarks**

The results of this method are valid only after the *send* method has been successfully completed. The line,

```
oServerXMLHTTPRequest.getResponseHeader("Content-Type");
```
returns the string "text/xml", assuming the server set "text/xml" as the content type.

The full list of header variables you can query can be accessed from the *getAllResponseHeaders* method.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[getAllResponseHeaders Method(ServerXMLHTTP/IServerXMLHTTPRequest)]
| send Method (ServerXMLHTTP/IServerXMLHTTPRequest) |
| setRequestHeader Method (ServerXMLHTTP/IServerXMLHTTPRequest) |

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
getSchema Method

Returns an ISchema object. The schema contains the namespace URI specified in the namespaceURI parameter that is passed to this method. The ISchema interface can be used to further obtain information about the schema object that is returned.

The following example shows the getSchema method being used to return a schema object.

Dim oSchemaCache as New IXMLDOMSchemaCollection2
Dim oSchema as ISchema
Dim nsTarget as String
nsTarget = "http://www.example.microsoft.com"
oSchemaCache.add nsTarget, PO.xsd
Set oSchema = oSchemaCache.getSchema(nsTarget)

[Script]

Implementation Syntax

var oSchema = oSchemaCache.getSchema(strNamespaceURI);

Parameters

strNamespaceURI
A string. The namespace of the schema to be retrieved from the schema cache.

Return Values

oSchema
An object. The schema object for the Namespace URI sent to the getSchema method.

[Visual Basic]
**Implementation Syntax**

Set `oSchema = oSchemaCache.getSchema(strNamespaceURI)`

**Parameters**

`strNamespaceURI`
A string. The namespace of the schema to be retrieved from the schema cache.

**Return Values**

`oSchema`
An object. The schema object for the specified schema.

[C/C++]

**Syntax**

```c
HRESULT getSchema(BSTR namespaceURI, ISchema** schema);
```

**Parameters**

`strNamespaceURI [in]`
A string. The namespace of the schema to be retrieved from the schema cache.

`oSchema [out,retval]`
An object. The schema object for the specified schema.

**Return Values**

S_OK
The value returned if successful.

E_POINTER
The value returned if the schema is NULL

E_FAIL
The value returned if another problem is detected.
To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchema interface | Using Namespaces in Schemas

Applies to: IXMLDOMSchemaCollection2
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
hasChildNodes Method

Provides a fast way to determine whether a node has children.

[Script]

Script Syntax

boolValue = oXMLDOMNode.hasChildNodes();

Return Value

Boolean. Returns True if this node has children.

Example

The following script example checks a node to see if it has any child nodes. If it does, it displays the number of child nodes it contains.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    currNode = xmlDoc.documentElement.firstChild;
    if (currNode.hasChildNodes()) {
        alert(currNode.childNodes.length);
    } else {
        alert("no child nodes");
    }
}
```

[Visual Basic]

Visual Basic Syntax
boolValue = oXMLDOMNode.hasChildNodes

**Return Value**

Boolean. Returns True if this node has children.

**Example**

The following Microsoft® Visual Basic® example checks a node to see if it has any child nodes. If it does, it displays the number of child nodes it contains.

```vba
Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set currNode = xmlDoc.documentElement.firstChild
    If currNode.hasChildNodes Then
        MsgBox currNode.childNodes.length
    Else
        MsgBox "no child nodes"
    End If
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT hasChildNodes(
    VARIANT_BOOL *hasChild);
```

**Parameters**

*hasChild [out,retval]*

Returns True if this node has children.

**C/C++ Return Values**
S_OK
    The value returned if successful.
S_FALSE
    The value returned when there are no children.
E_INVALIDARG
    The value returned if the hasChild parameter is Null.

Remarks

It always returns False for nodes that, by definition, cannot have children: the
IXMLDOMCDATASection, IXMLDOMComment, IXMLDOMNotation,
IXMLDOMProcessingInstruction, and IXMLDOMText nodes.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button [ ] in the upper-left corner of the page.

See Also

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection |
IXMLDOMCharacterData | IXMLDOMComment | DOMDocument |
IXMLDOMDocumentFragment | IXMLDOMDocumentType |
IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference |
IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction |
IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
hasFeature Method

Indicates support for the specified feature.

[Script]

Script Syntax

boolVal = objXMLDOMImplementation.hasFeature(feature, version);

Parameters

feature
A string that specifies the feature to test. In Level 1, valid feature values are "XML", "DOM", and "MS-DOM" (case-insensitive).

version
A string that specifies the version number to test, or if Null, tests for implementation of the feature in any version. In Level 1, "1.0" is the valid version value.

Return Value

Boolean. Returns True if the specified feature is implemented; otherwise False.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var bool = xmlDoc.implementation.hasFeature("DOM", "1.0");
alert(bool);

[Visual Basic]

Visual Basic Syntax

boolVal = objXMLDOMImplementation.hasFeature(feature, version)
Parameters

feature
   A string that specifies the feature to test. In Level 1, valid feature values are "XML", "DOM", and "MS-DOM" (case-insensitive).

version
   A string that specifies the version number to test, or if Null, tests for implementation of the feature in any version. In Level 1, "1.0" is the valid version value.

Return Value

Boolean. Returns True if the specified feature is implemented; otherwise False.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim bool As Boolean
bool = xmlDoc.implementation.hasFeature("DOM", "1.0")
MsgBox bool

[C/C++]

C/C++ Syntax

HRESULT hasFeature(
   BSTR feature,
   BSTR version,
   VARIANT_BOOL *hasFeature);

Parameters

feature [in]
   The feature to test. In Level 1, valid feature values are "XML", "DOM", and "MS-DOM" (case-insensitive).

version [in]
   The version number to test, or, if Null, tests for implementation of the feature in any version. In Level 1, "1.0" is the only valid version value.

hasFeature [out, retval]
True if the specified feature is implemented; otherwise False.

**C/C++ Return Values**

**S_OK**

The value returned if successful.

**C/C++ Example**

```c++
IXMLDOMImplementation *pIXMLDOMImplementation = NULL;
VARIANT_BOOL varbFlag;
BSTR bstrOutput = NULL;
BSTR bstrFeature = ::SysAllocString(_T("MS-DOM"));
HRESULT hr;
IXMLDOMDocument *pIXMLDOMDocument = NULL;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocument.
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_implementation(&pIXMLDOMImplementation);
    if(SUCCEEDED(hr) && pIXMLDOMImplementation)
    {
        hr = pIXMLDOMImplementation->hasFeature(bstrFeature, _T("1.0"));
        if(varbFlag == VARIANT_TRUE)
        {
            bstrOutput = ::SysAllocString(_T("Feature Supported"));
        }
        else
        {
            bstrOutput = ::SysAllocString(_T("Feature not Supported"));
        }
        ::MessageBox(NULL, bstrOutput, bstrFeature, MB_OK);
        ::SysFreeString(bstrOutput);
        bstrOutput = NULL;
    }
    catch(...)
    {
        if(bstrOutput)
        {
            ::SysFreeString(bstrOutput);
        }
        if(bstrFeature)
        {
            ::SysFreeString(bstrFeature);
        }
        if(pIXMLDOMImplementation)
        {
            pIXMLDOMImplementation->Release();
        }
    }
    ::SysFreeString(bstrFeature);
    bstrFeature = NULL;
    pIXMLDOMImplementation->Release();
}
```
DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished using it.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMImplementation
importNode Method

This method can be used to clone a node from a DOM object. In addition to supporting all the functionality offered by the cloneNode method, it enables cloning a node from DOM objects of different threading models. The clones created this way can be passed between such DOM objects. For example, a node from a free-threaded DOM object can be passed into an apartment-threaded DOM object. The resultant node object can then be added into the document using the appendChild method.

Note In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

[JavaScript]

Script Syntax

var clone = objXMLDOMDocument3.importNode(node, deep);

[Visual Basic]

Visual Basic Syntax

Set clone = objXMLDOMDocument3.importNode(node, deep);

[C/C++]

C/C++ Syntax Using Smart Pointers

IXMLDOMNodePtr clone = objXMLDOMDocument3->importNode(node, deep);

C/C++ Syntax

HRESULT IXMLDOMDocument3::importNode(
    ...[in] IXMLDOMNode* node,
    [in] VARIANT_BOOL deep,
    ...[out, retval] IXMLDOMNode** clone);
Parameters

node
An IXMLDOMNode object to be cloned.

deep
If true, any children of node will be cloned as well. If false, no children of node will be cloned.

clone
A clone of the node object, as specified above.

Return Values

S_OK
The node is cloned successfully.
E_FAIL
The node cannot be cloned properly and the resultant clone parameter is NULL.

Remarks

The importNode method does not allow for import of DOM nodes that are of one of the following enumerated node types: NODE_DOCUMENT, NODE_DOCUMENT_TYPE, NODE_ENTITY, and NODE_NOTATION.

In MSXML 5.0, deep must be VARIANT_TRUE, otherwise the method returns E_INVALIDARG.

Example

This example demonstrates how to use importNode to clone a node from a free-threaded DOM document and append the clone to a regular apartment-threaded DOM object. As a comparison, it also shows that a DOM object can clone a node from itself with importNode, just as it would with cloneNode.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (doc1.xml and doc2.xml)
- JScript Code (importNode.js)
- Visual Basic Code (importNode.frm)
- C/C++ Code (importNode.cpp)
- Output for importNode Example

Applies To

IXMLDOMDocument3

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode | IXMLDOMNodeList | IXMLDOMParseError | IXMLDOMSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Resource Files (doc1.xml and doc2.xml)

Use the following resource files for the **importNode** example.

**Input XML Data File (doc1.xml)**

```xml
<?xml version="1.0" encoding="utf-8"?>
<doc>
  <a class="1">doc1</a>
  <b class="1">author1</b>
</doc>
```

**Input XML Data File (doc2.xml)**

```xml
<?xml version="1.0" encoding="utf-8"?>
<doc>
  <a class="2">doc2</a>
  <b class="2">author2</b>
</doc>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
main();

function main()
{
  try {
    domFree = new ActiveXObject("MSXML2.FreeThreadedDOMDocument.5.0";
    domApt = new ActiveXObject("MSXML2.DOMDocument.5.0");
  }
  catch (e) {
    alert("Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office is not installed.
    Download and install MSXML 5.0 from http://msdn.microsoft.com before continuing.");
    return;
  }

  domApt.async = false;
  domApt.load("doc1.xml");

  domFree.async = false;
  domFree.load("doc2.xml");

  // Copy a node from domFree to domApt:
  // Fetch the "/doc" (node) from domFree (doc2.xml).
  // Clone node for import to domApt.
  // Append clone to domApt (doc1.xml).
  var node = domFree.selectSingleNode("/doc");
  var clone = domApt.importNode(node, true);
  domApt.documentElement.appendChild(clone);
  domApt.documentElement.appendChild(domApt.createTextNode("\n");
  alert("doc1.xml after importing /doc from doc2.xml:\n"+domApt.xml);

  // Clone a node using importNode() and append it to the same DOM:
  // Fetch the "doc/b" (node) from domApt (doc1.xml).
  // Clone node using importNode on domApt.
  // Append clone to domApt (doc1.xml).
  var node = domApt.selectSingleNode("doc/b");
  var clone = domApt.importNode(node, true);
  domApt.documentElement.appendChild(domApt.createTextNode("\t");
  domApt.documentElement.appendChild(domApt.createTextNode("\n");
  alert("doc1 after importing /doc/b from self:\n"+domApt.xml);

  // Clone a node and append it to the dom using cloneNode():
}
// Fetch "doc/a" (node) from domApt (doc1.xml).
// Clone node using cloneNode on domApt.
// Append clone to domApt (doc1.xml).

var node = domApt.selectSingleNode("doc/a");
var clone = node.cloneNode(true);
domApt.documentElement.appendChild(clone);
alert("doc1 after cloning a /doc/a from self:\n" + domApt.xml);

domApt.save('out.xml');
alert("a new document was saved to out.xml in the current working

}
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Private Sub Form_Load()

Dim domFree As FreeThreadedDOMDocument50
Dim domApt As DOMDocument50
Dim node As IXMLDOMNode
Dim clone As IXMLDOMNode
Dim msg As String

msg = ""
Set domFree = New FreeThreadedDOMDocument50
Set domApt = New DOMDocument50

domApt.async = False
If False = domApt.Load(App.Path + "\doc1.xml") Then
    MsgBox "can't load doc1.xml"
    Exit Sub
End If

domFree.async = False
If False = domFree.Load(App.Path + "\doc2.xml") Then
    MsgBox "can't load doc2.xml"
    Exit Sub
End If

' Copy a node from domFree to domApt:
'    Fetch the "/doc" (node) from domFree (doc2.xml).
'    Clone node for import to domApt.
'    Append clone to domApt (doc1.xml).
'
Set node = domFree.selectSingleNode("/doc")
Set clone = domApt.importNode(node, True)
domApt.documentElement.appendChild clone
msg = msg + "doc1.xml after importing /doc from doc2.xml:"
msg = msg + vbNewLine + domApt.xml + vbNewLine
Set node = Nothing
Set clone = Nothing

' Clone a node using importNode() and append it to self:
'    Fetch the "doc/b" (node) from domApt (doc1.xml).
'    Clone node using importNode() on domApt.
'    Append clone to domApt (doc1.xml).
'
Set node = domApt.selectSingleNode("/doc/b")


Set clone = domApt.importNode(node, True)
domApt.documentElement.appendChild(domApt.createTextNode(vbTab))
domApt.documentElement.appendChild clone
msg = msg + "doc1.xml after import /doc/b from self:"
msg = msg + vbNewLine + domApt.xml + vbNewLine
Set node = Nothing
Set clone = Nothing

' Clone a node and append it to the dom using cloneNode():
' Fetch "doc/a" (node) from domApt (doc1.xml).
' Append clone to domApt (doc1.xml).
' Set node = domApt.selectSingleNode("/doc/a")
Set clone = node.cloneNode(True)
domApt.documentElement.appendChild clone
msg = msg + "doc1.xml after child /doc/a from self:"
msg = msg + vbNewLine + domApt.xml + vbNewLine
Set node = Nothing
Set clone = Nothing

domApt.save App.Path + "\out.xml"
msg = msg + "a new document was saved to out.xml in the current work
MsgBox msg
End Sub

Try It!

1. Copy the first XML data (doc1.xml), and paste it into a text file. Save the file as doc1.xml.
2. Copy the second XML listing (doc2.xml), and paste it into a text file. Save the file as doc2.xml, in the same directory where you saved doc1.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as importNode.vbp to the same directory where you saved doc1.xml and doc2.xml. Name the form file importNode.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.
5. Copy the Visual Basic code listing above, and paste it into the form_load subroutine.
6. Execute the code by selecting Start from the Run menu.
7. Verify that the output is the same as that listed in Output for the importNode Example.
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

void main()
{
    IXMLDOMDocument3Ptr domFree=NULL;
    IXMLDOMDocument3Ptr domApt =NULL;
    IXMLDOMNodePtr node =NULL;
    IXMLDOMNodePtr clone =NULL;

    CoInitialize(NULL);

    if (FAILED(domFree.CreateInstance(__uuidof(FreeThreadedDOMDocument50))))
    {
        printf("Can't create domFree\n");
        return ;
    }

    if (FAILED(domApt.CreateInstance(__uuidof(DOMDocument50))))
    {
        printf("Can't create domApt\n");
        return ;
    }

    domApt->async = VARIANT_FALSE;
    if (VARIANT_FALSE == domApt->load("doc1.xml"))
    {
        printf("Can't load doc1.xml\n");
        return;
    }

    domFree->async = VARIANT_TRUE;
    if (VARIANT_FALSE == domFree->load("doc2.xml"))
    {
        printf("Can't load doc2.xml\n");
        return;
    }

    // Copy a node from domFree to domApt:
    // Fetch the "/doc" (node) from domFree (doc2.xml).
    // Clone node for import to domApt.
    // Append clone to domApt (doc1.xml).
    //
node = domFree->selectSingleNode("/doc");
if (NULL == node)
{
    printf("Can't fetch /doc from domFree\n");
    return;
}
clone = domApt->importNode(node, VARIANT_TRUE);
if (NULL == clone)
{
    printf("Can't clone /doc from doc2.xml\n");
    return;
}
domApt->documentElement->appendChild(clone);
domApt->documentElement->appendChild(domApt->createTextNode("\n"));
printf("doc1.xml after importing /doc from doc2.xml:\n");
printf(domApt->xml);
printf("\n");
node.Release();
node = NULL;
clone.Release();
clone = NULL;

// Clone a node using importNode() and append it to the same DOM:
// Fetch the "doc/b" (node) from domApt (doc1.xml).
// Clone node using importNode on domApt.
// Append clone to domApt (doc1.xml).
//
node = domApt->selectSingleNode("/doc/b");
if (NULL == node)
{
    printf("Can't fetch /doc/b from domApt\n");
    return;
}
clone = domApt->importNode(node, VARIANT_TRUE);
if (NULL == clone)
{
    printf("Can't clone /doc/b from doc1.xml\n");
    return;
}
domApt->documentElement->appendChild(domApt->createTextNode("\t"))
domApt->documentElement->appendChildChildren(domApt->createTextNode("\n"));
printf("doc1.xml after importing /doc/b from self:\n");
printf(domApt->xml);
printf("\n");
node.Release();
node = NULL;
clone.Release();
clone = NULL;

// Clone a node and append it to the dom using cloneNode():
// 1. Fetch "doc/a" (node) from domApt (doc1.xml).
// 2. Append clone to domApt (doc1.xml).

node = domApt->selectSingleNode("/doc/a");
if (NULL == node)
{
    printf("Can't fetch /doc/a from domApt\n");
    return;
}
clone = node->cloneNode(VARIANT_TRUE);
if (NULL == clone)
{
    printf("Can't clone /doc/a from doc1.xml\n");
    return;
}
domApt->documentElement->appendChild(clone);
printf("doc1.xml after cloning /doc/a from self:\n");
printf(domApt->xml);
printf("\n");
node.Release();
node = NULL;
clone.Release();
clone = NULL;

domApt->save("out.xml");
printf("a new document was saved to out.xml in the current workin

domApt.Release();
domFree.Release();
CoUninitialize();
}

Try It!

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "importNodeProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
4. Select FileView on the project browser, and highlight importNodeProj files. From the File menu, select New.
5. On the Files tab of the New dialog box, highlight C++ Source File. Then
type "importNode.xml" in the File name text box.

6. Click OK.

7. Copy the first XML data file (doc1.xml), and paste it into the text file you just created.

8. Repeat steps 4-7 for the second XML listing (doc2.xml).

Note You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ file above (importNode.cpp).

10. Build the sample by selecting Build importNodeProj.exe from the Build menu.

11. Execute the sample application by selecting !Execute importNodeProj.exe from the Build menu.

12. Verify that the output is the same as that listed in Output for the importNode Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Output for the importNode Example

When you build and run the importNode project, you should get the following output, displayed in three message boxes.

doc1.xml after importing /doc from doc2.xml:
<?xml version="1.0"?>
<doc>
  <a class="1">doc1</a>
  <b class="1">author1</b>
  <doc>
    <a class="2">doc 2</a>
    <b class="2">author 2</b>
  </doc>
</doc>

doc1.xml after importing /doc/b from self:
<?xml version="1.0"?>
<doc>
  <a class="1">doc1</a>
  <b class="1">author1</b>
  <doc>
    <a class="2">doc 2</a>
    <b class="2">author 2</b>
  </doc>
  <b class="1">author1</b>
</doc>

doc1.xml after cloning /doc/a from self:
<?xml version="1.0"?>
<doc>
  <a class="1">doc1</a>
  <b class="1">author1</b>
  <doc>
    <a class="2">doc 2</a>
    <b class="2">author 2</b>
  </doc>
  <b class="1">author1</b>
  <a class="1">doc1</a>
</doc>

A new document was saved to out.xml in the current working directory.
**insertBefore Method**

Inserts a child node to the left of the specified node, or at the end of the list.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNode.insertBefore(newChild, refChild);
```

**Parameters**

- `newChild`  
  An object. The address of the new node to be inserted.

- `refChild`  
  A variant. The address of the reference node; the `newChild` parameter is inserted to the left of the `refChild` parameter. If Null, the `newChild` parameter is inserted at the end of the child list.

**Return Value**

Object. On success, returns the child node that was inserted.

**Example**

The following script example creates a new IXMLDOMNode object and inserts it before the second child of the top-level node.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var newNode;
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
   var myErr = xmlDoc.parseError;
   alert("You have error " + myErr.reason);
} else {
```

root = xmlDoc.documentElement;
alert(root.xml);
newNode = xmlDoc.createElement("VIDEOS");
currNode = root.insertBefore(newNode, root.childNodes.item(1));
alert(root.xml);
}

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNode.insertBefore(newChild, refChild)

Parameters

newChild
An object. The address of the new node to be inserted.

refChild
A variant. The address of the reference node; the newChild parameter is
inserted to the left of the refChild parameter. If Null, the newChild
parameter is inserted at the end of the child list.

Return Value

An object. On success, returns the child node that was inserted.

Example

The following Visual Basic example creates a new IXMLDOMNode object and
inserts it before the second child of the top-level node.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim newNode As IXMLDOMNode
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
Set root = xmlDoc.documentElement
MsgBox root.xml
Set newNode = xmlDoc.createNode(NODE_ELEMENT, "VIDEOS", 
Set currNode = root.insertBefore(newNode, root.childNodes.Item(1))
MsgBox root.xml
End If

[C/C++]

C/C++ Syntax

HRESULT insertBefore(
    IXMLDOMNode *newChild,
    VARIANT refChild,
    IXMLDOMNode **outNewChild);

Parameters

newChild [in]
The address of the new node to be inserted. The node passed here must be a valid child of the current XML DOM document node. For example, if the current node is an attribute, you cannot pass another attribute in the newChild parameter, because an attribute cannot have an attribute as a child.

If newChild is of DOCUMENT_FRAGMENT node type, all its children are inserted in order before refChild. If newChild is already in the tree, it is first removed before it is reinserted before the refChild node. Read-only nodes, such as NODE_DOCUMENT_TYPE and NODE_ENTITY nodes, cannot be passed in the newChild parameter.

refChild [in]
The address of the reference node. The node specified is where the newChild node is to be inserted to the left as the preceding sibling in the child list. The node passed here must be a either a child node of the current node or Null. If the value is Null, the newChild node is inserted at the end of the child list. If the refChild node is not a child of the current node, an error is returned.

outNewChild [out, retval]
On success, the child node that was inserted. If Null, no object is created.
C/C++ Return Values

S_OK
The value returned if successful.

E_INVALIDARG
The value returned if the newChild parameter is Null.

E_FAIL
The value returned if an error occurs.

Remarks

The following are general remarks about the insertBefore method when DTD/schema rules or namespace prefixes are involved.

How DTD/Schema definitions are resolved

The newChild and refChild parameters can represent nodes in the same document or in different documents. When in the same document, the nodes retain their default attributes and data types. When in different documents, the nodes either lose or alter their default attributes, depending on whether the document that contains them has a document type definition (DTD) or other schema. MSXML attempts to correctly merge the different DTDs.

If the newChild node's DTD or schema differs from that of the original document, the nodes will be treated with the definitions in the new DTD, including default attributes and data types. If there is no DTD, the nodes keep their data types by picking up an instance definition in which elements are defined based upon how they are used in the current DOM document instance but any attributes within the newChild node fragment lose their data types and defaults. Cutting and pasting between documents with two different DTDs can result in an invalid document that might fail to parse after being saved.

How conflicts between prefixes and namespaces are resolved

If there is a conflict between prefixes on the containing element and the prefixes being added, the insertBefore operation fails and returns an error. For example, a conflict occurs when a new attribute referring to a different namespace is added to an element with the namespace, as in the following.
The error can result from the new attribute, where `myns` refers to a different namespace, "URN2," such as would result from a call to `createNode` ("attribute","myns:mynname","URN2").

```
myns:mynname="myattributevalue"
```

Namespace conflict errors occur only when setting attributes. Inserted child elements do not cause a namespace conflict.

When adding a document fragment, the containing element adds all namespaces and prefixes from the document fragment. If this causes a conflict with the containing element, `insertBefore` returns an error.

**Further remarks**

The use and operation of the `insertBefore` method depends on the value of the `nodeType` property for the `IXMLDOMNode` type object passed in the `newChild` parameter. The following topics provide further information about the results of calling `insertBefore` on different types of nodes.

- [Calling `insertBefore` on Attributes](#)
- [Calling `insertBefore` on Documents](#)
- [Calling `insertBefore` on Document Fragments](#)
- [Calling `insertBefore` on Elements](#)

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- [createNode Method](#) | [nodeType Property](#)

**Applies to:** [IXMLDOMAttribute](#) | [IXMLDOMCDATASection](#) | [IXMLDOMCharacterData](#) | [IXMLDOMComment](#) | [DOMDocument](#) | [IXMLDOMDocumentFragment](#) | [IXMLDOMDocumentType](#) | [IXMLDOMElement](#) | [IXMLDOMEntity](#) | [IXMLDOMEntityReference](#) | [IXMLDOMNode](#) | [IXMLDOMNotation](#) | [IXMLDOMProcessingInstruction](#) | [IXMLDOMText](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
### Calling `insertBefore` on Attributes

If the current node is an attribute, calling `insertBefore` has the following results, depending on the value of the `newChild` parameter.

<table>
<thead>
<tr>
<th>Node type passed in <code>newChild</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NODE_ATTRIBUTE</code>, <code>NODE_CDATA_SECTION</code>, <code>NODE_COMMENT</code>, <code>NODE_DOCUMENT</code>, <code>NODE_DOCUMENT_TYPE</code>, <code>NODE_ELEMENT</code>, <code>NODE_ENTITY</code>, <code>NODE_NOTATION</code>, <code>NODE_PROCESSING_INSTRUCTION</code></td>
<td>Returns an error. These node types cannot be children of an attribute.</td>
</tr>
<tr>
<td><code>NODE_DOCUMENT_FRAGMENT</code></td>
<td>Inserts the children of the document fragment (<code>newChild</code>) and returns <code>newChild</code>.</td>
</tr>
<tr>
<td><code>NODE_ENTITY_REFERENCE</code>, <code>NODE_TEXT</code></td>
<td>Inserts <code>newChild</code> and returns <code>newChild</code>.</td>
</tr>
<tr>
<td><code>NODE_CDATA_SECTION</code>, <code>NODE_COMMENT</code>, <code>NODE_ENTITY</code>, <code>NODE_NOTATION</code>, <code>NODE_PROCESSING_INSTRUCTION</code>, <code>NODE_TEXT</code></td>
<td>Returns an error. These node types either cannot have children or their children are read-only.</td>
</tr>
</tbody>
</table>

### Remarks

If an entity reference is added to an attribute, the namespace and the prefix of the entity reference are added to the element containing the attribute.
Calling `insertBefore` on Documents

If the current node is a document, calling `insertBefore` has the following results, depending on the value of the `newChild` parameter.

<table>
<thead>
<tr>
<th>Node type passed in <code>newChild</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ATTRIBUTE, NODE_CDATA_SECTION, NODE_DOCUMENT, NODE_ENTITY, NODE_ENTITY_REFERENCE, NODE_NOTATION, NODE_TEXT</td>
<td>Returns an error. These nodes are not valid as children of a document node.</td>
</tr>
<tr>
<td>NODE_COMMENT, NODE_PROCESSING_INSTRUCTION</td>
<td>Inserts <code>newChild</code> and returns <code>newChild</code>.</td>
</tr>
<tr>
<td>NODE_ELEMENT</td>
<td>Inserts <code>newChild</code> and returns <code>newChild</code>. By definition, an XML document (the Document node) can have only a single child. Therefore, an error is returned if the Document node already has a child.</td>
</tr>
<tr>
<td>NODE_DOCUMENT_FRAGMENT</td>
<td>Inserts the children of the document fragment (in <code>newChild</code>) and returns <code>newChild</code>. The insert operations are subject to the rules for child nodes and can fail if the document fragment children represent node types that cannot be inserted.</td>
</tr>
<tr>
<td>NODE_DOCUMENT_TYPE</td>
<td>Returns an error. The document type is read-only.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Calling `insertBefore` on Document Fragments

If the current node is a document fragment, calling `insertBefore` has the following results, depending on the value of the `newChild` parameter.

<table>
<thead>
<tr>
<th>Node type passed in <code>newChild</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ATTRIBUTE, NODE_DOCUMENT, NODE_DOCUMENT_TYPE</td>
<td>Returns an error. These nodes types are not valid as children of a document fragment.</td>
</tr>
<tr>
<td>NODECDATA_SECTION, NODECOMMENT, NODEELEMENT, NODEENTITYREFERENCE,</td>
<td></td>
</tr>
<tr>
<td>NODEPROCESSINGINSTRUCTION, NODETEXT</td>
<td></td>
</tr>
<tr>
<td>NODEDOCUMENT_FRAGMENT</td>
<td></td>
</tr>
<tr>
<td>NODEENTITY, NODENOTATION</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**

If `newChild` is a node of type DOCUMENT_FRAGMENT, all its children are inserted, in the same order, before `refChild`.

If an entity reference is added as a child to a document fragment, the namespace declaration of the entity reference is added to the containing element.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
# Calling insertBefore on Elements

If the current node is an element, calling `insertBefore` has the following results, depending on the value of the `newChild` parameter.

<table>
<thead>
<tr>
<th>Node type passed in <code>newChild</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_CDATA_SECTION, NODE_COMMENT, NODE_ELEMENT, NODE_ENTITY_REFERENCE, NODE_TEXT, NODE_PROCESSING_INSTRUCTION</td>
<td>Inserts <code>newChild</code> and returns <code>newChild</code>.</td>
</tr>
<tr>
<td>NODE_ATTRIBUTE, NODE_DOCUMENT, NODE_DOCUMENT_TYPE, NODE_ENTITY, NODE_NOTATION</td>
<td>Returns an error. These node types cannot be children of an element node.</td>
</tr>
<tr>
<td>NODE_DOCUMENT_FRAGMENT</td>
<td>Inserts the children of the document fragment (<code>newChild</code>) and returns <code>newChild</code>.</td>
</tr>
<tr>
<td>NODE_ENTITY_REFERENCE</td>
<td>Returns an error. Although the child nodes of an entity reference are the expanded entity, the children cannot be modified.</td>
</tr>
</tbody>
</table>

**Remarks**

When elements are added as children to other elements, they introduce their own namespace scoping. That is, if the element is from a default namespace, the default has scope beginning with the element being added.

If an entity reference is being added as a child to an element or document fragment, the namespace declaration of the entity reference is added to the containing element.
insertData Method

Inserts a string at the specified offset.

[Script]

Script Syntax

```
oXMLDOMCharacterData.insertData(offset, data);
```

Parameters

offset
A long integer specifying the offset, in characters, at which to insert the supplied string data.

data
A string containing the data that is to be inserted into the existing string.

Example

The following script example creates an IXMLDOMComment object and uses the insertData method to insert a string into it.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    comment = xmlDoc.createComment("Hello World!");
    comment.insertData(6, "Beautiful ");
    alert(comment.xml);
}
```

[Visual Basic]

Visual Basic Syntax
Parameters

offset
A long integer specifying the offset, in characters, at which to insert the supplied string data.

data
A string containing the data that is to be inserted into the existing string.

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMComment object and uses the insertData method to insert a string into it.

```vba
Dim xmlDoc As New Msxml2.DOMDocument50
Dim comment As IXMLDOMComment
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set comment = xmlDoc.createComment("Hello World!")
    comment.insertData 6, "Beautiful "
    MsgBox(comment.xml)
End If
```

[C/C++]

C/C++ Syntax

```c
HRESULT insertData(
    long offset,
    BSTR data);
```

Parameters

offset [in]
The offset, in characters, at which to insert the supplied string data.

data [in]
The string data that is to be inserted into the existing string.

C/C++ Return Values

S_OK
   The value returned if successful.
S_FALSE
   The value when returning Null.
E_FAIL
   The value returned if an error occurs.

Remarks

The length property is updated by this operation.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

length Property

Applies to: IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
**item Method**  
**IXMLDOMNamedNodeMap**

Allows random access to individual nodes within the collection.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNamedNodeMap.item(index);
```

**Parameters**

`index`

A long integer. An index of the item within the collection. The first item is zero.

**Return Value**

An object. Returns IXMLDOMNode. Returns Null if the index is out of range.

**Example**

The following script example creates an IXMLDOMNamedNodeMap object to retrieve the attributes for an element node selected using the SelectSingleNode method. It then iterates through the attributes, before displaying the name and value of each attribute in the collection.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oNamedNodeMap, nodeBook, str;
var str = "";
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
```
} else {
    xmlDoc.setProperty("SelectionLanguage", "XPath");
    nodeBook = xmlDoc.selectSingleNode("//book");
oNamedNodeMap = nodeBook.attributes;
    for (var i=0; i<oNamedNodeMap.length; i++) {
        str += "Attr" + i + " name: " +
        oNamedNodeMap.item(i).name + "
        Attr" + i + " value: " +
        oNamedNodeMap.item(i).text + "\n";
    }
    alert(str);
}

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNamedNodeMap.item(index)

Parameters

index
    A long integer. Index of the item within the collection. The first item is zero.

Return Value

An object. Returns IXMLDOMNode. Returns Null if the index is out of range.

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNamedNodeMap object to retrieve the attributes for an element node selected using the SelectSingleNode method. It then iterates through the attributes, before displaying the name and value of each attribute in the collection.

Dim xmlDoc As New MSXML2.DOMDocument50
Dim oNamedNodeMap As IXMLDOMNamedNodeMap
Dim nodeBook As IXMLDOMElemnt
Dim str As String
str = ""

xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set nodeBook = xmlDoc.selectSingleNode("//book")
  Set oNamedNodeMap = nodeBook.Attributes
  For i = 0 To (oNamedNodeMap.length - 1)
    str = str & _
    "Attr" & CStr(i) & " name: " & _
    oNamedNodeMap.Item(i).nodeName & vbCrLf & _
    "Attr" & CStr(i) & " value: " & _
    oNamedNodeMap.Item(i).nodeValue & vbCrLf
  Next
  MsgBox str
End If

[C/C++]

C/C++ Syntax

HRESULT get_item(
    long index,
    IXMLDOMNode **listItem);

Parameters

index [in]
    The index of the item within the collection. The first item is zero.

listItem [out, retval]
    The IXMLDOMNode object. Returns Null if the index is out of range.

C/C++ Return Values

S_OK
    The value returned if successful.

E_INVALIDARG
    The value returned if the listItem parameter is Null.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

IXMLDOMNode

Applies to: IXMLDOMNode | IXMLDOMNode | IXMLDOMNodeSelection | IXMLDOMNodeSelection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
item Method (IXMLDOMNodeList)

Allows random access to individual nodes within the collection.

[Script]

Script Syntax

```
var objXMLDOMNode = oXMLDOMNodeList.item(index);
```

Parameters

**index**

A long integer. An index of the item within the collection. The first item is zero.

Return Value

An object. Returns IXMLDOMNode. Returns Null if the index is out of range.

Example

The following script example creates an IXMLDOMNodeList object with the document’s getElementsByTagName method. It then iterates through the collection, displaying the text value of each item in the list.

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    objNodeList = xmlDoc.getElementsByTagName("author");
    for (var i=0; i<objNodeList.length; i++) {
        alert(objNodeList.item(i).text);
    }
```
Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNodeList.item(index)

Parameters

index
A long integer. An index of the item within the collection. The first item is zero.

Return Value

An object. Returns IXMLElement. Returns Null if the index is out of range.

Example

The following Microsoft® Visual Basic® example creates an IXMLElement object with the document's getElementsByTagName method. It then iterates through the collection, displaying the text value of each item in the list.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLElementList
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set objNodeList = xmlDoc.getElementsByTagName("author")
  For i = 0 To (objNodeList.length - 1)
    MsgBox objNodeList.Item(i).Text
  Next
End If

[C/C++]
C/C++ Syntax

HRESULT get_item(
    long index,
    IXMLDOMNode **listItem);

Parameters

index [in]
    An index of the item within the collection. The first item is number zero.

listItem [out, retval]
    An IXMLDOMNode. Returns Null if the index is out of range.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the listItem parameter is Null.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode

Applies to: IXMLDOMNodeList | IXMLDOMSelection
load Method

Loads an XML document from the specified location.

[Script]

Script Syntax

boolValue = oXMLDOMDocument.load(xmlSource);

Parameters

xmlSource
A string containing a URL that specifies the location of the XML file.

Return Value

Boolean. Returns True if the load succeeded; False if the load failed.

Example

The following script example creates a DOMDocument object and uses the load method to load a local XML file.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("C:\\temp\\books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    alert(xmlDoc.xml);
}
```

[Visual Basic]

Visual Basic Syntax
boolValue = oXMLDOMDocument.load(xmlSource)

**Parameters**

*xmlSource*

A string containing a URL that specifies the location of the XML file.

**Return Value**

Boolean. Returns True if the load succeeded; False if the load failed.

**Example**

The following Microsoft Visual Basic® example creates a DOMDocument object and uses the load method to load a local XML file.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.Load("C:\temp\books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox xmlDoc.xml
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT load(
    VARIANT xmlSource,
    VARIANT_BOOL *isSuccessful);
```

**Parameters**

*xmlSource* [in]

An indicator of the source XML to parse. This may be an URL (String/BSTR), a Request object (in an ASP page), an IStream, SAFEARRAY of bytes (VT_ARRAY|VT_UI1), a DOMDocument object, or
any object that supports IStream, ISequentialStream, or IPersistStream. See Remarks for more information.

isSuccessful [out, retval]
True if the load succeeded; False if the load failed.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value returned if the load fails.
E_INVALIDARG
The value returned if the isSuccessful parameter is Null.

Remarks

If the URL cannot be resolved or accessed or does not reference an XML document, this method returns an error and sets the documentElement property of the DOMDocument to Null.

The load method can also take any object that supports IStream and the Microsoft® Internet Information Services (IIS) Request object.

Calling load or loadXML on an existing document immediately discards the content of the document.

If loading an XML document from a resource, the load must be performed asynchronously or the load will fail. For example:

```wscript
Set objXML = CreateObject("Msxml2.DOMDocument.5.0")
ObjXML.async=true
objXML.load "res://msxml.dll/DEFAULTSS.XSL"
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    WScript.echo objXML.parseError.reason
    objXML.save "c:\defaultss.xsl"
}
```

You can use this method to check if the loaded XML document is well-formed.
You cannot use it to validate the XML document against a schema.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

documentElement Property | loadXML Method | Persistence and the DOM

Applies to: DOMDocument
loadXML Method

Loads an XML document using the supplied string.

[Script]

Script Syntax

boolValue = oXMLDOMDocument.loadXML(bstrXML);

Parameters

bstrXML
   A string containing the XML string to load into this XML document object. This string can contain an entire XML document or a well-formed fragment.

Return Value

Boolean. Returns True if the XML load succeeded. Returns False and sets the documentElement property of the DOMDocument to Null if the XML load failed.

Example

The following script example creates a DOMDocument object, and then uses its loadXML method to load the specified XML before displaying it.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.loadXML("<customer><first_name>Joe</first_name><last_name>Smith</last_name></customer>);
if (xmlDoc.parseError.errorCode <> 0) {
   var myErr = xmlDoc.parseError;
   alert("You have error " + myErr.reason);
} else {
   alert(xmlDoc.xml);
}
```

[Visual Basic]
**Visual Basic Syntax**

```vbnet
boolValue = oXMLDOMDocument.loadXML(bstrXML)
```

**Parameters**

*bstrXML*

A string containing the XML string to load into this XML document object. This string can contain an entire XML document or a well-formed fragment.

**Return Value**

Boolean. Returns True if the XML load succeeded. Returns False and sets the `documentElement` property of the `DOMDocument` to Null if the XML load failed.

**Example**

The following Microsoft® Visual Basic® example creates a `DOMDocument` object, and then uses its `loadXML` method to load the specified XML before displaying it.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
xmlDoc.async = False
xmlDoc.loadXML ("<customer>
    "<first_name>Joe</first_name>
    "<last_name>Smith</last_name></customer>"

If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox xmlDoc.xml
End If
```

**C/C++ Syntax**

```c
HRESULT loadXML(  
    BSTR bstrXML,  
    VARIANT_BOOL * isSuccessful);
```
Parameters

*bstrXML* [in]
An XML string to load into this XML document object. This string can contain an entire XML document or a well-formed fragment.

*isSuccessful* [out, retval]
True if the XML load succeeded. If the XML load failed, this method returns False and sets the *documentElement* property of the *DOMDocument* object to Null.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value returned if the load fails.
E_INVALIDARG
The value returned if the *isSuccessful* parameter is Null.

Remarks

Calling *load* or *loadXML* on an existing document immediately discards the content of the document. The *loadXML()* method will work only with UTF-16 or UCS-2 encodings.

You can use this method to check if the loaded XML document is well-formed. You cannot use it to validate the XML document against a schema.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

documentElement property | load Method | Persistence and the DOM
Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
matches Method

Checks if the node that is passed is contained in the current collection.

[Script]

Script Syntax

```
objXMLDOMNode = objXMLDOMSelection.matches(objXMLDOMNode);
```

Example

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oSelection, nodeBook, node;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  nodeBook = xmlDoc.selectSingleNode("//book");
  oSelection = xmlDoc.selectNodes("//*");
  node = oSelection.matches(nodeBook);
  alert(node.xml);
}
```

[Visual Basic]

Visual Basic Syntax

```
objXMLDOMNode = objXMLDOMSelection.matches(objXMLDOMNode)
```

Example

```
Dim xmlDoc As New Msxml2.DOMDocument50
Dim oSelection As IXMLDOMSelection
Dim nodeBook As IXMLDOMNode
Dim node As IXMLDOMNode
xmlDoc.setProperty "SelectionLanguage", "XPath"
```
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set nodeBook = xmlDoc.selectSingleNode("//book")
  Set oSelection = xmlDoc.selectNodes("/**")
  Set node = oSelection.matches(nodeBook)
  MsgBox node.xml
End If

[C/C++]

C/C++ Syntax

HRESULT matches(IXMLDOMNode* pNode, IXMLDOMNode** ppNode);

Parameters

pNode [in]
  The node that is passed in.
ppNode [out, retval]
  The Boolean result.

C/C++ Return Values

S_OK
  The value returned if method successful.
E_INVALIDARG
  The value returned if the pNode parameter is Null.

An error for cases that are not allowed, as outlined in the Extensible Stylesheet Language (XSL) specification (applied to ID, IDREF, and Ancestor).

Remarks

When matches is called with node A, it returns node B such that if B was set as the context on a query, A is in the result set of the query. If no such B node is
found, matches returns Null.

The matches method does not take into account the current context of the query.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSelection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**nextNode Method**

*(IXMLDOMNamedNodeMap)*

Returns the next node in the collection.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNamedNodeMap.nextNode();
```

**Return Value**

An IXMLDOMNode, which refers to the next node in the collection. Returns Null if there is no next node.

**Example**

The following script example creates an IXMLDOMNodeMap object and uses its `nextNode` method to iterate the collection.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNamedNodeMap;
var objNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    objNamedNodeMap = xmlDoc.documentElement.childNodes(1).attributes;
    for (var i=0; i<objNamedNodeMap.length; i++) {
        objNode = objNamedNodeMap.nextNode();
        alert(objNode.text);
    }
}
```

[Visual Basic]
**Visual Basic Syntax**

Set objXMLDOMNode = oXMLDOMNamedNodeMap.nextNode

**Return Value**

An IXMLDOMNode, which refers to the next node in the collection. Returns Null if there is no next node.

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMNamedNodeMap object and uses its nextNode method to iterate the collection.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNamedNodeMap As IXMLDOMNamedNodeMap
Dim objNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set objNamedNodeMap = xmlDoc.documentElement.childNodes(1).Attributes
    For i = 0 To (objNamedNodeMap.length - 1)
        Set objNode = objNamedNodeMap.nextNode
        MsgBox objNode.Text
    Next
End If
```

**C/C++ Syntax**

```c
HRESULT nextNode(
    IXMLDOMNode **nextItem);
```

**Parameters**

`nextItem` [out, retval]
The next node in the collection, or Null if there is no next node.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value when returning Null.
E_INVALIDARG
The value returned if the nextItem parameter is Null.

Remarks

The iterator initially points before the first node in the list so that the first call to the nextNode method returns the first node in the list.

This method returns Null when the current node is the last node or there are no items in the list. When the current node is removed from the list, subsequent calls to nextNode return Null. The iterator must be reset by calling the reset method.

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode | reset Method

Applies to: IXMLDOMNodeNamedNodeMap | IXMLDOMNodeList | IXMLDOMSelection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
nextNode Method
(IXMLDOMNodeList)

Returns the next node in the collection.

[Script]

Script Syntax

var objXMLDOMNode = oXMLDOMNodeList.nextNode();

Return Value

An IXMLDOMNode refers to the next node in the collection. Returns Null if there is no next node.

Example

The following script example creates an IXMLDOMNodeList object and uses its nextNode method to iterate the collection.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
var objNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    objNodeList = xmlDoc.getElementsByTagName("author");
    for (var i=0; i<objNodeList.length; i++) {
        objNode = objNodeList.nextNode();
        alert(objNode.text);
    }
}
```

[Visual Basic]
**Visual Basic Syntax**

Set objXMLDOMNode = oXMLDOMNodeList.nextNode

**Return Value**

An IXMLDOMNode refers to the next node in the collection. Returns Null if there is no next node.

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMNodeList object and uses its nextNode method to iterate the collection.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLDOMNodeList
Dim objNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set objNodeList = xmlDoc.getElementsByTagName("author")
    For i = 0 To (objNodeList.length - 1)
        Set objNode = objNodeList.nextNode
        MsgBox objNode.Text
    Next
End If
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT nextNode(
    IXMLDOMNode **nextItem);
```

**Parameters**

*nextItem* [out, retval]

The next node in the collection, or Null if there is no next node.
C/C++ Return Values

S_OK
   The value returned if successful.
S_FALSE
   The value returned if the nextItem parameter is Null.

Remarks

The iterator initially points before the first node in the list so that the first call to the nextNode method returns the first node in the list.

This method returns Null when the current node is the last node or there are no items in the list. When the current node is removed from the list, subsequent calls to nextNode return Null. The iterator must be reset by calling the reset method.

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode | reset Method

Applies to: IXMLDOMNodeNamedNodeMap | IXMLDOMNodeList | IXMLDOMSelection
**nodeFromID Method**

Returns the node that matches the ID attribute.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMDocument.nodeFromID(idString);
```

**Parameters**

*idString*

The string containing the value of the ID to match.

**Return Value**

An object. Returns the node that matches the supplied ID. If no nodes match, returns Null.

[Visual Basic]

**Visual Basic Syntax**

```vbscript
Set objXMLDOMNode = oXMLDOMDocument.nodeFromID(idString)
```

**Parameters**

*idString*

The string containing the value of the ID to match.

**Return Value**

An object. Returns the node that matches the supplied ID. If no nodes match, returns Null.
[C/C++]

C/C++ Syntax

HRESULT nodeFromID(
    BSTR idString,
    IXMLDOMNode **node);

Parameters

idString [in]
The value of the ID to match.
node [out, retval]
The node that matches the supplied ID. If no nodes match, this method returns Null.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value returned when there is no node with the given ID.
E_INVALIDARG
The value returned if the node parameter is Null.

Remarks

According to the XML 1.0 Recommendation (REC-xml-19980210), ID attribute values must be unique within their XML documents and no element can specify more than one ID attribute.

The nodeFromID method was designed to handle ID and IDREF relationships in XML, but does not require an attribute of type IDREF. It can be used generically, and is similar to the all collection in DHTML.

To reference a node with nodeFromID, the node must be typed as ID in the schema or document type definition (DTD). Simply naming an attribute "ID" does not set its data type.
This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
normalize Method

Normalizes all descendant elements by combining two or more adjacent text nodes into one unified text node.

[Script]

Script Syntax

```javascript
oXMLDOMElement.normalize();
```

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeRoot;
xmlDoc.async = false;
xmlDoc.loadXML("<root/>");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeRoot = xmlDoc.documentElement;
    nodeRoot.appendChild(xmlDoc.createTextNode("Hello "));  
    nodeRoot.appendChild(xmlDoc.createTextNode("World"));  
    nodeRoot.appendChild(xmlDoc.createTextNode("!"));  
    alert(nodeRoot.childNodes.length);
    nodeRoot.normalize();  
    alert(nodeRoot.childNodes.length);
}
```

[Visual Basic]

Visual Basic Syntax

```visualbasic
oXMLDOMElement.normalize
```

Example

```visualbasic
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeRoot As IXMLDOMElement
```
xmlDoc.async = False
xmlDoc.loadXML "<root/>
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set nodeRoot = xmlDoc.documentElement
  nodeRoot.appendChild xmlDoc.createTextNode("Hello ")
  nodeRoot.appendChild xmlDoc.createTextNode("World")
  nodeRoot.appendChild xmlDoc.createTextNode("!")
  MsgBox nodeRoot.childNodes.length
  nodeRoot.normalize
  MsgBox nodeRoot.childNodes.length
End If

[C/C++]

C/C++ Syntax

HRESULT normalize(void);

C/C++ Return Values

S_OK
  The value returned if successful.
E_FAIL
  The value returned if an error occurs.

C/C++ Example

BOOL DOMElementNormalize()
{
  BOOL bResult = FALSE;
  IXMLDOMElement *pIXMLDOMElement = NULL;
  IXMLDOMDocument *pIXMLDOMDocument = NULL;
  HRESULT hr;

  try
  {
    // Create an instance of DOMDocument and initialize
    // pIXMLDOMDocument.
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;
  }
  catch
  {
    // Handle exception.
  }
}

if(pIXMLDOMElement)
{
    hr = pIXMLDOMElement->normalize();
    if(SUCCEEDED(m_hr))
        bResult = TRUE;
    pIXMLDOMElement->Release();
}

} 
catch(...)
{
    if(pIXMLDOMElement)
        pIXMLDOMElement->Release();
    DisplayErrorToUser();
} 
return bResult;

Remarks

This method converts all text node descendants of this element (at any depth) into a "normal" form where no text nodes are adjacent. In normal form, text nodes can be separated only by markup, such as tags, comments, processing instructions, CDATA sections, and entity references. The normal form is useful for operations that require a particular document tree structure and ensures that the XML Document Object Model (DOM) view of a document is identical when saved and reloaded.

Collapsed node objects are deleted automatically.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMText

Applies to: IXMLDOMElement
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
open Method (IXMLHTTPRequest)

Initializes an MSXML2.XMLHTTP request and specifies the method, URL, and authentication information for the request.

[Script]

Script Syntax

oXMLHttpRequest.open(bstrMethod, bstrUrl, varAsync, bstrUser, bstrPassword);

Parameters

bstrMethod
The HTTP method used to open the connection, such as GET, POST, PUT, or PROPFIND.

bstrUrl
The requested URL. This can be either an absolute URL, such as "http://Myserver/MyPath/Myfile.asp", or a relative URL, such as "../MyPath/MyFile.asp".

varAsync [optional]
A Boolean indicator of whether the call is asynchronous. The default is True (the call returns immediately). If set to True, attach an onreadystatechange property callback so that you can tell when the send call has completed.

bstrUser [optional]
The name of the user for authentication. If this parameter is Null ("") or missing and the site requires authentication, the component displays a logon window.

bstrPassword [optional]
The password for authentication. This parameter is ignored if the user parameter is Null ("") or missing.

Example

The following script example creates an XMLHttpRequest object, and then uses the open
method to synchronously open an Active Server Pages (ASP) page. The script then posts an XML document to an ASP page, which uses the `load` method to load the document.

```javascript
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmlhttp.open("GET","http://myserver/save.asp", false);
xmlhttp.send(xmlDoc)
```

The following code belongs on the server.

```xml
<% xmlDoc.load(Request); %>
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
oXMLHttpRequest.open(bstrMethod, bstrUrl, varAsync, bstrUser, bstrPassword)
```

**Parameters**

- **bstrMethod**
  The HTTP method used to open the connection, such as GET, POST, PUT, or PROPFIND.

- **bstrUrl**
  The requested URL. This can be either an absolute URL, such as "http://Myserver/Mypath/Myfile.asp", or a relative URL, such as "./MyPath/MyFile.asp".

- **varAsync** [optional]
  A Boolean indicator of whether the call is asynchronous. The default is True (the call returns immediately). If set to True, attach an `onreadystatechange` property callback so that you can tell when the `send` call has completed.

- **bstrUser** [optional]
  The name of the user for authentication. If this parameter is Null (""") or missing and the site requires authentication, the component displays a logon window.

- **bstrPassword** [optional]
  The password for authentication. This parameter is ignored if the user parameter is Null (""") or missing.
[C/C++]

**C/C++ Syntax**

```c
HRESULT open(BSTR bstrMethod, BSTR bstrUrl, VARIANT varAsync,
VARIANT bstrUser, VARIANT bstrPassword);
```

**Parameters**

*bstrMethod* [in]

The HTTP method used to open the connection, such as PUT or PROPFIND.

*bstrUrl* [in]

The requested URL. This can be either an absolute URL, such as "http://Myserver/Mypath/Myfile.asp", or a relative URL, such as "./MyPath/MyFile.asp".

*varAsync* [in, optional]

A Boolean indicator as to whether the call is asynchronous. The default is True (the call returns immediately). If set to True, attach an onreadystatechange property callback so that you can tell when the send call has completed.

*bstrUser* [in, optional]

The name of the user for authentication. If this parameter is Null ("") or missing and the site requires authentication, the component displays a logon window.

*bstrPassword* [in, optional]

The password for authentication. This parameter is ignored if the user parameter is Null or missing.

**C/C++ Return Values**

*S_OK*

The value returned if successful.

**C/C++ Example**

```c
HRESULT hr;
IXMLHttpRequest *pIXMLHttpRequest = NULL;
```
try {
    // Create XMLHttpRequest object and initialize pIXMLHttpRequest.
    hr = pIXMLHttpRequest->open(_bstr_t(_T("PUT")),
        _bstr_t(_T("GET","http://MyServer/Sample.xml", false)),
        _variant_t(VARIANT_FALSE), _variant_t(""), _variant_t(""));
    if(SUCCEEDED(hr))
        ::MessageBox(NULL, _T("Success !"), _T(""), MB_OK);
} catch(...) {
    DisplayErrorToUser();
}

// Release pIXMLHttpRequest when finished with it.

**Remarks**

After calling this method, you must call send to send the request and data, if any, to the server.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

abort Method | onreadystatechange Property

Applies to: IXMLHTTPRequest
open Method
(ServerXMLHTTP/IServerXMLHTTP)

Initializes a request and specifies the method, URL, and authentication information for the request.

[Script]

Script Syntax

```
oServerXMLHTTPRequest.open(bstrMethod, bstrUrl, bAsync, bstrUser, bstrPassword);
```

Parameters

- **bstrMethod**
  The HTTP method used to open the connection, such as PUT or PROPFIND.
- **bstrUrl**
  The requested URL. This can be either an absolute URL, such as "http://Myserver/Mypath/Myfile.asp", or a relative URL, such as ".../MyPath/MyFile.asp".
- **bAsync** (optional)
  Boolean. Indicator as to whether the call is asynchronous. The default is False (the call does not return immediately).
- **bstrUser** (optional)
  The name of the user for authentication.
- **bstrPassword** (optional)
  The password for authentication. This parameter is ignored if the user parameter is Null or missing.

Example

```
<%@language=JScript%>
<%
var srvXmlHttp
srvXmlHttp = Server.CreateObject("Msxml2.ServerXMLHTTP.5.0");
srvXmlHttp.open ("GET", "http://myserver/myresponse.asp", false);
srvXmlHttp.send();
newsElement = srvXmlHttp.responseXML.selectSingleNode("/news/story1" %>

<html>
<body>
<p>Top News Story<p>
<%Response.write(newsElement.text);%>
</body>
</html>

[Visual Basic]

**Visual Basic Syntax**

```vbnet
oServerXMLHTTPRequest.open(bstrMethod, bstrUrl, bAsync, bstrUser, bstrPassword)
```

**Parameters**

*bstrMethod*
- The HTTP method used to open the connection, such as PUT or PROPFIND.

*bstrUrl*
- The requested URL. This can be either an absolute URL, such as "http://Myserver/Mypath/Myfile.asp", or a relative URL, such as ".../MyPath/MyFile.asp".

*bAsync* (optional)
- Boolean. Indicator as to whether the call is asynchronous. The default is False (the call does not return immediately).

*bstrUser* (optional)
- The name of the user for authentication. If this parameter is Null ("") or missing and the site requires authentication, the component displays a logon window.

*bstrPassword* (optional)
- The password for authentication. This parameter is ignored if the user parameter is Null or missing.

[C/C++]
**C/C++ Syntax**

```c
HRESULT open(BSTR bstrMethod, BSTR bstrUrl, VARIANT bAsync,
VARIANT bstrUser, VARIANT bstrPassword);
```

**Parameters**

*bstrMethod [in]*
- The HTTP method used to open the connection, such as PUT or PROPFIND.

*bstrUrl [in]*
- The requested URL. This can be either an absolute URL, such as "http://Myserver/Mypath/Myfile.asp", or a relative URL, such as ".../MyPath/MyFile.asp".

*bAsync [in, optional]*
- Boolean. Indicator as to whether the call is asynchronous. The default is False (the call does not return immediately).

*bstrUser [in, optional]*
- The name of the user for authentication. If this parameter is Null (""") or missing and the site requires authentication, the component displays a logon window.

*bstrPassword [in, optional]*
- The password for authentication. This parameter is ignored if the user parameter is Null or missing.

**C/C++ Return Values**

*S_OK*
- The value returned if successful.

**Remarks**

After calling this method you must call the send method to send the request and data (if any) to the server. Each send method must have a corresponding open method.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

abort Method

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
peekNode Method

Gets the next node that the nextNode method will return without advancing the list position.

[Script]

Script Syntax

var objXMLDOMNode = objXMLDOMSelection.peekNode();

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oSelection, nodeBook;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  oSelection = xmlDoc.selectNodes("//book");
  nodeBook = oSelection.peekNode();
  alert(nodeBook.xml);
}

[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNode = objXMLDOMSelection.peekNode

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim oSelection As IXMLDOMSelection
Dim nodeBook As IXMLDOMElement
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set oSelection = xmlDoc.selectNodes("//book")
    Set nodeBook = oSelection.peekNode()
    MsgBox nodeBook.xml
End If

[C/C++]

**C/C++ Syntax**

HRESULT peekNode (IXMLDOMNode** ppNode);

**Parameters**

**ppNode** [out, retval]
   The returned node, or Null if there are no more nodes or if E_PENDING is returned.

**C/C++ Return Values**

S_OK
   The value returned if the method is successful.
E_PENDING
   The value returned if the context document is still being built and the selection object has hit the end of the available nodes to match.

**Remarks**

Like nextNode, peekNode does not result in a snapshot of all matching nodes. Consecutive calls to peekNode will produce the same node over and over, independent of changes in the underlying tree.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
See Also

Applies to: IXMLDOMSelection
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
remove Method

Removes the specified namespace from a collection.

[Script]

Script Syntax

objXMLDOMSchemaCol.remove(namespaceURI);

Parameters

namespaceURI

The namespace to remove from the collection. This can be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute occurs on this argument (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

[Visual Basic]

Visual Basic Syntax

objXMLDOMSchemaCol.remove(namespaceURI);

Parameters

namespaceURI

The namespace to remove from the collection. This can be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute occurs on this argument (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).
[C/C++]

C/C++ Syntax

HRESULT remove(BSTR namespaceURI);

Parameters

namespaceURI [in]
The namespace to remove from the collection.

This can be any string that can be used in an xmlns attribute, but it cannot contain entity references. The same white space normalization that occurs on the xmlns attribute occurs on this argument (that is, leading and trailing white space is trimmed, new lines are converted to spaces, and multiple adjacent white space characters are collapsed into one space).

C/++Return Values

E_INVALIDARG
The value returned if the namespace does not exist in the collection.
E_FAIL
The value returned if the collection is read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
removeAll Method

Removes all the nodes from the collection described by the IXMLDOMSelection.

[Script]

**Script Syntax**

```javascript
objXMLDOMSelection.removeAll();
```

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oSelection;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  alert(xmlDoc.xml);
  oSelection = xmlDoc.selectNodes("//book");
oSelection.removeAll();
  alert(xmlDoc.xml);
}
```

[Visual Basic]

**Visual Basic Syntax**

```vbasic
objXMLDOMSelection.removeAll()
```

**Example**

```vbasic
Dim xmlDoc As New Msxml2.DOMDocument50
Dim oSelection As IXMLDOMSelection
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
```
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    MsgBox xmlDoc.xml
    Set oSelection = xmlDoc.selectNodes("//book")
    oSelection.removeAll
    MsgBox xmlDoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT removeAll();

C/C++ Return Values

S_OK
    The value returned if the method is successful.
E_PENDING
    The value returned if all nodes cannot be found at this time (in which case
    no nodes are removed).

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button [ ] in the upper-left corner of the page.

See Also

Applies to: IXMLDOMSelection
removeAttribute Method

Removes or replaces the named attribute.

[Script]

Script Syntax

oXMLDOMElement.removeAttribute(name);

Parameters

name

A string specifying the name of the attribute to be removed or replaced.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("/book");
    alert(nodeBook.attributes.length);
    nodeBook.removeAttribute("id");
    alert(nodeBook.attributes.length);
}

[Visual Basic]

Visual Basic Syntax

oXMLDOMElement.removeAttribute(name)

Parameters
name

A string specifying the name of the attribute to be removed or replaced.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    MsgBox nodeBook.Attributes.length
    nodeBook.removeAttribute "id"
    MsgBox nodeBook.Attributes.length
End If

[C/C++]

C/C++ Syntax

HRESULT removeAttribute(
    BSTR name);

Parameters

name [in]
    The name of the attribute to be removed or replaced.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value returned when no attribute with the given name is found.
E_FAIL
    The value returned if an error occurs.
C/C++ Example

```c
IXMLDOMElement *pIXMLDOMElement = NULL;
_bstr_t bstrAttributeName = _T("dateCreated");
IXMLDOMDocument *pIXMLDOMDocument = NULL;
HRESULT hr;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocum
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;

    hr = pIXMLDOMElement->removeAttribute(bstrAttributeName);
    if(SUCCEEDED(hr))
    {
        // Attribute removed.
    }
    pIXMLDOMElement->Release();
    pIXMLDOMElement = NULL;
    // Release pIXMLDOMDocument when finished with it.
}
catch(...)
{
    // Release pIXMLDOMDocument if it exists.
    if(pIXMLDOMElement)
        pIXMLDOMElement->Release();
    DisplayErrorToUser();
}
```

Remarks

If the specified attribute has a default value, this is equivalent to a replace
operation: The current value is removed and a new attribute is created with the
default value. This operation also resets the specified property of IXMLDOMNode.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

specified Property
Applies to: IXMLDOMElement
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
removeAttributeNode Method

Removes the specified attribute from this element.

[Script]

Script Syntax

var objXMLDOMAttribute = oXMLDOMElement.removeAttributeNode(DOMAttribute);

Parameters

DOMAttribute
An object that supplies the IXMLDOMAttribute object to be removed from this element.

Return Value

An object. Returns the removed element.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodeId;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    nodeId = nodeBook.getAttributeNode("id");
    alert(nodeBook.attributes.length);
    nodeBook.removeAttributeNode(nodeId);
    alert(nodeBook.attributes.length);
}

[Visual Basic]
Visual Basic Syntax

Set objXMLDOMAttribute = oXMLDOMElement.removeAttributeNode(DOMAttribute)

Parameters

DOMAttribute
   An object that supplies the IXMLDOMAttribute object to be removed from this element.

Return Value

An object. Returns the removed element.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim nodeId As IXMLDOMAttribute
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
   Dim myErr
   Set myErr = xmlDoc.parseError
   MsgBox("You have error " & myErr.reason)
Else
   Set nodeBook = xmlDoc.selectSingleNode("//book")
   Set nodeId = nodeBook.getAttributeNode("id")
   MsgBox nodeBook.Attributes.length
   nodeBook.removeAttributeNode nodeId
   MsgBox nodeBook.Attributes.length
End If

[C/C++]

C/C++ Syntax

HRESULT removeAttributeNode(
   IXMLDOMAttribute *DOMAttribute,
   IXMLDOMAttribute **attributeNode);
Parameters

DOMAttribute [in]
   The DOMAttribute object that is to be removed from this element.
attributeNode [out, retval]
   The removed element.

C/C++ Return Values

S_OK
   The value returned if successful.
E_FAIL
   The value returned if an error occurs.

C/C++ Example

IXMLDOMElement *pIXMLDOMElement = NULL;
_bstr_t bstrAttributeName = _T("dateCreated");
IXMLDOMAttribute *pIXMLDOMAttribute = NULL;
IXMLDOMAttribute *pRemovedIXMLDOMAttribute = NULL;
IXMLDOMDocument *pIXMLDOMDocument = NULL;
HRESULT hr;

try
{
   // Create an instance of DOMDocument and initialize pIXMLDOMDocument
   // Load/create an XML fragment.
   hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
   SUCCEEDED(hr) ? 0 : throw hr;

   if(pIXMLDOMElement)
   {
      hr = pIXMLDOMElement->getAttributeNode(bstrAttributeName, &pIXMLDOMA
      if(SUCCEEDED(hr) && pIXMLDOMAttribute)
      {
         hr = pIXMLDOMElement->removeAttributeNode(pIXMLDOMAttribute, &pRemov
         if(SUCCEEDED(hr))
            // Attribute node removed.
            bResult = TRUE;
            pIXMLDOMAttribute->Release();
      }
      pIXMLDOMElement->Release();
   }
   // Release pIXMLDOMDocument when finished with it.
catch(...) {
    // Release pIXMLDOMDocument if it exists.
    if(pIXMLDOMAttribute)
        pIXMLDOMAttribute->Release();
    if(pIXMLDOMElemnt)
        pIXMLDOMElemnt->Release();
    DisplayErrorToUser();
}

Remarks

If the attribute has a default value, this call also creates a new attribute with the default value, associates the new attribute with this element, and resets the attribute's specified property.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

specified Property | IXMLDOMAttribute

Applies to: IXMLDOMElemnt
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**removeChild Method**

Removes the specified child node from the list of children and returns it.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNode.removeChild(childNode);
```

**Parameters**

- **childNode**
  
  An object. Child node to be removed from the list of children of this node.

**Return Value**

An object. Returns the removed child node.

**Example**

The following script example creates an IXMLDOMNode object (currNode), removes a child node from it, and displays the text of the removed node.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var currNode;
var oldChild;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  root = xmlDoc.documentElement;
  currNode = root.childNodes.item(1);
  oldChild = currNode.removeChild(currNode.childNodes.item(1));
  alert(oldChild.text);
}
```
Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNode.removeChild(childNode)

Parameters

childNode
An object. The child node to be removed from the list of children of this node.

Return Value

An object. Returns the removed child node.

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode object (currNode), removes a child node from it, and displays the text of the removed node.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMElement
Dim currNode As IXMLDOMNode
Dim oldChild As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  Set currNode = root.childNodes.Item(1)
  Set oldChild = currNode.removeChild(currNode.childNodes.Item(1))
  MsgBox oldChild.Text
End If

[C/C++]
C/C++ Syntax

HRESULT removeChild(
    IXMLDOMNode *childNode,
    IXMLDOMNode **outOldChild);

Parameters

*childNode [in]
The child node to be removed from the list of children of this node.

*outOldChild [out, retval]
The removed child node. If Null, the childNode object is not removed.

C/C++ Return Values

S_OK
The value returned if successful.

E_INVALIDARG
The value returned if the oldChild parameter is not a child of this node,
when the specified oldChild is read-only and cannot be removed, or when
oldChild is Null.

E_FAIL
The value returned if an error occurs.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection |
IXMLDOMCharacterData | IXMLDOMComment | DOMDocument |
IXMLDOMDocumentFragment | IXMLDOMDocumentType |
IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference |
IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction |
IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
removeNamedItem Method

Removes an attribute from the collection.

[Script]

Script Syntax

var objXMLDOMNode = oXMLDOMNamedNodeMap.removeNamedItem(name);

Parameters

name
The string specifying the name of the attribute to remove from the collection.

Return Value

An object. Returns the node removed from the collection. Returns Null if the named node is not an attribute.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodeBook = xmlDoc.selectSingleNode("//book");
    alert(nodeBook.attributes.length);
    nodeBook.attributes.removeNamedItem("id");
    alert(nodeBook.attributes.length);
}

[Visual Basic]
Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNodeMap.removeNamedItem(name)

Parameters

name
A string specifying the name of the attribute to remove from the collection.

Return Value

An object. Returns the node removed from the collection. Returns Null if the named node is not an attribute.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    MsgBox nodeBook.Attributes.length
    nodeBook.Attributes.removeNamedItem "id"
    MsgBox nodeBook.Attributes.length
End If

[C/C++]

C/C++ Syntax

HRESULT removeNamedItem(
    BSTR name,
    IXMLDOMNode **namedItem);

Parameters
name [in]
The name of the attribute to remove from the collection.

namedItem [out, retval]
The node removed from the collection. Returns Null if the named node is not an attribute.

C/C++ Return Values

S_OK
The value returned if successful.

S_FALSE
The value when returning Null.

E_FAIL
The value returned if an error occurs.

C/C++ Example

IXMLDOMNode *pIXMLDOMNode = NULL;
IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMap = NULL;
BSTR bstrAttributeName = ::SysAllocString(_T("dateModified"));
IXMLDOMElement *pIXMLDOMElement = NULL;
VARIANT varValue;
HRESULT hr;
IXMLDOMDocument *pIXMLDOMDocument = NULL;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocument
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;

    if(pIXMLDOMElement)
    {
        hr = pIXMLDOMElement->get_attributes(&pIXMLDOMNamedNodeMap);
        if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMap)
        {
            hr = pIXMLDOMNamedNodeMap->removeNamedItem(bstrAttributeName, &pIXMLDOMNode);
            if(SUCCEEDED(hr) && pIXMLDOMNode)
            {
                pIXMLDOMNode->get_nodeValue(&varValue);
                ::MessageBox(NULL, _bstr_t(varValue), _T("Removed Item"), MB_OK);
                pIXMLDOMNode->Release();
                pIXMLDOMNode = NULL;
            }
        }
    }
}
} pIXMLDOMNamedNodeMap->Release();
pIXMLDOMNamedNodeMap = NULL;
}
pIXMLDOMElement->Release();
pIXMLDOMElement = NULL;
::SysFreeString(bstrAttributeName);
bstrAttributeName = NULL;
} catch(...) 
{
 if(bstrAttributeName)
 ::SysFreeString(bstrAttributeName);
 if(pIXMLDOMElement)
 pIXMLDOMElement->Release();
 if(pIXMLDOMNamedNodeMap)
 pIXMLDOMNamedNodeMap->Release();
 if(pIXMLDOMNode)
 pIXMLDOMNode->Release();
 DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMNamedNodeMap
**removeNext Method**

Removes the next node.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = objXMLDOMSelection.removeNext();
```

**Example**

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var oSelection;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    oSelection = xmlDoc.selectNodes("//book");
    while (oSelection.peekNode() != null) {
        oSelection.removeNext();
    }
    alert(xmlDoc.xml);
}
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
Set objXMLDOMNode = objXMLDOMSelection.removeNext
```

**Example**

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim oSelection As IXMLDOMSelection
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
```
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set oSelection = xmlDoc.selectNodes("//book")
    While Not (oSelection.peekNode() Is Nothing)
        oSelection.removeNext
    Wend
    MsgBox xmlDoc.xml
End If

[C/C++]

C/C++ Syntax

HRESULT removeNext(IXMLDOMNode ** ppNode);

Parameters

ppNode [out, retval]
The node that was removed, or Null if there is no nextNode to remove. If the parameter is Null, the removed node is not returned, but is still removed.

C/C++ Return Values

S_OK
The value returned if the method is successful.

S_FALSE
The value returned if no nodes left in the selection.

E_PENDING
The value returned if all nodes cannot be found at this time (in which case no nodes are removed).

Remarks

The removeNext method is equivalent to the following (except that it also works for attributes).
```javascript
var node = list.peekNode();
node.parentNode.removeChild(node);
```

The side effect is that the length of the collection is decremented and the `nextNode` and `item` methods will not return it because it has been removed.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ▼ in the upper-left corner of the page.

**See Also**

Applies to: `IXMLDOMSelection`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
removeQualifiedItem Method

Removes the attribute with the specified namespace and attribute name.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNamedNodeMap.removeQualifiedItem(baseName, namespaceURI);
```

Parameters

`baseName`
The string specifying the base name of the attribute, without namespace qualification.

`namespaceURI`
The string specifying the namespace prefix that qualifies the attribute name.

Return Value

An object. Returns the attribute node removed, or Null if no node was removed.

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook;
xmlDoc.setProperty("SelectionLanguage", "XPath");
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  nodeBook = xmlDoc.selectSingleNode("//book");
  alert(nodeBook.attributes.length);
  nodeBook.attributes.removeQualifiedItem("id", "");
  alert(nodeBook.attributes.length);
}
```
[Visual Basic]

Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNamedNodeMap.removeQualifiedItem(baseName, namespaceURI)

Parameters

baseName
The string specifying the base name of the attribute, without namespace qualification.

namespaceURI
The string specifying the namespace prefix that qualifies the attribute name.

Return Value

An object. Returns the attribute node removed, or Null if no node was removed.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    MsgBox nodeBook.Attributes.length
    nodeBook.Attributes.removeQualifiedItem "id", ""
    MsgBox nodeBook.Attributes.length
End If

[C/C++]

C/C++ Syntax
HRESULT removeQualifiedItem(
    BSTR baseName,
    BSTR namespaceURI,
    IXMLDOMNode **qualifiedItem);

Parameters

baseName [in]
The base name of the attribute, without namespace qualification.
namespaceURI [in]
The namespace prefix that qualifies the attribute name.
qualifiedItem [out, retval]
The attribute node removed, or Null if no node was removed.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value when returning Null.
E_FAIL
The value returned if an error occurs.

C/C++ Example

IXMLDOMNode *pIXMLDOMNode = NULL;
IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMap = NULL;
BSTR bstrAttributeName = ::SysAllocString(_T("dateModified"));
IXMLDOMElement *pIXMLDOMElement = NULL;
VARIANT varValue;
HRESULT hr;
IXMLDOMDocument *pIXMLDOMDocument = NULL;

try
{
// Create an instance of DOMDocument and initialize pIXMLDOMDocument
// Load/create an XML fragment.
hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
SUCCEEDED(hr) ? 0 : throw hr;

if(pIXMLDOMElement)
hr = pIXMLDOMElement->get_attributes(&pIXMLDOMNamedNodeMap);
if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMap)
{
    hr = pIXMLDOMNamedNodeMap->removeQualifiedItem(bstrAttributeName, NULL &pIXMLDOMNode);
    if(SUCCEEDED(hr) && pIXMLDOMNode)
    {
        pIXMLDOMNode->get_nodeValue(&varValue);
        ::MessageBox(NULL, _bstr_t(varValue), _T("Removed Item"), MB_OK);
        pIXMLDOMNode->Release();
        pIXMLDOMNode = NULL;
    }
    pIXMLDOMNamedNodeMap->Release();
    pIXMLDOMNamedNodeMap = NULL;
}
pIXMLElement->Release();
pIXMLElement = NULL;
::SysFreeString(bstrAttributeName);
bstrAttributeName = NULL;
}
catch(...)
{
    if(bstrAttributeName)
        ::SysFreeString(bstrAttributeName);
    if(pIXMLElement)
        pIXMLElement->Release();
    if(pIXMLDOMNamedNodeMap)
        pIXMLDOMNamedNodeMap->Release();
    if(pIXMLDOMNode)
        pIXMLDOMNode->Release();
    DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.

**Remarks**

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ▼ in the upper-left corner of the page.

**See Also**

Applies to: [IXMLDOMNamedNodeMap](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**replaceChild Method**

Replaces the specified old child node with the supplied new child node.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNode = oXMLDOMNode.replaceChild(newChild, oldChild);
```

**Parameters**

- **newChild**
  
  An object. The address of the new child that is to replace the old child. If Null, **oldChild** is removed without a replacement.

- **oldChild**
  
  An object. The address of the old child that is to be replaced by the new child.

**Return Value**

An object. Returns the old child that is replaced.

**Example**

The following example creates a new IXMLElement object, **newElem**, and replaces the specified child node with **newElem**.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var root;
var newElem;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error "+ myErr.reason);
} else {
    root = xmlDoc.documentElement;
```
newElem = xmlDoc.createElement("PAGES");
root.childNodes.item(1).replaceChild(newElem, root.childNodes.item(1).childNodes.item(0));
alert(root.childNodes.item(1).xml);
}

[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNode = oXMLDOMNode.replaceChild(newChild, oldChild)

**Parameters**

*newChild*
An object. The address of the new child that is to replace the old child. If Null, the *oldChild* parameter is removed without a replacement.

*oldChild*
An object. The address of the old child that is to be replaced by the new child.

**Return Value**

An object. Returns the old child that is replaced.

**Example**

The following Microsoft® Visual Basic® example creates a new IXMLDOMNode object, *newElem*, and replaces the specified child node with *newElem*.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim root As IXMLDOMNode
Dim newElem As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load ("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set root = xmlDoc.documentElement
  Set newElem = xmlDoc.createElement("PAGES")
  root.childNodes.Item(1).childNodes.Item(0);
MessageBox root.childNodes.Item(1).xml
End If

[C/C++]

C/C++ Syntax

HRESULT replaceChild(
    IXMLDOMNode *newChild,
    IXMLDOMNode *oldChild,
    IXMLDOMNode **outOldChild);

Parameters

newChild [in]
The address of the new child that is to replace the old child. If Null, the oldChild parameter is removed without a replacement.

oldChild [in]
The address of the old child that is to be replaced by the new child.

outOldChild [out, retval]
The old child that is replaced. If Null, no object is created.

C/C++ Return Values

S_OK
    The value returned if successful.

E_INVALIDARG
    The value returned if the newChild node cannot be inserted as a child of this node, when the specified oldChild is not a child of this node, or if the oldChild parameter is Null.

E_FAIL
    The value returned if an error occurs.

Remarks

This operation depends on the value of the nodeType property.

NODE_ATTRIBUTE
This operation depends on the value of the `newChild` parameter:

NODE_ATTRIBUTE,  
NODE_CDATA_SECTION, NODE_COMMENT,  
NODE_DOCUMENT, NODE_DOCUMENT_TYPE,  
NODE_ELEMENT, NODE_ENTITY,  
NODE_NOTATION,  
NODE_PROCESSING_INSTRUCTION

NODE_DOCUMENT_FRAGMENT

Returns an error. These node types cannot be children of an attribute.

NODE_ENTITY_REFERENCE, NODE_TEXT

Replaces `oldChild` with the children of the document fragment in `newChild` and returns `oldChild`.

Replaces the specified `oldChild` with the supplied `newChild` and returns `oldChild`.

NODE_CDATA_SECTION, NODE_COMMENT, NODE_ENTITY, NODE_NOTATION, NODE_PROCESSING_INSTRUCTION, NODE_TEXT

Returns an error. These node types either cannot have children or their children are read-only.

**NODE_DOCUMENT**

This operation depends on the value of the `newChild` parameter.

NODE_ATTRIBUTE,  
NODE_CDATA_SECTION, NODE_DOCUMENT,  
NODE_ENTITY,  
NODE_ENTITY_REFERENCE,  
NODE_NOTATION,  
NODE_PROCESSING_INSTRUCTION

NODE_COMMENT,  
NODE_PROCESSING_INSTRUCTION

NODE_DOCUMENT_TYPE, NODE_ELEMENT

Returns an error. These nodes are not valid as children of a document node.

Replaces the specified `oldChild` with the supplied `newChild` and returns `oldChild`.

Replaces `oldChild` with `newChild` and returns `oldChild`. By definition, an XML document (the document node) can have only a single child. Therefore, an error is returned if the document node already has a child.

Replaces the specified `oldChild` with
the children of the document fragment (newChild) and returns oldChild. The insert operations are subject to the rules for child nodes and can fail if the document fragment children represent node types that cannot be inserted.

Returns an error. The document type is read-only.

This operation depends on the value of the newChild parameter.

Returns an error. These nodes types are not valid as children of a document fragment.

Replaces the specified oldChild with the supplied newChild and returns oldChild.

Replaces the specified oldChild with the children of the document fragment (newChild) and returns oldChild. The insert operations are subject to the rules for child nodes and can fail if the document fragment children represent node types that cannot be inserted.

Returns an error. Entities and notations are read-only and cannot be inserted into a document.

This operation depends on the value of the newChild parameter.


NODE_ELEMENT, NODE_ENTITY_REFERENCE, NODE_TEXT, NODE_PROCESSING_INSTRUCTION

Replaces the specified oldChild with newChild and returns oldChild.

NODE_ATTRIBUTE, NODE_DOCUMENT, NODE_DOCUMENT_TYPE, NODE_ENTITY, NODE_NOTATION

Returns an error. These node types cannot be children of an element node.

NODE_DOCUMENT_FRAGMENT

Replaces the specified oldChild with the children of the document fragment (newChild) and returns oldChild. The insert operations are subject to the rules for child nodes and can fail if the document fragment children represent node types that cannot be inserted.

NODE_ENTITY_REFERENCE

Returns an error. Although the child nodes of an entity reference are the expanded entity, the children cannot be modified.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

nodeType Property

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
replaceData Method

Replaces the specified number of characters with the supplied string.

[Script]

Script Syntax

oXMLDOMCharacterData.replaceData(offset, count, data);

Parameters

offset
The long integer value specifying the offset, in characters, at which to start replacing string data.

count
The long integer value specifying the number of characters to replace.

data
The string containing the new data that replaces the old string data.

Example

The following script example creates a new IXMLDOMComment object, and then replaces the first five characters with a new string.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    comment = xmlDoc.createComment("Hello World!");
    alert(comment.xml);
    comment.replaceData(0, 5, "Goodbye, Cruel");
    alert(comment.xml);
}
```
[Visual Basic]

**Visual Basic Syntax**

oXMLDOMCharacterData.replaceData(offset, count, data)

**Parameters**

**offset**

The long integer value specifying the offset, in characters, at which to start replacing string data.

**count**

The long integer value specifying the number of characters to replace.

**data**

The string containing the new data that replaces the old string data.

**Example**

The following Microsoft® Visual Basic® example creates a new IXMLDOMComment object, and then replaces the first five characters with a new string.

```vbscript
Dim xmlDoc As New Msxml2.DOMDocument50
Dim comment As IXMLDOMComment
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set comment = xmlDoc.createComment("Hello World!"
    MsgBox comment.xml
    comment.replaceData 0, 5, "Goodbye, Cruel"
    MsgBox comment.xml
End If
```

[C/C++]

**C/C++ Syntax**
HRESULT replaceData(
    long offset,
    long count,
    BSTR data);

Parameters

offset [in]
The offset, in characters, at which to start replacing string data.
count [in]
The number of characters to replace.
data [in]
The new data that replaces the old string data.

C/C++ Return Values

S_OK
The value returned if successful.
S_FALSE
The value when returning Null.
E_FAIL
The value returned if an error occurs.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | IXMLDOMText
reset Method
(IXMLDOMNamedNodeMap)

Resets the iterator.

[Script]

Script Syntax

```
oXMLDOMNamedNodeMap.reset();
```

Example

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodeAttribute, s;
xDoc.setProperty("SelectionLanguage", "XPath");
xDoc.async = false;
xDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  nodeBook = xmlDoc.selectSingleNode("/book");
  nodeBook.attributes.reset();
  nodeAttribute = nodeBook.attributes.nextNode();
  alert(nodeAttribute.text);
}
```

[Visual Basic]

Visual Basic Syntax

```
oXMLDOMNamedNodeMap.reset
```

Example

```
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
```
Dim nodeAttribute As IXMLDOMAttribute
Dim s As String
xmlDoc.setProperty "SelectionLanguage", "XPath"
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    nodeBook.Attributes.Reset
    Set nodeAttribute = nodeBook.Attributes.nextNode()
    MsgBox nodeAttribute.Text
End If

[C/C++]

**C/C++ Syntax**

HRESULT reset(void);

**C/C++ Return Values**

S_OK
   The value returned if successful.

**Remarks**

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: IXMLDOMNamedNodeMap
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
reset Method (IXMLDOMNodeList)

Resets the iterator.

[Script]

Script Syntax

oXMLDOMNodeList.reset();

Example

The following script example creates an IXMLDOMNodeList object and iterates the collection using the nextNode method. It then uses the reset method to reset the iterator to point before the first node in the list.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
var objNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode<>0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  objNodeList = xmlDoc.getElementsByTagName("author");
  for (var i=0; i<objNodeList.length; i++) {
    objNode = objNodeList.nextNode();
    alert(objNode.text);
  }
  objNodeList.reset();
  objNode = objNodeList.nextNode();
  alert(objNode.text);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
oXMLDOMNodeList.reset
```
Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNodeList object and iterates the collection using the nextNode method. It then uses the reset method to reset the iterator to point before the first node in the list.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLDOMNodeList
Dim objNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set objNodeList = xmlDoc.getElementsByTagName("author")
    For i = 0 To (objNodeList.length - 1)
        Set objNode = objNodeList.nextNode
        MsgBox objNode.Text
    Next
    objNodeList.Reset
    Set objNode = objNodeList.nextNode
    MsgBox objNode.Text
End If

[C/C++]

C/C++ Syntax

HRESULT reset(void);

C/C++ Return Values

S_OK
    The value returned if successful.

Remarks

This method reinitializes the iterator to point before the first node in the IXMLDOMNodeList so that the next call to nextNode returns the first item in the list.
This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[nextNode Method](#)

Applies to: IXMLDOMNodeList | IXMLDOMSelection
reset Method
(IXMLDOMParseErrorCollection)

Resets the internal position to start, so that the next method will return the first error in the list.

[Script]

Script Syntax

objIXMLDOMParseErrorCollection.reset();

[Visual Basic]

Visual Basic Syntax

objIXMLDOMParseErrorCollection.reset();

[C/C++]

C/C++ Syntax

HRESULT reset()

Parameters

None.

Return Values

S_OK
    The value returned if successful. This method never fails.

Example
The following example illustrates the reset method. An attempt to validate XML data against an XML Schema results in two errors, because the XML file contains two invalid <book> elements. The code calls next on the resultant error collection object to walk through the collection, then calls the reset method and returns to the first error.

This example uses the same two resource files used in the allErrors example, books.xml and books.xsd. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- **Resource Files (books.xml and books.xsd)**
- **JScript Code (reset.js)**
- **Visual Basic Code (reset.frm)**
- **C/C++ Code (reset.cpp)**
- **Output for the reset Example**

### Applies To
**IXMLDOMParseErrorCollection Interface**

### Versioning
MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ▼ in the upper-left corner of the page.

### See Also
- _newEnum Property |
- length Property |
- item Property |
- next Property
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
JScript Code (reset.js)

var xs, xd;

main();

function main() {
  try {
    xs = new ActiveXObject("MSXML2/XMLSchemaCache.5.0");
    xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
  }
  catch (e) {
    alert("Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office is not installed.
    Download and install MSXML 5.0 from http://msdn.microsoft.com
    before continuing.");
    return;
  }

  try {
    xs.add("urn:books", "books.xsd");
  }
  catch (e) {
    alert("Failed to add to schema cache: " + e.description);
    return;
  }

  try {
    xd.schemas = xs;
    xd.async = false;
    xd.validateOnParse = false;
    xd.load("books.xml");
    xd.setProperty("MultipleErrorMessages", true);
  }
  catch (e) {
    alert("can't load books.xml : " + e.description);
    return;
  }

  var err = xd.validate();
  if (err.errorCode != 0) {
    var allErrs = err.allErrors;
    var e = allErrs.next;
    alert("errorItem: "+e.reason+"\n");
    var e = allErrs.next;
    alert("errorItem: "+e.reason+"\n");
```
allErrs.reset;
var e = allErrs.next;
alert("errorItem (reset): "+e.reason+"\n");
}
else
  alert("valid dom as follows:\n" + xd.xml);
}

function alert(msg)
{
  WScript.echo(msg);
}
```

**Try It!**

1. Copy the XML data ([books.xml](#)), and paste it into a text file. Save the file as **books.xml**.
2. Copy the XSD listing ([books.xsd](#)), and paste it into a text file. Save the file as **books.xsd**, in the same directory where you saved **books.xml**.
3. Copy the JScript listing above, and paste it into a text file. Save the file as **reset.js**, in the same directory where you saved **books.xml** and **books.xsd**.
4. Double click the **reset.js** file from Windows Explorer to launch the application. Alternatively, you can type "reset.js" from a command prompt.

   **Note** Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in [Output for the reset Example](#).
Visual Basic Code (reset.frm)

Dim er As IXMLDOMParseError2
Dim errs As IXMLDOMParseErrorCollection
Dim xd As New DOMDocument50
Dim xs As New XMLSchemaCache50

' Add xs to xd.
x.s.Add "urn:books", App.Path & "\books.xsd"
Set xd.schemas = xs
xd.async = False
xd.validateOnParse = False
xd.setProperty "MultipleErrorMessages", True

' Load the XML file to xd.
xd.Load App.Path & "\books.xml"

' Validate the XML.
Set er = xd.Validate
If er.errorCode <> 0 Then
    Set errs = er.allErrors
    Set er = errs.Next
    msg = "errorItem: " & er.reason
    Set er = errs.Next
    msg = msg & "errorItem: " & er.reason

    ' Reset to examine the first error again.
    errs.Reset
    Set er = errs.Next
    msg = msg & "errorItem (reset): " & er.reason
Else
    msg = "DOM is valid:" & vbCrLf & xd.xml
End If
MsgBox msg

Try It!

1. Copy the XML data (books.xml), and paste it into a text file. Save the file as books.xml.
2. Copy the XSD listing (books.xsd), and paste it into a text file. Save the file as books.xsd, in the same directory where you saved books.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as reset.vbp to the same directory where you saved books.xml and
books.xsd. Name the form file reset.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.
5. Copy the Visual Basic code listing above, and paste it into the form_load subroutine.
6. Execute the code by selecting Start from the Run menu.
7. Verify that your output is the same as that listed in Output for the reset Example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

int main(int argc, char* argv[]) {
    HRESULT hr;
    CoInitialize(NULL);

    IXMLDOMDocument3Ptr pXMLDoc;
    hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XML DOM");
        return -1;
    }
    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_FALSE;
    hr = pXMLDoc->setProperty("MultipleErrorMessages", VARIANT_TRUE);
    if (FAILED(hr)) {
        printf("Failed to enable mulitple validation errors\n");
        return -1;
    }
    if(pXMLDoc->load("books.xml")!=VARIANT_TRUE) {
        printf("Failed to load DOM from books.xml. %s\n",
            (LPCSTR)pXMLDoc->parseError->reason);
        return -1;
    }

    IXMLDOMDocument3Ptr pXSDDoc;
    hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
    if (FAILED(hr)) {
        printf("Failed to CreateInstance on an XSD Doc");
        return -1;
    }
    pXSDDoc->async = VARIANT_FALSE;
    pXSDDoc->validateOnParse = VARIANT_FALSE;
    if(pXSDDoc->load("books.xsd")!=VARIANT_TRUE) {
        printf("Failed to load XSD from books.xsd. %s\n",}
(LPCSTR)pXSDDoc->parseError->reason);
    return -1;

IXMLDOMSchemaCollectionPtr pSCache;
hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr)) {
    printf("Cannot instantiate XMLSchemaCache50\n");
    return -1;
}
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr)) {
    printf("Cannot add 'urn:books' to schema cache.\n");
    return -1;
}

// Validate the entire DOM object.
IXMLDOMParseError2Ptr pError = pXMLDoc->validate();
if (pError->errorCode != 0) {
    IXMLDOMParseErrorCollectionPtr pErrSet;
pErrSet = pError->allErrors;
pError = pErrSet->next;
    printf("errorItem: %s", (LPCSTR)pError->reason);
pError = pErrSet->next;
    printf("errorItem: %s", (LPCSTR)pError->reason);

    // Reset the cursor to revisit the first error item.
    hr = pErrSet->reset();
    if (SUCCEEDED(hr)) {
        pError = pErrSet->next;
        printf("errorItem (reset): %s", (LPCSTR)pError->reason);
    }
} else
    printf("DOM is valid:\n%s\n", (LPCSTR)pXMLDoc->xml);

pXMLDoc.Release();
pXSDDoc.Release();
pSCache.Release();
pError.Release();
CoUninitialize();

return 0;
Try It!

2. From the **File** menu, select **New**. On the **Projects** tab of the **New** dialog box that appears, select **Win32 Console Application** in the left pane. Then type "resetProj" in the **Project name** field. For the project **Location** field, either accept the default setting or choose another location. Click **OK**.
3. The **Win32 Console Application** property page will appear. For the type of Console Application, select **An empty project** and click **Finish**. When the **New Project Information** box displays, click **OK**.
4. Select **FileView** on the project browser, and highlight **resetProj files**. From the **File** menu, select **New**.
5. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "books.xml" in the **File name** text box. Be sure to put quotes around the name of the text file. Otherwise, Visual C++ will generate an unwanted .txt suffix to the XML file.
6. Click **OK**.
7. Copy the XML data file (books.xml), and paste it into the text file you just created.
8. Repeat steps 4-7 for the XSD listing (books.xsd).

   **Note** You can also copy these files into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ listing above (reset.cpp).
10. Build the sample by selecting **Build resetProj.exe** from the **Build** menu.
11. Execute the sample application by selecting **!Execute resetProj.exe** from the **Build** menu.
12. Verify that your output is the same as that listed in **Output for the reset Example**.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
Output for the reset Example

When you build and run the `reset` example, you should get the following output.

errorItem: Element content is invalid according to the DTD/Schema.
Expecting: price.

errorItem: Element content is invalid according to the DTD/Schema.
Expecting: pub_date.

errorItem (reset): Element content is invalid according to the DTD/S
Expecting: price.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
reset Method (IXSLProcessor)

Resets the state of the processor to the state it was in prior to calling the transform method.

[Script]

Script Syntax

```
objXSLProcessor.reset();
```

Example

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xslt.stylesheet = xslDoc;
    var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmlDoc.async = false;
    xmlDoc.load("books.xml");
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    } else {
        xslProc = xslt.createProcessor();
        xslProc.input = xmlDoc;
        xslProc.setParameter("param1", "Hello");
        xslProc.transform();
        alert(xslProc.output);
        xslProc.reset();
        alert(xslProc.output);
    }
}
```

Sample.xsl
The parameter value was: Hello

[Visual Basic]

**Visual Basic Syntax**

```vbnet
objXSLProcessor.reset

**Example**

```vbnet
Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
    xmlDoc.async = False
    xmlDoc.Load "books.xml"
    If (xmlDoc.parseError.errorCode <> 0) Then
        Dim myErr
        Set myErr = xmlDoc.parseError
        MsgBox("You have error " & myErr.reason)
    Else
        Set xslProc = xslt.createProcessor()
        xslProc.input = xmlDoc
        xslProc.addParameter "param1", "Hello"
        xslProc.Transform
        MsgBox xslProc.output
        xslProc.Reset
        MsgBox xslProc.output
    End If
End If
```

**Sample.xsl**
C/C++ Syntax

HRESULT reset();

Remarks

It does not reset any other properties, such as stylesheet or startMode.

The reset method will cause any internally buffered output to be thrown away (that is, it aborts the asynchronous transformation). Calling reset results in the readyState value going from READYSTATE_INTERACTIVE back to READYSTATE_LOADED.

A re-entrant call to reset from inside the scope of a call to transform results in that transform call finishing as soon as it can, returning E_FAIL with the error message "User aborted transform."

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor
save Method

Saves an XML document to the specified location.

[Script]

Script Syntax

oXMLDOMDocument.save(destination);

Parameters

destination
An object. The object can represent a file name, an ASP Response object, a DOMDocument object, or a custom object that supports persistence. See Remarks for more information.

[Visual Basic]

Visual Basic Syntax

oXMLDOMDocument.save(destination)

Parameters

destination
An object. The object can represent a file name, an ASP Response object, a DOMDocument object, or a custom object that supports persistence. See Remarks for more information.

Example

The following Microsoft® Visual Basic® example creates a DomDocument object from a string, then saves the document to a file in the application folder. If you look at the resulting file you will see that, instead of one continuous line of text,
after each tag or data string. That is because of the vbNewLine constant inserted in the string at the appropriate locations.

Private Sub Form_Load()
    Dim doc As New DOMDocument50
    doc.async = False
    doc.validateOnParse = False
    doc.resolveExternals = False

    doc.loadXML _
    "<?xml version='1.0'?>" + vbNewLine + _
    "<doc title='test'>" + vbNewLine + _
    "  <page num='1'>" + vbNewLine + _
    "    <para title='Saved at last'>" + vbNewLine + _
    "      This XML data is finally saved." + vbNewLine + _
    "    </para>" + vbNewLine + _
    "  </page>" + vbNewLine + _
    "  <page num='2'>" + vbNewLine + _
    "    <para>" + vbNewLine + _
    "      This page is intentionally left blank." + vbNewLine
    "    </para>" + vbNewLine + _
    "  </page>" + vbNewLine + _
    "</doc>" + vbNewLine

    Path = App.Path + "\saved.xml"
    doc.save Path
End Sub

[C/C++]

C/C++ Syntax

HRESULT save(
    VARIANT destination);

Parameters

destination [in]
    The type of object to save. This object can represent a file name, an ASP Response object, an XML document object, or a custom object that supports persistence. See Remarks for more information.

C/C++ Return Values
S_OK
The value returned if successful.

XML_BAD_ENCODING
The value returned if the document contains a character that does not belong in the specified encoding. The character must use a numeric entity reference. For example, the Japanese Unicode character 20013 does not fit into the encoding Windows-1250 (the Central European alphabet) and therefore must be represented in markup as the numeric entity reference &amp;#20013; or &amp;#x4E2D; This version of save does not automatically convert characters to the numeric entity references.

E_INVALIDARG
The value returned if a string was provided, but it is not a valid file name.

E_ACCESSDENIED
The value returned if a save operation is not permitted.

E_OUTOFMEMORY
The value returned if the save operation must allocate buffers.

(Other values)
Any other file system error can be returned in the save(string) case.

Example

BOOL DOMDocSaveLocation()
{
    BOOL bResult = FALSE;
    IXMLDOMDocument *pIXMLDOMDocument = NULL;
    HRESULT hr;

    try
    {
        _variant_t varString = _T("D:\sample.xml");
        // Initialize pIXMLDOMDocument (create a DOMDocument).
        // Load document.
        hr = pIXMLDOMDocument-&gt;save(varString);
        if(SUCCEEDED(hr))
            bResult = TRUE;
    }
    catch(...)
    {
        DisplayErrorToUser();
        // Release the IXMLDOMDocument interface.
    }
    // Release the IXMLDOMDocument interface when finished with it.
    return bResult;
### Remarks

The behavior differs based on the object specified by the `objTarget` parameter.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Specifies the file name. This must be a file name rather than a URL. The file is created, if necessary, and the contents are replaced entirely with the saved document. This mode is not intended for use from a secure client, such as Microsoft® Internet Explorer.</td>
</tr>
</tbody>
</table>

```vbscript
dim xmldoc
set xmldoc = Server.CreateObject("Msxml2.DOMDocument.5.0")
xmldoc.load("c:\myfile.xml")
xmldoc.save(Server.MapPath("sample.xml"))
```

The ASP Response object sends the document back to the client that invoked the ASP script.

```vbscript
dim xmldoc
set xmldoc = Server.CreateObject("Msxml2.DOMDocument.5.0")
xmldoc.load(Server.MapPath("sample.xml"))
xmldoc.save(Response)
```

Duplicates the original document. It is the equivalent of saving the document and reparsing it. The document goes through full persistence through XML markup, thereby testing the persistability of your XML document.

```vbscript
<script language="jscript">
    var xmldoc1 = new ActiveXObject("Msxml2.DOMDocument.5.0");
    var xmldoc2 = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmldoc1.load("sample.xml");
    xmldoc1.save(xmldoc2.XMLDocument);
</script>
```

Any other custom COM object that supports `QueryInterface` for `IPersistStream` or `IPersistStreamInit` can also be provided here. This document will be saved accordingly. In the `IStream` case, the `IStream` methods will be called as it saves the document; in the `IPersistStreamLoad` will be called with an `IStream` that supports `Seek`, and `Stat` methods.
External entity references in <DOCTYPE>, <ENTITY>, <NOTATION>, and XML namespace declarations are not changed; they point to the original document. A saved XML document might not load if the URLs are not accessible from the location in which you saved the document.

Character encoding is based on the encoding attribute in the XML declaration, such as <?xml version="1.0" encoding="windows-1252"?>. When no encoding attribute is specified, the default setting is UTF-8.

Validation is not performed during save, which can result in an invalid document that does not load again because of a specified document type definition (DTD).

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Persistence and the DOM

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Reference
selectNodes Method

Applies the specified pattern-matching operation to this node's context and returns the list of matching nodes as IXMLDOMNodeList.

[Script]

**Script Syntax**

```javascript
var objXMLDOMNodeList = oXMLDOMNode.selectNodes(expression);
```

**Parameters**

**expression**
A string specifying an XPath expression.

**Return Value**

An object. Returns the collection of nodes selected by applying the given pattern-matching operation. If no nodes are selected, returns an empty collection.

**Example**

The following script example creates an IXMLDOMNodeList object containing the nodes specified by the `expression` parameter (for example, all the `<xsl:template>` nodes in an XSLT style sheet). It then displays the number of nodes contained in the node list.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var objNodeList;
xmlDoc.async = false;
xmlDoc.load("hello.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xmlDoc.setProperty("SelectionNamespaces", "xmlns:xsl='http://w";
    xmlDoc.setProperty("SelectionLanguage", "XPath");
```

```
objNodeList = xmlDoc.documentElement.selectNodes("//xsl:template")
alert(objNodeList.length);
}

[Visual Basic]

**Visual Basic Syntax**

Set objXMLDOMNodeList = oXMLDOMNode.selectNodes(expression)

**Parameters**

*expression*

A string specifying an XPath expression.

**Return Value**

An object. Returns the collection of nodes selected by applying the given pattern-matching operation. If no nodes are selected, returns an empty collection.

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMNodeList object containing the nodes specified by the *expression* parameter (for example, all the <xsl:template> nodes in an XSLT style sheet). It then displays the number of nodes contained in the node list.

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim objNodeList As IXMLDOMNodeList
xmlDoc.async = False
xmlDoc.Load "hello.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    xmlDoc.setProperty "SelectionNamespaces", "xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
    xmlDoc.setProperty "SelectionLanguage", "XPath"
    Set objNodeList = xmlDoc.documentElement.selectNodes("//xsl:template")
    MsgBox objNodeList.length
End If
```
C/C++ Syntax

HRESULT selectNodes(
    BSTR expression,
    IXMLDOMNodeList **resultList);

Parameters

expression [in]
    A string specifying an XPath expression.
resultList [out, retval]
    The list of nodes selected by applying the given pattern-matching operation.
    If no nodes are selected, returns an empty node list.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the resultList parameter is Null.

Example

The following Microsoft® Visual C++ example creates an IXMLDOMNodeList object containing the nodes specified by the expression parameter (for example, all the <xsl:template> nodes in an XSLT stylesheet). It then displays the number of nodes contained in the node list.

XpathSelectNodes.cpp

#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

void dump_com_error(_com_error &e);

int main(int argc, char* argv[])
{

CoInitialize(NULL);
try{
    IXMLDOMDocument2Ptr pXMLDoc = NULL;
    HRESULT hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));

    // Set parser property settings
    pXMLDoc->async = VARIANT_FALSE;

    // Load the sample XML file
    hr = pXMLDoc->load("hello.xsl");

    // If document does not load report the parse error
    if(hr!=VARIANT_TRUE)
    {
        IXMLDOMParseErrorPtr pError;
        pError = pXMLDoc->parseError;
        _bstr_t parseError = _bstr_t("At line ") + _bstr_t(pError->Getline())
                           + _bstr_t("\\n") + _bstr_t(pError->Getreason());
        MessageBox(NULL, parseError, "Parse Error", MB_OK);
        return 0;
    }
    // Otherwise, build node list using SelectNodes
    // and returns its length as console output
    else
    {
        pXMLDoc->setProperty("SelectionLanguage", "XPath");
        // Set the selection namespace URI if the nodes
        // you wish to select later use a namespace prefix
        pXMLDoc->setProperty("SelectionNamespaces",
                           "xmlns:xsl='http://www.w3.org/1999/XSL/Transform'");
        IXMLDOMElementPtr pXMLDocElement = NULL;
        pXMLDocElement = pXMLDoc->documentElement;
        IXMLDOMNodeListPtr pXMLDomNodeList = NULL;
        pXMLDomNodeList = pXMLDocElement->selectNodes("//xsl:template");
        int count = 0;
        count = pXMLDomNodeList->length;
        printf("The number of <xsl:template> nodes is %i.\n", count);
    }
    catch(_com_error &e)
    {
        dump_com_error(e);
    }
    return 0;
}

void dump_com_error(_com_error &e)
{
    printf("Error\n");
    printf("\a\tCode = %08lx\n", e.Error());
    printf("\a\tCode meaning = %s", e.ErrorMessage());
_bstr_t bstrSource(e.Source());
_bstr_t bstrDescription(e.Description());
printf("\tSource = %s\n", (LPCSTR) bstrSource);
printf("\tDescription = %s\n", (LPCSTR) bstrDescription);
}

**Try It!**

1. Set up a new empty project in Visual C++ for a Win32 console application.
   
   For more information, see [Set Up My Visual C++ Project](#).

2. Copy the **XpathSelectNodes.cpp** file and paste it in as code for a new
   Visual C/C++ source file in your project.

3. Copy the **hello.xsl** file from the Hello, World! (XSLT) tutorial application
   and paste and save it as a resource file in the same folder as your project.

4. Build and execute the project as a console EXE application.

**Results and Output**

When run successfully using the sample code and files, the application will return as standard console output the following:

The number of `<xsl:template>` nodes is 2.

**Remarks**

For more information about using the `selectNodes` method with namespaces,
see the [setProperty Method](#) topic.

The `selectSingleNode` method is similar to the `selectNodes` method, but returns only the first matching node rather than the list of all matching nodes.

`IXMLDOMNodeList` is live and immediately reflects changes to the nodes that appear in the list.

This member is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

**Note** Previously, in MSXML 3.0 and earlier versions, the selection object created by calling the `selectNodes` method would gradually calculate the
node-set. If the DOM tree was modified, while the `selectNodes` call was still actively iterating its contents, the behavior could potentially change the nodes that were selected or returned. In MSXML 4.0 and later, the node-set result is fully calculated at the time of selection. This ensures that the iteration is simple and predictable. In rare instances, this change might impact legacy code written to accommodate previous behavior.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[Using XSLT with DOM or SAX](#) | [selectSingleNode method](#) | [IXMLDOMNodeList](#) | [XPath Syntax](#) | [Patterns](#)

Applies to: [IXMLDOMAttribute](#) | [IXMLDOMCDATASection](#) | [IXMLDOMCharacterData](#) | [IXMLDOMComment](#) | [DOMDocument](#) | [IXMLDOMDocumentFragment](#) | [IXMLDOMDocumentType](#) | [IXMLDOMEElement](#) | [IXMLDOMEntity](#) | [IXMLDOMEntityReference](#) | [IXMLDOMNode](#) | [IXMLDOMNotation](#) | [IXMLDOMProcessingInstruction](#) | [IXMLDOMText](#)
selectSingleNode Method

Applies the specified pattern-matching operation to this node’s context and returns the first matching node.

[Script]

Script Syntax

```javascript
var objXMLDOMNode = oXMLDOMNode.selectSingleNode(queryString);
```

Parameters

**queryString**
A string specifying an XPath expression.

Return Value

An object. Returns the first node that matches the given pattern-matching operation. If no nodes match the expression, returns a null value.

Example

The following script example creates an IXMLDOMNode object and sets it to the first instance of an AUTHOR node with a BOOK parent. It then displays the text of the node.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var currNode;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xmlDoc.setProperty("SelectionLanguage", "XPath");
    currNode = xmlDoc.selectSingleNode("//book/author");
    alert(currNode.text);
```
Visual Basic Syntax

Set objXMLDOMNode = oXMLDOMNode.selectSingleNode(queryString)

Parameters

queryString
   A string specifying an XPath expression.

Return Value

An object. Returns the first node that matches the given pattern-matching operation. If no nodes match the expression, object value is Nothing.

Example

The following Microsoft® Visual Basic® example creates an IXMLDOMNode object and sets it to the first instance of an AUTHOR node with a BOOK parent. It then displays the text of the node.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim currNode As IXMLDOMNode
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
   Dim myErr
   Set myErr = xmlDoc.parseError
   MsgBox("You have error " & myErr.reason)
Else
   xmlDoc.setProperty "SelectionLanguage", "XPath"
   Set currNode = xmlDoc.selectSingleNode("//book/author")
   MsgBox currNode.Text
End If

[C/C++]

C/C++ Syntax
HRESULT selectSingleNode(
    BSTR queryString,
    IXMLDOMNode **resultNode);

**Parameters**

*queryString* [in]
  A string specifying an XPath expression.

*resultNode* [out, retval]
  The first node that is selected by the given pattern-matching operation. If no
  nodes match the expression, returns a null value.

**C/C++ Return Values**

S_OK
  The value returned if successful.

S_FALSE
  The value returned if there is no match.

E_INVALIDARG
  The value returned if the *resultNode* parameter is Null.

**Remarks**

The `selectSingleNode` method is similar to the `selectNodes` method, but
returns only the first matching node rather than the list of all matching nodes.

This member is an extension of the World Wide Web Consortium (W3C)
Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

**See Also**

Using XSLT with DOM or SAX | `selectNodes` Method | `setProperty` Method | XPath Syntax

Applies to: `IXMLDOMAttribute` | `IXMLDOMCDATASection` |
send Method (IXMLHTTPRequest)

Sends an HTTP request to the server and receives a response.

[Script]

Script Syntax

```
oXMLHttpRequest.send(varBody);
```

Parameters

`varBody`

The body of the message being sent with the request.

Example

```
xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
xmlhttp.open("GET", "http://localhost/sample.xml", false);
xmlhttp.send();
alert(xmlhttp.responseTextXML.xml);
```

[Visual Basic]

Visual Basic Syntax

```
oXMLHttpRequest.send(varBody)
```

Parameters

`varBody`

The body of the message being sent with the request.

Example

```
Dim xmlhttp As New Msxml2.xmlhttp
xmlhttp.open "GET", "http://localhost/sample.xml", False
xmlhttp.send
MsgBox xmlhttp.responseXML.xml

[C/C++]

C/C++ Syntax

HRESULT send(VARIANT varBody);

Parameters

varBody [in, optional]
   The body of the message being sent with the request.

C/C++ Return Values

S_OK
   The value returned if successful.

Remarks

This method is synchronous or asynchronous, depending on the value of the bAsync parameter in the open method call. If open is called with bAsync == False, this call does not return until the entire response is received or the protocol stack times out. If open is called with bAsync == True, this call returns immediately.

This method takes one optional parameter, which is the requestBody to use. The acceptable VARIANT input types are BSTR, SAFEARRAY of UI1 (unsigned bytes), IDispatch to an XML Document Object Model (DOM) object, and IStream *. You can use only chunked encoding (for sending) when sending IStream * input types. The component automatically sets the Content-Length header for all but IStream * input types.

If the input type is a BSTR, the response is always encoded as UTF-8. The caller must set a Content-Type header with the appropriate content type and include a charset parameter.

If the input type is a SAFEARRAY of UI1, the response is sent as is without
additional encoding. The caller must set a Content-Type header with the appropriate content type.

If the input type is an XML DOM object, the response is encoded according to the encoding attribute on the <? declaration in the document. If there is no XML declaration or encoding attribute, UTF-8 is assumed.

If the input type is an IStream *, the response is sent as is without additional encoding. The caller must set a Content-Type header with the appropriate content type.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

open Method

Applies to: IXMLHTTPRequest
send Method
(ServerXMLHTTP/IServerXMLHTTP)

Sends an HTTP request to the server and receives a response.

[Script]

Script Syntax

oServerXMLHttpRequest.send(varBody);

Parameters

varBody
The body of the message being sent with the request.

Example

<%@language=JScript%>
<% var srvXmlHttp srvXmlHttp = Server.CreateObject("Msxml2.ServerXMLHTTP.5.0"); srvXmlHttp.open("GET", "http://myserver/myresponse.asp", false); srvXmlHttp.send(); newsElement = srvXmlHttp.responseXML.selectSingleNode("/news/story1" %>

<html>
<body>
<p>Top News Story</p>
<%Response.write(newsElement.text);%>
</body>
</html>

[Visual Basic]

Visual Basic Syntax
oServerXMLHttpRequest.send(varBody)

**Parameters**

*varBody*

The body of the message being sent with the request.

**Example**

```csharp
Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send
MsgBox xmlServerHttp.responseXML.xml
```

**C/C++ Syntax**

```c++
HRESULT send(VARIANT varBody);
```

**Parameters**

*varBody* [in, optional]

The body of the message being sent with the request.

**C/C++ Return Values**

*S_OK*

The value returned if successful.

**Remarks**

This method is synchronous or asynchronous, depending on the value of the bAsync parameter in the open method call. If open is called with bAsync == False, this call does not return until the entire response is received or the protocol stack times out. If open is called with bAsync == True, this call returns immediately.
This method takes one optional parameter, which is the requestBody to use. The acceptable VARIANT input types are BSTR, SAFEARRAY of UI1 (unsigned bytes), IDispatch to an XML Document Object Model (DOM) object, and IStream *. You can only use chunked encoding (for sending) when sending IStream *input types. The component automatically sets the Content-Length header for all but IStream *input types.

If the input type is a BSTR, the response is always encoded as UTF-8. The caller must set a Content-Type header with the appropriate content type and include a charset parameter.

If the input type is a SAFEARRAY of UI1, the response is sent as is without additional encoding. The caller must set a Content-Type header with the appropriate content type.

If the input type is an XMLDOM object, the response is encoded according to the encoding attribute on the <? XML declaration in the document. If there is no XML declaration or encoding attribute, UTF-8 is assumed.

If the input type is an IStream *, the response is sent as is without additional encoding. The caller must set a Content-Type header with the appropriate content type.

**Important** You cannot call multiple send methods for a single open method. Instead, for each send method, you must call a corresponding open method. For example:

```javascript
osrvXm1Http.open()

osrvXm1Http.send()

osrvXm1Http.open()

osrvXm1Http.send()
```

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**
open Method

Applies to: I_ServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
setAttribute Method

Sets the value of the named attribute.

[Script]

Script Syntax

```
oXMLDOMElement.setAttribute(name, value);
```

Parameters

name

The string specifying the name of the attribute. If the attribute with that name already exists, its value is changed. If the attribute with that name does not exist, it is created.

value

The variant that supplies the value for the named attribute.

Example

```
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  nodeBook = xmlDoc.selectSingleNode("//book");
  nodeBook.setAttribute("PublishDate", String(Date()));
  alert(nodeBook.getAttribute("PublishDate"));
}
```

[Visual Basic]

Visual Basic Syntax
oXMLDOMElement.setAttribute(name, value)

**Parameters**

**name**
The string specifying the name of the attribute. If the attribute with that name already exists, its value is changed. If the attribute with that name does not exist, it is created.

**value**
The variant that supplies the value for the named attribute.

**Example**

```plaintext
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error ", & myErr.reason)
Else
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    nodeBook.setAttribute "PublishDate", Now
    MsgBox nodeBook.getAttribute("PublishDate")
End If
```

[C/C++]

**C/C++ Syntax**

```plaintext
HRESULT setAttribute(
    BSTR name,
    VARIANT value);
```

**Parameters**

**name** [in]
The name of the attribute. If an attribute with that name already exists, its value is changed. If an attribute with that name does not exist, it is created.

**value** [in]
The value for the named attribute.

**C/C++ Return Values**

S_OK  
The value returned if successful.
E_FAIL  
The value returned if an error occurs.

**C/C++ Example**

See the example in the `getAttribute` method.

**Remarks**

If an attribute with the supplied name already exists, this method changes its value to the supplied `value` parameter. The supplied string is not parsed, so any markup, such as syntax intended to be recognized as an entity reference, is treated as literal text and must be appropriately escaped by the implementation when it is written out.

To assign an attribute value that contains entity references, the user must create `IXMLDOMAttribute` plus any `IXMLDOMText` and `IXMLDOMEntityReference` objects, build the appropriate subtree, and call the `setAttributeNode` method.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`IXMLDOMAttribute` | `IXMLDOMText` | `IXMLDOMEntityReference` | `setAttributeNode` method

Applies to: `IXMLDOMElement`
setAttributeNode Method

Sets or updates the supplied attribute node on this element.

[Script]

Script Syntax

var objXMLDOMAttribute = oXMLDOMElement.XMLDOMElement(DOMAttribute);

Parameters

DOMAttribute

An object that contains the attribute node to be associated with this element.

Return Value

An object. Returns Null unless the new attribute replaces an existing attribute with the same name, in which case this method returns the previous, replaced attribute node.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodePublishDate;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodePublishDate = xmlDoc.createAttribute("PublishDate");
    nodePublishDate.value = String(Date());
    nodeBook = xmlDoc.selectSingleNode("//book");
    nodeBook.setAttributeNode(nodePublishDate);
    alert(nodeBook.getAttribute("PublishDate"));
}

[Visual Basic]
Visual Basic Syntax

Set objXMLDOMAttribute = oXMLDOMElement.XMLDOMElement(DOMAttribute)

Parameters

DOMAttribute
An object that contains the attribute node to be associated with this element.

Return Value

An object. Returns Null unless the new attribute replaces an existing attribute with the same name, in which case this method returns the previous, replaced attribute node.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim nodePublishDate As IXMLDOMAttribute
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodePublishDate = xmlDoc.createAttribute("PublishDate")
    nodePublishDate.Value = Now
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    nodeBook.setAttributeNode nodePublishDate
    MsgBox nodeBook.getAttribute("PublishDate")
End If

[C/C++]

C/C++ Syntax

HRESULT setAttributeNode(
    IXMLDOMAttribute *DOMAttribute,
    IXMLDOMAttribute **attributeNode);
Parameters

DOMAttribute [in]
   An attribute node that is to be associated with this element.
attributeNode [out, retval]
   Null unless the new attribute replaces an existing attribute with the same
   name, in which case this method returns the previous, replaced attribute
   node.

C/C++ Return Values

S_OK
   The value returned if successful.
E_FAIL
   The value returned if an error occurs.

C/C++ Example

See the example in the getAttributeNode method.

Remarks

You cannot add an existing attribute to an element until you first remove it from
its previous element. Also, you cannot add a namespace-qualified attribute when
it uses the same prefix as another attribute with a different namespaceURI.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

namespaceURI Property

Applies to: IXMLDOMElement
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
setNamedItem Method

Adds the supplied node to the collection.

[Script]

Script Syntax

var objXMLDOMNode = oIXMLDOMNamedNodeMap.setNamedItem(newItem);

Parameters

newItem

The object containing the attribute to be added to the collection.

Return Value

An object. Returns the attribute successfully added to the collection.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeBook, nodePublishDate;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    nodePublishDate = xmlDoc.createAttribute("PublishDate");
    nodePublishDate.value = String(Date());
    nodeBook = xmlDoc.selectSingleNode("//book");
    nodeBook.attributes.setNamedItem(nodePublishDate);
    alert(nodeBook.getAttribute("PublishDate"));
}

[Visual Basic]

Visual Basic Syntax
Set objXMLDOMNode = oIXMLDOMNamedNodeMap.setNamedItem(newItem)

**Parameters**

*newItem*

The object containing the attribute to be added to the collection.

**Return Value**

An object. Returns the attribute successfully added to the collection.

**Example**

```vbnet
Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeBook As IXMLDOMElement
Dim nodePublishDate As IXMLDOMAttribute
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodePublishDate = xmlDoc.createAttribute("PublishDate")
    nodePublishDate.Value = Now
    Set nodeBook = xmlDoc.selectSingleNode("//book")
    nodeBook.Attributes.setNamedItem nodePublishDate
    MsgBox nodeBook.getAttribute("PublishDate")
End If
```

**C/C++ Syntax**

```c
HRESULT setNamedItem(
    IXMLDOMNode *newItem,
    IXMLDOMNode **nameItem);
```

**Parameters**

*newItem* [in]

An attribute to be added to the collection.
nameItem [out, retval]

An attribute successfully added to the collection. If Null, no object is created.

C/C++ Return Values

S_OK
The value returned if successful.

E_INVALIDARG
The value returned if the newItem parameter is Null.

E_FAIL
The value returned if an error occurs.

C/C++ Example

HRESULT hr;
IXMLDOMDocument *pIXMLDOMDocument = NULL;
IXMLDOMNode *pIXMLDOMNode = NULL;
IXMLDOMNamedNodeMap *pIXMLDOMNamedNodeMap = NULL;
BSTR bstrAttributeName = ::SysAllocString(_T("dateModified"));
IXMLDOMAttribute *pIXMLDOMAttribute = NULL;
IXMLDOMElement *pIXMLDOMElement = NULL;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocument.
    // Load/create an XML fragment.
    hr = m_pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;

    if(pIXMLDOMElement)
    {
        hr = pIXMLDOMElement->get_attributes(&pIXMLDOMNamedNodeMap);
        if(SUCCEEDED(hr) && pIXMLDOMNamedNodeMap)
        {
            hr = m_pIXMLDOMDocument->createAttribute(bstrAttributeName,
                &pIXMLDOMAttribute);
            if(SUCCEEDED(hr) && pIXMLDOMAttribute)
            {
                hr = pIXMLDOMAttribute->put_nodeValue(_variant_t(_T("year 2000")));
                hr = pIXMLDOMNamedNodeMap->setNamedItem(pIXMLDOMAttribute);
                if(SUCCEEDED(hr) && pIXMLDOMNode)
                {
                    pIXMLDOMNode->Release();
                    pIXMLDOMNode = NULL;
                }
            }
        }
    }
}
Remarks

If an attribute already exists with the name in IXMLDOMNode, the supplied replaces the existing attribute. The attribute name appears in its IXMLDOMNode property.

If the newItem node type is not NODE_ATTRIBUTE, setNamedItem returns an error. For example, it is not possible to modify entities or notations, which are read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode
Applies to: IXMLDOMNamedNodeMap
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
**setOption Method**

Sets one of the following options:

- SXH_OPTION_URL_CODEPAGE
- SXH_OPTION_ESCAPE_PERCENT_IN_URL
- SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS
- SXH_OPTION_SELECT_CLIENT_SSL_CERT

For more information about these options, see **Remarks**.

**[Script]**

**Script Syntax**

```javascript
oServerXMLHTTPRequest.setOption(option, value);
```

**Parameters**

- `option`  
  The option whose value is to be set.
- `value`  
  The value to which the specified option is to be set.

**[Visual Basic]**

**Visual Basic Syntax**

```vbscript
oServerXMLHTTPRequest.setOption(option, value)
```

**Parameters**

- `option`  
  The option whose value is to be set.
- `value`  
  The value to which the specified option is to be set.
C/C++ Syntax

HRESULT setOption(SERVERXMLHTTP_OPTION option, VARIANT value);

Parameters

option [in]
The option whose value is to be set.

value [in]
The value to which the specified option is to be set.

C/C++ Return Values

S_OK
Value returned if successful.

Remarks

The following table lists the options available to the setOption method.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXH_OPTION_URL_CODEPAGE</td>
<td>By default, CP_UTF8 is the code page used to convert the Unicode URL string (specified in the open method) to a single-byte representation.</td>
</tr>
<tr>
<td>0</td>
<td>The SXH_OPTION_URL_CODEPAGE option allows the client to override this default with a different code page value. The client should specify an unsigned integer value for the code page.</td>
</tr>
<tr>
<td>SXH_OPTION_ESCAPE_PERCENT_IN_URL</td>
<td>By default, escaping unsafe ANSI characters in the URL (for example, &quot;</td>
</tr>
</tbody>
</table>
> "%20") does not escape the % character itself.

The SXH_OPTION_ESCAPE_PERCENT_IN_URL option allows the client to change this behavior. The client should specify a Boolean True/False value for this option.

SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS

The SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS option is a DWORD mask of various flags that can be set to change this default behavior. The default value is to ignore all problems. You must set this option before calling the send method. The flags are as follows:

SXH_SERVER_CERT_IGNORE_UNKNOWN_CA = 256

Unknown certificate authority

SXH_SERVER_CERT_IGNORE_WRONG_USAGE = 512

Malformed certificate such as a certificate with no subject name.

SXH_SERVER_CERT_IGNORE_CERT_CN_INVALID = 4096

Mismatch between the visited hostname and the certificate name being used on the server.

SXH_SERVER_CERT_IGNORE_CERT_DATE_INVALID = 8192

The date in the certificate is invalid or has expired.

SXH_SERVER_CERT_IGNORE_ALL_SERVER_ERRORS = 13056

All certificate errors.

To turn off a flag, you subtract it from the default value, which is the sum of all flags. For example, to catch an invalid date in a certificate, you turn off the SXH_SERVER_CERT_IGNORE_CERT_DATE_INVALID flag as follows:

shx.setOption(2, shx.getOption(2) _
               - SXH_SERVER_CERT_IGNORE_CERT_DATE_INVALID)

SXH_OPTION_SELECT_CLIENT_SSL_CERT
By default, the value of this option is an empty string (""), which means pick the first certificate in the local store to send if the server requests a client certificate.

The SXH_OPTION_SELECT_CLIENT_SSL_CERT option is a string that lets you select which client certificate from the local store should be sent. You must set this option before calling the send method. The following example sets the client certificate option to request the client certificate named "MSXML":

```
shx.setOption(3, "MSXML")
```

**Note** Option -1, SXH_OPTION_URL, is a read-only return value, and can therefore not be set by the setOption method. For more information about this option, see [getOption Method](#).

**See Also**

[open Method](#)

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
setProperty Method

This method is used to set second-level properties on the DOM object.

[Script]

Script Syntax

objXMLDOMDocument2.setProperty(name, value);

Parameters

name
The name of the property to be set. For a list of properties that can be set using this method, see second-level properties.

value
The value of the specified property. For a list of property values that can be set using this method, see second-level properties.

Example

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var selection;
xmlDoc.loadXML("<Customer><Name>Microsoft</Name></Customer>");
xmlDoc.setProperty("SelectionLanguage", "XPath");
selection = xmlDoc.selectNodes("Customer/Name");
alert(selection.expr + " -- " + selection.item(0).xml);

The preceding message box displays "Customer/Name --
<Name>Microsoft</Name>".

selection.expr = "/Customer";
alert(selection.expr + " -- " + selection.item(0).xml);

The preceding message box displays "/Customer --<Customer>
<Name>Microsoft</Name></Customer>".

[Visual Basic]
Visual Basic Syntax

```vbnet
objXMLDOMDocument2.setProperty(name, value)
```

**Parameters**

**name**
The name of the property to be set. For a list of properties that can be set using this method, see *second-level properties*.

**value**
The value of the specified property. For a list of property values that can be set using this method, see *second-level properties*.

**Example**

```vbnet
Dim xmldoc As New Msxml2.DOMDocument50
Dim selection As Msxml2.IXMLDOMSelection

xmldoc.loadXML ("<Customer><Name>Microsoft</Name></Customer>")
xmldoc.setProperty "SelectionLanguage", "XPath"
Set selection = xmldoc.selectNodes("Customer/Name")
MsgBox selection.expr + " -- " + selection.Item(0).xml

The preceding message box displays "Customer/Name -- <Name>Microsoft</Name>".

selection.expr = "/Customer"
MsgBox selection.expr + " -- " + selection.Item(0).xml

The preceding message box displays "/Customer --<Customer> <Name>Microsoft</Name></Customer>".
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT setProperty (BSTR name, VARIANT value);
```

**Parameters**
**name** [in]

The name of the property to be set. For a list of properties that can be set using this method, see *second-level properties*.

**value** [in]

The value of the specified property. For a list of property values that can be set using this method, see *second-level properties*.

### C/C++ Return Values

- **S_OK**
  - Value returned if successful.
- **E_FAIL**
  - Value returned if name or value is invalid.

### Remarks

Examples of the second-level properties include `SelectionLanguage`, `ValidateOnParse`, `ServerHTTPRequest`.

This method may not be applied to the first-level DOM properties that are exposed directly on a DOM object. Examples of the first-level properties include `validateOnParse`, `async`, `parseError`, etc.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

### See Also

- `selectSingleNode Method`
- `selectNodes Method`
- `document Function`
- `second-level properties`
- `getProperty Method`

Applies to: `IXMLDOMDocument2`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
setProxy Method
(ServerXMLHTTP/IServerXMLHTTP)

Specify proxy configuration.

[Script]

Script Syntax

oSrvXMLHTTPRequest.setProxy(proxySetting, varProxyServer, varBypassList);

Parameters

proxySetting
  The proxy configuration whose value is to be set. For a list of available settings and a description of how to use those settings, see Remarks.

varProxyServer
  The name of a proxy server or a list of proxy server names.

varBypassList
  The list of locally known host names or IP addresses for which you want to permit bypass of the proxy server.

Example

No examples available at this time.

[Visual Basic]

Visual Basic Syntax

oSrvXMLHTTPRequest.setProxy(proxySetting, varProxyServer, varBypassList);

Parameters
proxySetting
The proxy configuration whose value is to be set. For a list of available settings and a description of how to use those settings, see Remarks.

varProxyServer
The name of a proxy server or a list of proxy server names.

varBypassList
The list of locally known host names or IP addresses for which you want to permit bypass of the proxy server.

[C/C++]

C/C++ Syntax

HRESULT setProxy(
    SXH_PROXY_SETTING proxySetting,
    VARIANT varProxyServer,
    VARIANT varBypassList);

Parameters

proxySetting [in]
The proxy configuration whose value is to be set. For a list of available settings and a description of how to use those settings, see Remarks.

varProxyServer [in, optional]
The name of a proxy server or a list of proxy server names.

varBypassList [in, optional]
The list of locally known host names or IP addresses for which you want to permit bypass of the proxy server.

C/C++ Return Values

S_OK
Value returned if successful.

E_FAIL
Value returned if name or value is invalid.

Remarks
The following table lists the settings available for use with the `setProxy` method.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXH_PROXY_SET_PRECONFIG</td>
<td>The <code>SXH_PROXY_SET_PRECONFIG</code> option can be used to specify that a previously established static proxy configuration should be used. When this option is used, the configuration is taken from the Windows registry. This is the case if you used the WinHTTP proxy configuration utility, proxycfg.exe, to set proxy settings at the client computer. Currently, the <code>SXH_PROXY_SET_DEFAULT</code> shares the same settings level as <code>SXH_PROXY_SET_PRECONFIG</code>. It is intended, however, to support auto-proxy configuration features that might be included with future released products.</td>
</tr>
<tr>
<td>SXH_PROXY_SET_DEFAULT</td>
<td></td>
</tr>
<tr>
<td>SXH_PROXY_SET_DIRECT</td>
<td>The <code>SXH_PROXY_SET_DIRECT</code> option can be used to specify that all HTTP and HTTPS servers should be accessed directly.</td>
</tr>
<tr>
<td>SXH_PROXY_SET_PROXY</td>
<td>The <code>SXH_PROXY_SET_PROXY</code> option can be used to specify one or more proxy servers, and an optional bypass list. If a proxy is not specified for a given protocol, and the server is not in the bypass list, the server cannot be accessed.</td>
</tr>
</tbody>
</table>

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

`setProxyCredentials Method`
Applies to: IServerXMLHTTPRequest/ServerXMLHTTP Members
setProxyCredentials Method (ServerXMLHTTP/IServerXMLHTTP)

Specify proxy authentication credentials.

[Script]

Script Syntax

oSrvXMLHTTPRequest.setProxyCredentials(username, password);

Parameters

username
The name of the user to be authenticated.
password
The password for the user to be authenticated.

[Visual Basic]

Visual Basic Syntax

oSrvXMLHTTPRequest.setProxyCredentials(username, password)

Parameters

username
The name of the user to be authenticated.
password
The password for the user to be authenticated.

[C/C++]

C/C++ Syntax
HRESULT setProxyCredentials(
    BSTR bstrUserName,
    BSTR bstrPassword);

**Parameters**

*bstrUserName* [in]
   The name of the user to be authenticated.

*bstrPassword* [in]
   The password for the user to be authenticated.

**C/C++ Return Values**

S_OK
   Value returned if successful.

E_FAIL
   Value returned if name or value is invalid.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

*[setProxy Method]*

Applies to: *IServerXMLHTTPRequest* / *ServerXMLHTTP Members*
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
setRequestHeader Method
(IXMLHTTPRequest)

Specifies the name of an HTTP header.

[Script]

Script Syntax

oXMLHttpRequest.setRequestHeader(bstrHeader, bstrValue);

Parameters

bstrHeader
A string. A header name to set; for example, "depth". This parameter should not contain a colon and should be the actual text of the HTTP header.

bstrValue
A string. The value of the header; for example, "infinity".

Note You must call the open method before you call this method. Otherwise, an error will occur.

Example

The following script example posts a DOMDocument to an Active Server Page (ASP) on a server and returns the result as a new XML document.

HTML File (form.htm)

<HTML>
<HEAD>
<SCRIPT LANGUAGE="VBScript">
Function onLoad()
    Dim mydata, pi
    Set mydata = CreateObject("Msxml2.DOMDocument.5.0")
Set pi = mydata.createProcessingInstruction("xml", "version='1.0'
mydata.insertBefore pi, mydata.firstChild
End function
Function sendInfo()
    Dim MyHttp
    'Do validation of input data before sending it.
    If(Not(customerName.value = "")) then
        With MyData.documentElement
            .getElementsByTagName("Name").item(0).text = customerName.v
            .getElementsByTagName("Phone").item(0).text = customerPhone
        End With
    Set MyHttp=CreateObject("Msxml2.XMLHTTP.5.0")
    MyHttp.open "POST", "http://localhost/httpreqserver.asp", Fals
    'Simulate message sent by a custom user agent.
    MyHttp.setRequestHeader "User-Agent", "MyCustomUser"
    MyData.async = False
    MyHttp.send mydata.XMLDocument
    Document.Write MyHttp.responseText
    Else
        Document.Write "Invalid data."
    End If
End function
</SCRIPT>
</HEAD>
<BODY LANGUAGE="JScript" ONLOAD="Return onLoad()">
<TABLE BORDER="2" ALIGN="center">
    <TR><TD WIDTH="150" ALIGN="center">
        <LABEL>Name</LABEL>
    </TD><TD>
        <INPUT NAME="customerName" TYPE="EDIT"/>
    </TD></TR>
    <TR><TD WIDTH="150" align="CENTER">
        <LABEL>Telephone number</LABEL>
    </TD><TD>
        <INPUT NAME="customerPhoneNum" type="EDIT"/>
    </TD></TR>
</TABLE>
<XML id="MyData">
<MyStructure>
    <Name/>
    <Phone/>
</MyStructure>
ASP File (httpreqserver.asp)

<%@LANGUAGE="Jscript"%>
<%
    Response.Expires = -1000;
    // Load the posted XML document.
    var doc = Server.CreateObject("Msxml2.DOMDocument.5.0");
    doc.async=false;
    doc.load(Request);
    var result = Server.CreateObject("Msxml2.DOMDocument.5.0");
    // Now process the order and build the result document.
    var userAgent = Request.ServerVariables("HTTP_User-Agent");
    var OutputString="Data for " +
    doc.documentElement.childNodes.item(0).text +
    "(" + doc.documentElement.childNodes.item(1).text +
    ") added";
    Response.ContentType = "text/xml";
    if(userAgent == "MyCustomUser")
    {
        result.loadXML("<result>" + OutputString + "</result>");
        var pi = result.createProcessingInstruction("xml", "version='1
        result.insertBefore( pi, result.firstChild);
        result.save(Response);
    }
    else
    {
        Response.Write("<P><B>" + OutputString+"</B></P>");
    }
%>

Try It!

To run this sample, you need access to a computer running Internet Information Services (IIS) 5.0 or later.

1. Copy the HTML code provided above, and paste it into Notepad.
2. Save the file a form.htm to a valid Web virtual directory, such as C:\Inetpub\wwwroot, on your Web server computer.

   **Note** If you are not running IIS locally on your computer, locate the following line:

   MyHttp.open "POST", "http://localhost/httpreqserver.asp", False
Substitute the name of the remote computer for the characters "localhost", and save the file.

3. Copy the ASP code provided above, and paste it into Notepad.
4. Save the file as \texttt{httpreqserver.asp}, in the same Web virtual directory you used in Step 1.
5. Open your browser to the Web URL location where you saved the sample HTML file, such as http://localhost/form.htm.
6. Enter the name and phone number information, and click \textbf{Send Information} to add/submit the XML to the ASP request server page.

\textbf{Output}

When run, the sample Web application should return as output in the browser the name and phone number you entered as input. To verify that this information was generated using the XML <\texttt{result}> node string from \texttt{httpreqserver.asp}, you can do the following:

1. From within Internet Explorer, click \textbf{View}, and then click \textbf{Source}.
2. You can observe that the source matches the string that was generated by the ASP request server page in this format:

\begin{verbatim}
<?xml version="1.0"?>
<result>Data for [name input] ([phone number input]) added.</result>
\end{verbatim}

\textbf{[Visual Basic]}

\textbf{Visual Basic Syntax}

\texttt{oXMLHttpRequest.setRequestHeader(bstrHeader, bstrValue)}

\textbf{Parameters}

\textit{bstrHeader}
A string. A header name to set; for example, "depth". This parameter should not contain a colon and should be the actual text of the HTTP header.

\textit{bstrValue}
A string. The value of the header; for example, "infinity".
C/C++ Syntax

HRESULT setRequestHeader(BSTR bstrHeader, BSTR bstrValue);

Parameters

bstrHeader [in]
A header name to set; for example, "depth". This parameter should not contain a colon and should be the actual text of the HTTP header.

bstrValue [in]
The value of the header; for example, "infinity".

C/C++ Return Values

S_OK
The value returned if successful.

Remarks

If another header already exists with this name, it is replaced.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

getResponseHeader Method

Applies to: IXMLHTTPRequest
setRequestHeader Method
(ServerXMLHTTP/IServerXMLHTTP)

Specifies the name of an HTTP header.

Remarks
If another header already exists with this name, it is replaced.

[Script]

Script Syntax

```javascript
oXMLServerHTTPRequest.setRequestHeader(bstrHeader, bstrValue);
```

Parameters

`bstrHeader`
The header name to set; for example, "depth". This parameter should not contain a colon and should be the actual text of the HTTP header.

`bstrValue`
The value of the header; for example, "infinity".

[Visual Basic]

Visual Basic Syntax

```vbnet
oXMLServerHTTPRequest.setRequestHeader(bstrHeader, bstrValue)
```

Parameters

`bstrHeader`
The header name to set; for example, "depth". This parameter should not contain a colon and should be the actual text of the HTTP header.
**bstrValue**
The value of the header; for example, "infinity".

[C/C++]

**C/C++ Syntax**

HRESULT setRequestHeader(BSTR bstrHeader, BSTR bstrValue);

**Parameters**

*bstrHeader* [in]
The header name to set; for example, "depth". This parameter should not contain a colon and should be the actual text of the HTTP header.

*bstrValue* [in]
The value of the header; for example, "infinity".

**C/C++ Return Values**

S_OK
The value returned if successful.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

getResponseHeader Method

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
setStartMode Method

Performs subsets of larger XSL Transformations (XSLT) by selecting the XSLT mode with which to start. This minimizes the amount of XSLT processing.

The default value of the start mode is the empty string, "".

Example

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
xslt.stylesheet = xslDoc;
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
xslProc = xslt.createProcessor();
xslProc.input = xmlDoc;
xslProc.setStartMode("view");
xslProc.transform();
alert(xslProc.output);
```

File Name: Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>
```

[Script]
**Script Syntax**

```javascript
objXSLProcessor.setStartMode(mode, namespaceURI);
```

**Parameters**

*mode*

The desired mode as a string. It must be the base name part of the qualified name.

*namespaceURI (optional)*

The full namespace URI to fully qualify the start mode name.

[Visual Basic]

**Visual Basic Syntax**

```vbnet
objXSLProcessor.setStartMode(mode, namespaceURI)
```

**Parameters**

*mode*

The desired mode as a string. It must be the base name part of the qualified name.

*namespaceURI (optional)*

The full namespace URI to fully qualify the start mode name.

**Example**

```vbnet
Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslt.stylesheet = xslDoc
```
xmlDoc.async = False
xmlDoc.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set xslProc = xslt.createProcessor()
    xslProc.input = xmlDoc
    xslProc.setStartMode "view"

    xslProc.Transform
    MsgBox xslProc.output
End If
End If

File Name: Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
    <xsl:output method="html"/>
    <xsl:param name="param1"/>
    <xsl:template match="/">
        Hello
    </xsl:template>
    <xsl:template match="/" mode="edit">
        In Edit Mode
    </xsl:template>
    <xsl:template match="/" mode="view">
        In View Mode
    </xsl:template>
</xsl:stylesheet>

[C/C++]

C/C++ Syntax

HRESULT setStartMode(BSTR mode, BSTR namespaceURI);

Parameters

mode [in]
    The desired mode as a string. It must be the base name part of the qualified name.

namespaceURI [in, optional]
    The full namespace URI to fully qualify the start mode name.
C/C++ Return Values

S_OK
   Success
E_FAIL
   The value returned if the value of the readyState property is
   READYSTATE_INTERACTIVE.
E_INVALIDARG
   The value returned if the mode base name contains a colon character or is
   an invalid name.

Remarks

Using setStartMode is essentially the same as an XSLT style sheet that starts
with the following rule.

<xsl:template match="/">
   <xsl:apply-templates select="*" mode="{mode}"/>
</xsl:template>

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor

Other Resources

W3C Qualified Names section of the XSLT Version 1.0 W3C Recommendation |
W3C Namespaces in XML Recommendation
setTimeouts Method

Specifies timeout settings for resolving the domain name, establishing the connection to the server, sending the data, and receiving the response. The timeout parameters of the setTimeouts method are specified in milliseconds, so a value of 1000 would represent 1 second. A value of zero represents an infinite timeout. There are four separate timeout parameters: `resolveTimeout`, `connectTimeout`, `sendTimeout`, and `receiveTimeout`. When calling the setTimeouts method, all four values must be specified. The timeouts are applied at the Winsock layer.

[Script]

Script Syntax

```javascript
oServerXMLHTTPRequest.setTimeouts(resolveTimeout, connectTimeout, sendTimeout, receiveTimeout);
```

Parameters

`resolveTimeout`
A long integer. The value is applied to mapping host names (such as "www.microsoft.com") to IP addresses; the default value is infinite, meaning no timeout.

`connectTimeout`
A long integer. The value is applied to establishing a communication socket with the target server, with a default timeout value of 60 seconds.

`sendTimeout`
A long integer. The value applies to sending an individual packet of request data (if any) on the communication socket to the target server. A large request sent to a server will normally be broken up into multiple packets; the send timeout applies to sending each packet individually. The default value is 5 minutes.

`receiveTimeout`
A long integer. The value applies to receiving a packet of response data from the target server. Large responses will be broken up into multiple
packets; the receive timeout applies to fetching each packet of data off the socket. The default value is 60 minutes.

**Example**

```javascript
var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
var lResolve = 5 * 1000;
var lConnect = 5 * 1000;
var lSend = 15 * 1000;
var lReceive = 15 * 1000;
xmlServerHttp.setTimeouts(lResolve, lConnect, lSend, lReceive);
xmlServerHttp.open("GET", "http://localhost/sample.xml", false);
xmlServerHttp.send();
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
oServerXMLHTTPRequest.setTimeouts(resolveTimeout, connectTimeout, sendTimeout, receiveTimeout)
```

**Parameters**

**resolveTimeout**

A long integer. The value is applied to mapping host names (such as "www.microsoft.com") to IP addresses; the default value is infinite, meaning no timeout.

**connectTimeout**

A long integer. The value is applied to establishing a communication socket with the target server, with a default timeout value of 60 seconds.

**sendTimeout**

A long integer. The value applies to sending an individual packet of request data (if any) on the communication socket to the target server. A large request sent to a server will normally be broken up into multiple packets; the send timeout applies to sending each packet individually. The default value is 5 minutes.

**receiveTimeout**

A long integer. The value applies to receiving a packet of response data from the target server. Large responses will be broken up into multiple packets; the receive timeout applies to fetching each packet of data off the
socket. The default value is 60 minutes.

Example

Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
Dim lResolve, lConnect, lSend, lReceive As Long
lResolve = 5 * 1000
lConnect = 5 * 1000
lSend = 15 * 1000
lReceive = 15 * 1000
xmlServerHttp.setTimeouts lResolve, lConnect, lSend, lReceive
xmlServerHttp.open "GET", "http://localhost/sample.xml", False
xmlServerHttp.send

[C/C++]

C/C++ Syntax

HRESULT setTimeouts (long resolveTimeout, long connectTimeout,
 long sendTimeout, long receiveTimeout)

Parameters

resolveTimeout [in]
A long integer. The value is applied to mapping host names (such as
"www.microsoft.com") to IP addresses; the default value is infinite,
meaning no timeout.

connectTimeout [in]
A long integer. The value is applied to establishing a communication socket
with the target server, with a default timeout value of 60 seconds.

sendTimeout [in]
A long integer. The value applies to sending an individual packet of request
data (if any) on the communication socket to the target server. A large
request sent to a server will normally be broken up into multiple packets;
the send timeout applies to sending each packet individually. The default
value is 5 minutes.

receiveTimeout [in]
A long integer. The value applies to receiving a packet of response data
from the target server. Large responses will be broken up into multiple
packets; the receive timeout applies to fetching each packet of data off the
socket. The default value is 60 minutes.

C/C++ Return Values

S_OK
   Value returned if successful.

Remarks

The setTimeouts method should be called before the open method. None of the parameters is optional.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

open Method

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
splitText Method

Splits this text node into two text nodes at the specified offset and inserts the new text node into the tree as a sibling that immediately follows this node.

[Script]

Script Syntax

```javascript
var objXMLDOMText = oXMLDOMText.splitText(offset);
```

Parameters

**offset**

A long integer. The number of characters at which to split this text node into two nodes, starting from zero.

Return Value

An object. Returns the new text node.

Example

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var nodeRoot, nodeText, newNodeText;
xmlDoc.async = false;
xmlDoc.loadXML("<root>Hello World!</root>");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  nodeRoot = xmlDoc.documentElement;
  nodeText = nodeRoot.firstChild;
  alert(nodeRoot.childNodes.length);
  newNodeText = nodeText.splitText(6);
  alert(nodeRoot.childNodes.length);
}
```

[Visual Basic]
Visual Basic Syntax

Set objXMLDOMText = oXMLDOMText.splitText(offset)

Parameters

offset
A long integer. The number of characters at which to split this text node into two nodes, starting from zero.

Return Value

An object. Returns the new text node.

Example

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeRoot As IXMLDOMElement
Dim nodeText As IXMLDOMText
Dim newNodeText As IXMLDOMText
xmlDoc.async = False
xmlDoc.loadXML "<root>Hello World!</root>"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set nodeRoot = xmlDoc.documentElement
    Set nodeText = nodeRoot.firstChild
    MsgBox nodeRoot.childNodes.length
    Set newNodeText = nodeText.splitText(6)
    MsgBox nodeRoot.childNodes.length
End If

[C/C++]

C/C++ Syntax

HRESULT splitText(
    long offset,
    IXMLDOMText **rightHandTextNode);
Parameters

offset [in]
The number of characters at which to split this text node into two nodes, starting from zero.

rightHandTextNode [out, retval]
The new text node.

C/C++ Return Values

S_OK
The value returned if successful.

S_FALSE
The value when returning Null.

E_FAIL
The value returned if an error occurs.

C/C++ Example

IXMLDOMText *pIXMLDOMTextTemp = NULL;
IXMLDOMText *pIXMLDOMText = NULL;
IXMLDOMNode *pIXMLDOMNodeTemp = NULL;
IXMLDOMElement *pIXMLDOMElement = NULL;
IXMLDOMNode *pIXMLDOMNode = NULL;
DOMNodeTyp DOMNodeType;
VARIANT varValue;
HRESULT hr;

try
{
    // Create an instance of DOMDocument and initialize pIXMLDOMDocument
    // Load/create an XML fragment.
    hr = pIXMLDOMDocument->get_documentElement(&pIXMLDOMElement);
    SUCCEEDED(hr) ? 0 : throw hr;

    if(pIXMLDOMElement)
    {
        hr = pIXMLDOMElement->QueryInterface(IID_IXMLDOMNode, (void **)&pIXMLDOMNode);
        SUCCEEDED(hr) ? 0 : throw hr;

        pIXMLDOMElement->Release();
        pIXMLDOMElement = NULL;
    }
}
while(pIXMLDOMNode)
{
hr = pIXMLDOMNode->get_nodeType(&DOMNodeTyp);
if(SUCCEEDED(hr) && (DOMNodeTyp == NODE_TEXT))
{
hr = pIXMLDOMNode->QueryInterface(IID_IXMLDOMText,
(void**)&pIXMLDOMText);
SUCCEEDED(hr) ? 0 : throw hr;

if(pIXMLDOMText)
{
hr = pIXMLDOMText->splitText(2, &pIXMLDOMTextTemp);
if(SUCCEEDED(hr) && pIXMLDOMTextTemp)
{
hr = pIXMLDOMTextTemp->get_nodeValue(&varValue);
if(SUCCEEDED(hr))
::MessageBox(NULL, _bstr_t(varValue), _T("Notation
pIXMLDOMTextTemp->Release();
pIXMLDOMTextTemp = NULL;
}
pIXMLDOMText->Release();
pIXMLDOMText = NULL;
}
pIXMLDOMNode->Release();
pIXMLDOMNode = NULL;
break;
}
pIXMLDOMNode->get_firstChild(&pIXMLDOMNodeTemp);
pIXMLDOMNode->Release();
pIXMLDOMNode->Release();
pIXMLDOMNode = pIXMLDOMNodeTemp;
pIXMLDOMNodeTemp = NULL;
}
}
catch(...)
{
if(pIXMLDOMNodeTemp)
pIXMLDOMNodeTemp->Release();
if(pIXMLDOMNode)
pIXMLDOMNode->Release();
if(pIXMLDOMTextTemp)
pIXMLDOMTextTemp->Release();
if(pIXMLDOMText)
pIXMLDOMText->Release();
if(pIXMLDOMElememt)
pIXMLDOMElememt->Release();
DisplayErrorToUser();
}
// Release pIXMLDOMDocument when finished with it.
Remarks

If you specify an offset of zero, the first text node is empty and the right-hand text node contains the previous contents of the node. If you specify an offset beyond the end of the string, a new, empty right-hand text node is created.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXMLDOMCDATASection | IXMLDOMText
**substringData Method**

Retrieves a substring of the full string from the specified range.

[Script]

**Script Syntax**

`strValue = oXMLDOMCharacterData.substringData(offset, count);`

**Parameters**

*offset*

A long integer value indicating the offset, in characters, from the beginning of the string. An offset of zero indicates copying from the start of the data.

*count*

A long integer value indicating the number of characters to retrieve from the specified offset.

**Return Value**

A string. Returns the substring.

**Example**

The following script example creates an IXMLDOMComment object (comment), and then uses the substringData method to retrieve the first five characters of the object.

```javascript
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var comment;
var MyStr;
xmlDoc.async = false;
xmlDoc.load("books.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
```
} else {
    comment = xmlDoc.createComment("Hello World!");
    MyStr = comment.substringData(0, 5);
    alert(MyStr);
}

[Visual Basic]

**Visual Basic Syntax**

```
strValue = oXMLDOMCharacterData.substringData(offset, count)
```

**Parameters**

- `offset`  
  A long integer value indicating the offset, in characters, from the beginning of the string. An offset of zero indicates copying from the start of the data.

- `count`  
  A long integer value indicating the number of characters to retrieve from the specified offset.

**Return Value**

A string. Returns the substring.

**Example**

The following Microsoft® Visual Basic® example creates an IXMLDOMComment object (comment), and then uses the substringData method to retrieve the first five characters of the object.

```
Dim xmlDoc As New Msxml2.DOMDocument50
Dim comment As IXMLDOMComment
Dim MyStr As String
xmlDoc.async = False
xmlDoc.Load("books.xml")
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    Set comment = xmlDoc.createComment("Hello World!")
```

MyStr = comment.substringData(0, 5)
MsgBox MyStr
End If

[C/C++]

C/C++ Syntax

HRESULT substringData(
    long offset,
    long count,
    BSTR *data);

Parameters

offset [in]
The offset, in characters, from the beginning of the string. An offset of zero indicates copying from the start of the data.

count [in]
The number of characters to retrieve from the specified offset.

data [out, retval]
The substring to return.

C/C++ Return Values

S_OK
    The value returned if successful.
S_FALSE
    The value when returning Null.

Remarks

If the offset and count parameters indicate a range beyond the end of the string, the returned substring continues only until the end of the string data.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
Applies to: IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | IXMLDOMText
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
transform Method

Starts the transformation process or resumes a previously failed transformation.

[Script]

Script Syntax

boolValue = objXSLProcessor.transform();

Example

var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
  var myErr = xmlDoc.parseError;
  alert("You have error " + myErr.reason);
} else {
  xslt.stylesheet = xslDoc;
  var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
  xmlDoc.async = false;
  xmlDoc.load("books.xml");
  if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
  } else {
    xslProc = xslt.createProcessor();
    xslProc.input = xmlDoc;
    xslProc.transform();
    alert(xslProc.output);
  }
}

File Name: Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver.../>
Hello
</xsl:template>
<xsl:template match="/" mode="edit">
  In Edit Mode
</xsl:template>
<xsl:template match="/" mode="view">
  In View Mode
</xsl:template>
</xsl:stylesheet>

[Visual Basic]

**Visual Basic Syntax**

boolValue = objXSLProcessor.transform

**Example**

```vbnet
Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
  Dim myErr
  Set myErr = xmlDoc.parseError
  MsgBox("You have error " & myErr.reason)
Else
  Set xslt.stylesheet = xslDoc
  xmlDoc.async = False
  xmlDoc.Load "books.xml"
  If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
  Else
    Set xslProc = xslt.createProcessor()
    xslProc.input = xmlDoc
    xslProc.Transform
    MsgBox xslProc.output
  End If
End If
```

**File Name: Sample2.xsl**
C/C++ Syntax

HRESULT transform (VARIANT_BOOL* pDone);

Parameters

pDone [out, retval]
The returned success message.

C/C++ Return Values

E_FAIL
The value returned if the value of the readyState property is not
READYSTATE_LOADED or READYSTATE_COMPLETE, or if a transformation error
is encountered, for example, a script engine error.
VARIANT_FALSE
The value returned if the end of the input tree is reached and more input is
pending.
VARIANT_TRUE
The value returned if the entire transformation is completed successfully.

Remarks

When the end of the transformation is reached successfully, VARIANT_TRUE is
returned.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IXSLProcessor
transformNode Method

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation.

[Script]

Script Syntax

strValue = oXMLDOMNode.transformNode(objStylesheet);

Parameters

objstylesheet
   An object. A valid XML document or DOM node that consists of XSLT elements that direct the transformation of this node.

Return Value

A string. Returns a string that contains the product of the transformation of this XML document based on the XSLT style sheet.

Example

The following Microsoft® JScript® example fills an HTML DIV tag with the result of transformNode.

<SCRIPT>
   // Load data.
   var source = new ActiveXObject("Msxml2.DOMDocument.5.0");
   source.async = false;
   source.load("books.xml");
   if (xmlDoc.parseError.errorCode <> 0) {
      var myErr = xmlDoc.parseError;
      alert("You have error " + myErr.reason);
   } else {
      // Load style sheet.
      var stylesheet = new ActiveXObject("Msxml2.DOMDocument.5.0");
   }
stylesheet.async = false
stylesheet.load("sample.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    // Fill a div tag with the result of the transform
    divInfo.innerHTML = source.transformNode(stylesheet);
}
</SCRIPT>

[Visual Basic]

**Visual Basic Syntax**

strValue = oXMLDOMNode.transformNode(objStylesheet)

**Parameters**

`objstylesheet`

An object. A valid XML document or DOM node that consists of XSLT elements that direct the transformation of this node.

**Return Value**

A string. Returns a string that contains the product of the transformation of this XML document based on the XSLT style sheet.

**Example**

The following Microsoft® Visual Basic® example displays the result of `transformNode`.

Dim source As New Msxml2.DOMDocument50
Dim stylesheet As New Msxml2.DOMDocument50

' Load data.
source.async = False
source.Load "books.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
Dim myErr
Set myErr = xmlDoc.parseError
MsgBox("You have error " & myErr.reason)
Else
' Load style sheet.
stYLESHEET.async = False
stylesheet.Load "sample.xsl"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    ' Do the transform.
    MsgBox source.transformNode(stylesheet)
End If
End If

[C/C++]

C/C++ Syntax

HRESULT transformNode(
    IXMLDOMNode *stylesheet,
    BSTR *xmlString);

Parameters

stylesheet [in]
    A valid XML document or DOM node that consists of XSLT elements that
direct the transformation of this node.
xmlString [out, retval]
    A string that contains the product of the transformation of this XML
document based on the XSLT style sheet.

C/C++ Return Values

S_OK
    The value returned if successful.
E_INVALIDARG
    The value returned if the stylesheet or xmlString parameter is Null.

Remarks
The stylesheet parameter must be either a DOMDocument node, in which case the document is assumed to be an Extensible Stylesheet Language (XSL) style sheet, or a Document Object Model (DOM) node in the XSL style sheet, in which case this node is treated as a standalone style sheet fragment.

The source node defines a context in which the style sheet operates, but navigation outside this scope is allowed. For example, a style sheet can use the id function to access other parts of the document.

This method supports both standalone and embedded style sheets and also provides the ability to run a localized style sheet fragment against a particular source node.

This member is an extension of the World Wide Web Consortium (W3C) DOM.

For more information about XSLT style sheets, see the XSLT Reference.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

Using XSLT with DOM or SAX | id Function

Applies to: IXMLDOMAttribute | IXMLDOMCDATASection | IXMLDOMCharacterData | IXMLDOMComment | DOMDocument | IXMLDOMDocumentFragment | IXMLDOMDocumentType | IXMLDOMElement | IXMLDOMEntity | IXMLDOMEntityReference | IXMLDOMNode | IXMLDOMNotation | IXMLDOMProcessingInstruction | IXMLDOMText
**transformNodeToObject Method**

Processes this node and its children using the supplied XSL Transformations (XSLT) style sheet and returns the resulting transformation in the supplied object.

[Script]

**Script Syntax**

`oXMLDOMNode.transformNodeToObject(stylesheet, outputObject);`

**Parameters**

*stylesheet*

An object. A valid XML document or DOM node that consists of XSLT elements that direct the transformation of this node.

*outputObject*

An object. On return, contains the product of the transformation of this XML document based on the XSLT style sheet. If the variant represents the DOMDocument object, the document is built according to its properties and its child nodes are replaced during this transformation process. The XML transformation can also be sent to a stream.

**Example**

The following Microsoft® JScript® example sets up the new XML document object named `result` before making the call to `transformNodeToObject`.

```<SCRIPT>
    // Load data.
    var source = new ActiveXObject("Msxml2.DOMDocument.5.0");
    source.async = false;
    source.load("data.xml");
    if (xmlDoc.parseError.errorCode <> 0) {
        var myErr = xmlDoc.parseError;
        alert("You have error " + myErr.reason);
    } else {
```
// Load style sheet.
var stylesheet = new ActiveXObject("Msxml2.DOMDocument.5.0");
stlyesheet.async = false;
stlyesheet.load("style.xsl");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    // Set up the resulting document.
    var result = new ActiveXObject("Msxml2.DOMDocument.5.0");
    result.async = false;
    result.validateOnParse = true;

    // Parse results into a result DOM Document.
    source.transformNodeToObject(stylesheet, result);
}
</SCRIPT>

[Visual Basic]

**Visual Basic Syntax**

```vbnet
oXMLDOMNode.transformNodeToObject(stylesheets, outputObject)
```

**Parameters**

- **stylesheet**
  - An object. A valid XML document or DOM node that consists of XSLT elements that direct the transformation of this node.

- **outputObject**
  - An object. On return, contains the product of the transformation of this XML document based on the XSLT style sheet. If the variant represents the **DOMDocument** object, the document is built according to its properties and its child nodes are replaced during this transformation process. The XML transformation can also be sent to a stream.

**Example**

The following Microsoft Visual Basic® example demonstrates the application of multiple style sheets to an XML file in succession.
Dim Source As New Msxml2.DOMDocument50
Dim stylesheet As New Msxml2.DOMDocument50
Dim stylesheet2 As New Msxml2.DOMDocument50
Dim result As New Msxml2.DOMDocument50
Dim result2 As New Msxml2.DOMDocument50

' Load data.
Source.async = False
Source.Load "sample.xml"
If (xmlDoc.parseError.errorCode <> 0) Then
    Dim myErr
    Set myErr = xmlDoc.parseError
    MsgBox("You have error " & myErr.reason)
Else
    ' Load style sheet.
    stylesheet.async = False
    stylesheet.Load "stylesheet1.xsl"
    If (xmlDoc.parseError.errorCode <> 0) Then
        Dim myErr
        Set myErr = xmlDoc.parseError
        MsgBox("You have error " & myErr.reason)
    Else
        ' Set up the resulting document.
        result.async = False
        result.validateOnParse = True
        result2.async = False
        result2.validateOnParse = True

        ' Parse results into a result DOM Document.
        Source.transformNodeToObject stylesheet, result
        stylesheet2.async = False
        stylesheet2.Load "stylesheet2.xsl"

        result.transformNodeToObject stylesheet2, result2
        MsgBox result2.xml
    End If
End If

File Name: Sample.xml

<?xml version="1.0"?>
<COLLECTION dateCreated="01-04-2000">
    <BOOK>
        <TITLE>Splish Splash</TITLE>
        <AUTHOR>Paula Thurman</AUTHOR>
        <PUBLISHER>Scootney</PUBLISHER>
        <PRICE>250</PRICE>
    </BOOK>
</COLLECTION>
<BOOK>
  <TITLE>Lover Birds</TITLE>
  <AUTHOR>Cynthia Randall</AUTHOR>
  <PUBLISHER>Lucerne Publishing</PUBLISHER>
  <PRICE>200</PRICE>
</BOOK>

<BOOK>
  <TITLE>The Sundered Grail</TITLE>
  <AUTHOR>Eva Corets</AUTHOR>
  <PUBLISHER>Lucerne Publishing</PUBLISHER>
  <PRICE>100</PRICE>
</BOOK>

</COLLECTION>

Stylesheet1.xsl

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  <xsl:template match="/">
    <PriceList>
      <xsl:for-each select="COLLECTION/BOOK">
        <xsl:sort select="TITLE" data-type="text"/>
        <xsl:copy>
          <xsl:apply-templates select="*"/>
        </xsl:copy>
      </xsl:for-each>
    </PriceList>
  </xsl:template>
</xsl:stylesheet>

Stylesheet2.xsl

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  <xsl:template match="/">
    <LowPriceBooks>
      <xsl:for-each select="*/BOOK[not(PRICE > '220')]">
        <xsl:copy>
          <xsl:apply-templates select="*"/>
        </xsl:copy>
      </xsl:for-each>
    </LowPriceBooks>
  </xsl:template>
</xsl:stylesheet>
Final Output

<?xml version="1.0"?>
<LowPriceBooks><BOOK>
<TITLE>The Sundered Grail</TITLE>
<AUTHOR>Eva Corets</AUTHOR><PUBLISHER>Lucerne Publishing</PUBLISHER>
<PRICE>100</PRICE></BOOK><BOOK><TITLE>Lover Birds</TITLE>
<AUTHOR>Cynthia Randall</AUTHOR>
<PUBLISHER>Lucerne Publishing</PUBLISHER><PRICE>200</PRICE>
</BOOK></LowPriceBooks>

[C/C++]

C/C++ Syntax

HRESULT transformNodeToObject(
    IXMLDOMNode *stylesheet,
    VARIANT outputObject);

Parameters

stylesheet [in]
    A valid XML document or DOM node that consists of XSL elements that
direct the transformation of this node.

outputObject [in]
    An object that contains the product of the transformation of this XML
document based on the XSLT style sheet. If the variant represents
DOMDocument, the document is built according to its properties and its child
nodes are replaced during this transformation process. If the variant
contains an IStream interface, the XML transformation is sent to this
stream.

C/C++ Return Values

S_OK
The value returned if successful.

E_INVALIDARG
The value returned if the stylesheet or outputObject parameter is Null.

Example

This example sets up the new DOMDocument object and puts it into a VARIANT before making the call to transformNodeToObject.

```cpp
// p is the XML source that is to be transformed, pXSL is the style
// Create an empty DOM document for the result. (Error checking omit
// for brevity.)
hr = CoCreateInstance(CLSID_DOMDocument50, NULL, CLSCTX_INPROC_SERVER,
    IID_IDispatch, (void**)&pDoc);
hr = pDoc->QueryInterface(IID_IDispatch, (void**)&pDisp);
vObject.vt = VT_DISPATCH; // the new object
vObject.pdispVal = pDisp;
hr = p->transformNodeToObject(pXSL, vObject); // Transformation is
    // present in pDoc.
```

Remarks

The stylesheet parameter must be either a DOMDocument node, in which case
the document is assumed to be an XSLT style sheet, or a DOM node in the XSLT
style sheet, in which case this node is treated as a standalone style sheet
fragment.

The source node defines a context in which the style sheet operates, but
navigation outside this scope is allowed. For example, a style sheet can use the
id function to access other parts of the document.

This method supports both standalone and embedded style sheets and also
provides the ability to run a localized style sheet fragment against a particular
source node.

The transformNodeToObject method always generates a Unicode byte-order
mark, which means it cannot be used in conjunction with other Active Server
Pages (ASP) Response.Write or Response.BinaryWrite calls.

For more information about XSLT, see the XSLT Reference.
This member is an extension of the World Wide Web Consortium (W3C) DOM.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[Using XSLT with DOM or SAX](#) | [id Function](#)

Applies to: [IXMLDOMAttribute](#) | [IXMLDOMCDATASection](#) | [DOMDocument](#)
[IXMLDOMCharacterData](#) | [IXMLDOMComment](#) | [IXMLDOMDocumentFragment](#) | [IXMLDOMDocumentType](#) | [IXMLDOMElemen](#) | [IXMLDOMEntity](#) | [IXMLDOMEntityReference](#) | [IXMLDOMNode](#) | [IXMLDOMNotation](#) | [IXMLDOMProcessingInstruction](#) | [IXMLDOMText](#)
validate Method

Performs run-time validation on the currently loaded document using the currently loaded document type definition (DTD), schema, or schema collection.

[Script]

Script Syntax

objXMLDOMDocument2.validate();

Return Value

A ParseError object indicating exactly what error occurred, if any.

Example

The following Microsoft® JScript® example shows how to validate at runtime.

```javascript
var xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmldoc.async = false
// This will validate on load because validateOnParse is set to true // by default.
xmldoc.load("http://server/myData.xml");
// You make a change to an attribute:
xmldoc.documentElement.setAttribute("a", "123");
// Now you want to verify your document is still valid:
var err = xmldoc.validate();
if (err.errorCode == 0)
{
    alert("Document is valid");
}
else
{
    alert("Validation error:" + err.reason);
}
```

[Visual Basic]

Visual Basic Syntax
objXMLDOMDocument2.validate

**Return Value**

A ParseError object indicating exactly what error occurred, if any.

[C/C++]

**C/C++ Syntax**

HRESULT validate(IXMLDOMParseError **errorObj);

**C/C++ Return Values**

An IErrorInfo containing a formatted error message indicating what went wrong, and one of the following HRESULTs (also returned in the ErrorInfo.Number property).

<table>
<thead>
<tr>
<th>Return value</th>
<th>Hexadecimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_PENDING</td>
<td>0x8000000A</td>
<td>Document readyState is not 4, indicating the document is not completely loaded.</td>
</tr>
<tr>
<td>S_OK</td>
<td>0</td>
<td>The document is valid according to DTD or schemas.</td>
</tr>
<tr>
<td>S_FALSE</td>
<td>1</td>
<td>The document is invalid according to DTD or schemas. For error information, see the returned IXMLDOMParseError object.</td>
</tr>
</tbody>
</table>

The following possible validation errors returned with S_FALSE are listed, together with the errorCode value.

<table>
<thead>
<tr>
<th>Message ID</th>
<th>Error Code (Hex Value)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML_E_NODTD</td>
<td>0xC00CE224</td>
<td>The validate method failed because a DTD or schema was</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>XML_E_NOTWF 0x00CE223</td>
<td>The validate method failed because the document does not contain exactly one root node.</td>
<td></td>
</tr>
<tr>
<td>XML_ENTITY_UNDEF 0x00CE002</td>
<td>Reference to undefined entity '%1'. Entity '%1' contains an infinite entity reference loop.</td>
<td></td>
</tr>
<tr>
<td>XML_INFINITE_ENTITY_LOOP 0x00CE003</td>
<td>Cannot use the NDATA keyword in a parameter entity declaration. Cannot use a general parsed entity '%1' as the value for attribute '%2'. Cannot use unparsed entity '%1' in an entity reference. Cannot reference an external general parsed entity '%1' in an attribute value.</td>
<td></td>
</tr>
</tbody>
</table>
The element '%1' is used but not declared in the DTD or schema.
The attribute '%1' references the ID '%2', which is not defined in the document.
Element cannot be empty according to the DTD or schema.
Element content is incomplete according to the DTD or schema.
The name of the top-most element must match the name of the DOCTYPE declaration.
Element content is invalid according to the DTD or schema.
The attribute '%1' on this
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML_ATTRIBUTE_NOT_DEFINED 0xC00CE015</td>
<td>element is not defined in the DTD or schema. Attribute '%1' has a value that does not match the fixed value defined in the DTD or schema.</td>
</tr>
<tr>
<td>XML_ATTRIBUTE_FIXED 0xC00CE016</td>
<td>Attribute '%1' has a value that does not match the fixed value defined in the DTD or schema.</td>
</tr>
<tr>
<td>XML_ATTRIBUTE_VALUE 0xC00CE017</td>
<td>Attribute '%1' has an invalid value according to the DTD or schema. Text is not allowed in this element according to the DTD or schema.</td>
</tr>
<tr>
<td>XML_ILLEGAL_TEXT 0xC00CE018</td>
<td>An attribute declaration cannot contain multiple fixed values: '%1'.</td>
</tr>
<tr>
<td>XML_MULTI_FIXED_VALUES 0xC00CE019</td>
<td>Reference to undeclared element: '%1'. Attribute '%1' must be a #FIXED attribute.</td>
</tr>
<tr>
<td>XML_ELEMENT_UNDEFINED 0xC00CE01C</td>
<td>Required attribute '%1' is</td>
</tr>
<tr>
<td>XML_XMLNS_FIXED 0xC00CE01E</td>
<td></td>
</tr>
<tr>
<td>XML_REQUIRED_ATTRIBUTEMISSING 0xC00CE020</td>
<td></td>
</tr>
</tbody>
</table>
Remarks

This method only validates fully loaded documents (readyState == 4).

The validate method returns an IXMLDOMParseError that is independent of the value returned by the parseError property on a document. Only the errorCode and reason properties of the returned value are set.

Unlike the load method, validate will fail if there is no DTD or schema applied to the document element. Therefore, validate will not be able tell you whether the document is just well-formed.

The validate method does not parse new schemas, but can import a schema from a SchemaCache associated with the document through the schemas property. If there is no schema for a given namespace, the elements in that namespace will not be validated.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

readyState Property | IXMLDOMParseError

Applies to: IXMLDOMDocument2
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
validate Method
(IXMLDOMSchemaCollection2/XML)

Performs run-time validation on the documents in the schema cache that have not been compiled and validated. If a schema has its validateOnLoad property set to true when it is loaded into the cache, the schema will be compiled and validated at that time. The validate method is used when validateOnLoad is set to false. Steps 2 and 3 are postponed until the validate method is called.

The Schema compilation steps are:

1. Check Syntax and resolve names.
2. Load and preprocess external schemas.
3. Compile schema items respecting W3Cs rules.

[Script]

**Script Syntax**

```javascript
objXMLSchemaCache.validate();
```

**Return Value**

None.

[Visual Basic]

**Visual Basic Syntax**

```vbnet
objXMLSchemaCache.validate
```

**Return Value**

None.
[C/C++]

**C/C++ Syntax**

HRESULT validate();

**C/C++ Return Values**

None.

**See Also**

[validateOnLoad Property](#)

Applies to: [IXMLDOMSchemaCollection2/XMLDOMSchemaCollection](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
validateNode Method

 Validates a specified DOM fragment. A DOM fragment is a sub-tree spanning from a given node.

 [JScript]

 Script Syntax

 var objErr = objXMLDOMDocument3.validateNode(node);

 [Visual Basic]

 Visual Basic Syntax

 Set objErr = objXMLDOMDocument3.validateNode(node);

 [C/C++]

 C/C++ Syntax Using Smart Pointers

 IXMLDOMParseErrorPtr objErr = objXMLDOMDocument3->validateNode(node)

 C/C++ Syntax

 HRESULT IXMLDOMDocument3::validateNode(
    ...[in] IXMLDOMNode* node,
    ...[out, retval] IXMLDOMParseError** objErr);

 Parameters

 node
 An IXMLDOMNode object representing the node from which the fragment or sub-tree expands.

 objErr
 A reference to the IXMLDOMParseError object that indicates exactly what error occurred, if any. Use the properties on this object to examine specific
parse error information.

**Return Values**

**S_OK**  
Validation was successful.

**E_FAIL**  
The node failed to validate for one of the following reasons:
- The DOM document was not completely loaded
- The DOM node does not belong to this DOM document instance
- The DOM node type was not element or attribute.

**S_FALSE**  
Validation errors were encountered. Check the returned `objErr` object for detailed parse error information.

**Note**  
This method returns the same HRESULT value that is returned by the `IXMLDOMDocument2::import` method.

**Remarks**

In the syntax for Visual Basic and JScript above, an `objXMLDOMDocument3` object is a DOMDocument instance that implements the `IXMLDOMDocument3` interface.

Unlike `IXMLDOMDocument2::validate`, `validateNode` will not cause a new schema to load. Furthermore, `validateNode` assumes that the rest of the document is valid. Without this assumption it would not be possible to validate ID/IDREFs at all with the `validateNode` method.

The `validateNode` method is applicable only to element or attribute nodes located within the same DOM document. Any attempt to call it on other types of nodes or nodes in other DOM documents results in an error.

**Example**

The `validateNode` example shows how to validate a DOM fragment at run time. The example uses two resource files, `validateNode.xml` and `validateNode.xsd`. The first is an XML data file, and the second is the XML Schema for the XML data. The first `<book>` element in `validateNode.xml` is valid against
validateNode.xsd; the second <book> element is not.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource Files (validateNode.xml and validateNode.xsd)
- JScript Code (validateNode.js)
- Visual Basic Code (validateNode.frm)
- C/C++ Code (validateNode.cpp)
- Output for validateNode Example

Applies To

IXMLDOMDocument3

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDOMNode | IXMLDOMNodeList | IXMLDOMParseError | IXMLDOMSchemaCollection/XMLSchemaCache
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer...
Resource Files (*validateNode.xml* and *validateNode.xsd*)

Use the following resource files for the `validateNode` example.

**Input XML Data File (validateNode.xml)**

```xml
<?xml version="1.0"?>
<x:books xmlns:x="urn:books">
  <book id="bk001">
    <author>Hightower, Kim</author>
    <title>The First Book</title>
    <genre>Fiction</genre>
    <price>44.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>An amazing story of nothing.</review>
  </book>
  <book id="bk003">
    <author>Nagata, Suanne</author>
    <title>Becoming Somebody</title>
    <genre>Biography</genre>
    <review>A masterpiece of the fine art of gossiping.</review>
  </book>
</x:books>
```

**XML Schema Definition File (validateNode.xsd)**

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:books"
  xmlns:bks="urn:books">
  <xsd:element name="books" type="bks:BooksForm"/>
  <xsd:complexType name="BooksForm">
    <xsd:sequence>
      <xsd:element name="book" type="bks:BookForm"
        minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```
<xsd:complexType name="BookForm">
  <xsd:sequence>
    <xsd:element name="author" type="xsd:string"/>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="genre" type="xsd:string"/>
    <xsd:element name="price" type="xsd:float"/>
    <xsd:element name="pub_date" type="xsd:date"/>
    <xsd:element name="review" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
JScript Code (validateNode.js)

```javascript
var xs, xd;

main();

function main() {
  try {
    xs = new ActiveXObject("MSXML2.XMLSchemaCache.5.0");
    xd = new ActiveXObject("MSXML2.DOMDocument.5.0");
  }
  catch (e) {
    alert("Mirossoft XML Core Services (MSXML) 5.0 for Microsoft Office is not installed.
    Download and install MSXML 5.0 from http://msdn.microsoft.com
    before continuing.");
    return;
  }

  try {
    xs.add("urn:books", "validateNode.xsd");
    xd.schemas = xs;
    xd.async = false;
    xd.validateOnParse = false;
  }
  catch (e) {
    alert("Failed to add to schema cache: " + e.description);
    return;
  }

  try {
    xd.load("validateNode.xml");
  }
  catch (e) {
    alert("can't load validateNode.xml : " + e.description);
    return;
  }

  var err = xd.validate();
  if (err.errorCode != 0) {
    alert("invalid dom: \n" + err.reason);
  } else {
    alert("dom is valid: \n" + xd.xml);
  }

  var nlist = xd.selectNodes("//book");
  for (var node = nlist.nextNode(); node != null; node = nlist.nextNode(
```
```javascript
{  
  var err = xd.validateNode(node);

  if (err.errorCode != 0)
    alert("invalid node:\n" + err.reason);
  else
    alert(node.xml);
}
}

function alert(msg)
{
  WScript.echo(msg);
}

Try It!

1. Copy the XML data (validateNode.xml), and paste it into a text file. Save the file as validateNode.xml.
2. Copy the XSD listing (validateNode.xsd), and paste it into a text file. Save the file as validateNode.xsd, in the same directory where you saved validateNode.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as validateNode.js, in the same directory where you saved validateNode.xml and validateNode.xsd.
4. Double click the validateNode.js file from Windows Explorer to launch the application. Alternatively, you can type "validateNode.js" from a command prompt.

  Note  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in Output for the validateNode Example.
Visual Basic Code (validateNode.frm)

```vbnet
Dim xd As New MSXML2.DOMDocument50
Dim xs As New MSXML2/XMLSchemaCache50
Dim er As IXMLDOMParseError
Dim nlist As IXMLDOMNodeList
Dim node As IXMLDOMNode

xs.Add "urn:books", App.Path & "\validateNode.xsd"
Set xd.schemas = xs
xd.validateOnParse = False
xd.async = False
xd.Load App.Path & "\validateNode.xml"

Set er = xd.validate
If er.errorCode <> 0 Then
    MsgBox "invalid dom: " & vbCrLf & er.reason
Else
    MsgBox "dom is valid: " & vbCrLf & er.reason & vbCrLf & xd.xml
End If

Set nlist = xd.selectNodes('//book')
i = 0
For Each node In nlist
    Set er = xd.validateNode(node)
    If er.errorCode <> 0 Then
        MsgBox "invalid node: " & vbCrLf & er.reason
    Else
        MsgBox node.xml
    End If
Next
```

Try It!

1. Copy the XML data ([validateNode.xml](#)), and paste it into a text file. Save the file as `validateNode.xml`.
2. Copy the XSD listing ([validateNode.xsd](#)), and paste it into a text file. Save the file as `validateNode.xsd`, in the same directory where you saved `validateNode.xml`.
3. Create a Standard EXE project in Visual Basic. Save the empty project as `validateNode.vbp` to the same directory where you saved `validateNode.xml` and `validateNode.xsd`. Name the form file `validateNode.frm`. 
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v50.
5. Copy the Visual Basic code listing above, and paste it into the form_load subroutine.
6. Execute the code by selecting Start from the Run menu.
7. Verify that the output is the same as that listed in Output for the validateNode Example.
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

void dump_com_error(_com_error &e);
void show_parse_error(_bstr_t ctx, IXMLDOMParseErrorPtr pErr);

int main(int argc, char* argv[])
{
    IXMLDOMDocument3Ptr pXMLDoc = NULL;
    IXMLDOMDocument3Ptr pXSDDoc = NULL;
    IXMLDOMSchemaCollectionPtr pSCache = NULL;
    IXMLDOMNodePtr pNode = NULL;
    IXMLDOMNodeListPtr pNodelist = NULL;
    IXMLDOM ParseErrorPtr pError = NULL;
    HRESULT hr;
    long i = 0;
    long x = 0;

    CoInitialize(NULL);
    try{
        // Load validateNode.xml into a DOM instance.
        hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
        if (FAILED(hr)) {
            MessageBox(NULL, "Cannot create DOMDocument50 xml",
                        "CreateInstance", MB_OK);
            return 0;
        }
        pXMLDoc->async = VARIANT_FALSE;
        pXMLDoc->validateOnParse = VARIANT_FALSE;
        if(pXMLDoc->load("validateNode.xml") != VARIANT_TRUE)
        {
            show_parse_error(_bstr_t("Can't load validateNode.xml\n"),
                              pXMLDoc->parseError);
            return 0;
        }

        // Load validateNode.xsd into a DOM instance.
        hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
        if (FAILED(hr)) {
            MessageBox(NULL, "Cannot create DOMDocument50 for xsd",
                        "CreateInstance", MB_OK);
            return 0;
        }
    }
    catch(...)
    {
        return 0;
    }
}
pXSDDoc->async = VARIANT_FALSE;
pXSDDoc->validateOnParse = VARIANT_FALSE;
if(pXSDDoc->load("validateNode.xsd") != VARIANT_TRUE)
{
    show_parse_error(_bstr_t("Can't load validateNode.xsd\n"),
                   pXSDDoc->parseError);
    return 0;
}

// Create a schema cache.
hr = pSCache.CreateInstance(__uuidof(XMLSchemaCache50));
if (FAILED(hr)) {
    MessageBox(NULL, "Cannot create XMLSchemaCache50",
               "CreateInstance", MB_OK);
    return 0;
}

// Point the XML doc's schema to the loaded schema cache.
pXMLDoc->schemas = pSCache.GetInterfacePtr();

hr = pSCache->add("urn:books", pXSDDoc.GetInterfacePtr());
if (FAILED(hr)) {
    MessageBox(NULL, "Cannot add schema", "add", MB_OK);
    return 0;
}

// Validate the entire DOM object.
pError = pXMLDoc->validate();
if (pError->errorCode != 0) {
    show_parse_error(_bstr_t("invalid dom:\n"), pError);
} else
    MessageBox(NULL, pXMLDoc->xml, "import", MB_OK);

// Validate all book nodes, node by node.
pNodelist = pXMLDoc->selectNodes("//book");
x = pNodelist->length;
for(i = 1; i <= x; i++)
{
    pNode = pNodelist->nextNode();
    pError = pXMLDoc->validateNode(pNode);
    if (pError->errorCode != 0)
    {
        show_parse_error(_bstr_t("invalid node:\n"), pError);
    } else
    MessageBox(NULL, pNode->xml, "validateNode", MB_OK);
}
catch(_com_error &e)
void dump_com_error(_com_error &e)
{
    printf("Error
");
    printf("%a\n", e.Error());
    printf("%a\n", e.ErrorMessage());
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());
    printf("%a\n", (LPCSTR) bstrSource);
    printf("%a\n", (LPCSTR) bstrDescription);
}

void show_parse_error(_bstr_t ctx, IXMLDOMParseErrorPtr pError)
{
    _bstr_t parseError = ctx + _bstr_t(pError->Getreason());
    MessageBox(NULL, parseError, "Parse Error", MB_OK);
}

Try It!

2. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "validateNodeProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
3. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
4. Select FileView on the project browser, and highlight validateNodeProj files. From the File menu, select New.
5. On the Files tab of the New dialog box, highlight C++ Source File. Then type "validateNode.xml" in the File name text box.
6. Click OK.
7. Copy the XML data file (*validateNode.xml*), and paste it into the text file you just created.
8. Repeat steps 4-7 for the XSD listing (*validateNode.xsd*).

   **Note** You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 4-7 for the C++ file above (*validateNode.cpp*).
10. Build the sample by selecting **Build validateNodeProj.exe** from the **Build** menu.
11. Execute the sample application by selecting **!Execute validateNodeProj.exe** from the **Build** menu.
12. Verify that the result is the same as that listed in **Output for the validateNode Example**.
Output for the validateNode Example

When you build and run the validateNode code, you should get the following output, displayed in three message boxes.

- The first message box contains the following text:

  invalid dom: 
  Content for element 'review' is invalid according to the DTD/Schema. 
  Expecting: price.

  This message is the result of using the import() method to import the entire DOM object.

- The second message is the output from the validateNode method when it is applied to the first <book> node, which is valid against the schema:

  <book id="bk001"> 
    <author>Hightower, Kim</author> 
    <title>The First Book</title> 
    <genre>Fiction</genre> 
    <price>44.95</price> 
    <pub_date>2000-10-01</pub_date> 
    <review>An amazing story of nothing</review> 
  </book>

- The final message is the output from the validateNode method when it is applied to the second <book> node, which is invalid against the schema:

  invalid node: 
  Content for element 'review' is invalid according to the DTD/Schema. 
  Expecting: price.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
waitForResponse Method

Allows the requesting server to suspend execution while waiting for an asynchronous send operation to complete.

[Script]

Script Syntax

oServerXMLHTTPRequest.waitForResponse(timeoutInSeconds);

Parameters

timeoutInSeconds (optional)

Specifies the number of seconds to wait for an asynchronous send operation to complete.

Example

```javascript
var xmlServerHttp = new ActiveXObject("Msxml2.ServerXMLHTTP.5.0");
xmlServerHttp.open("GET", "http://localhost/sample.xml", true);
xmlServerHttp.send();
while (xmlServerHttp.readyState !== 4) {
    xmlServerHttp.waitForResponse(1000);
}
```

[Visual Basic]

Visual Basic Syntax

```vbnet
oServerXMLHTTPRequest.waitForResponse(timeoutInSeconds)
```

Parameters

timeoutInSeconds (optional)

Specifies the number of seconds to wait for an asynchronous send operation to complete.
**Example**

```vbnet
Dim xmlServerHttp As New Msxml2.ServerXMLHTTP50
xmlServerHttp.open "GET", "http://localhost/sample.xml", True
xmlServerHttp.send
While xmlServerHttp.readyState <> 4
    xmlServerHttp.waitForResponse 1000
Wend
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT waitForResponse(VARIANT timeoutInSeconds, VARIANT_BOOL * isSuccessful);
```

**Parameters**

- `timeoutInSeconds` [in, optional]
  Specifies the number of seconds to wait for an asynchronous send operation to complete.
- `isSuccessful` [out, retval]
  The `waitForResponse` method is more efficient than polling the `readyState` property, which is the only way to wait for an asynchronous send using the `XMLHTTP` component. The caller can supply an optional `timeout` parameter, specified in seconds. The default timeout (if one is not specified) is INFINITE (-1). The method returns True if a response is received within the time allotted, or False if a timeout occurs. If the method times out, the request is not aborted; the caller can continue to wait for the request in a subsequent call to the `waitForResponse` method. Calling `waitForResponse` after a synchronous send method returns immediately and has no useful effect. The `async` parameter to the `open` method controls whether the send is synchronous (default) or asynchronous. The results of this method are valid only after the send method has been completed successfully.

**C/C++ Return Values**

- `S_OK`
  The value returned if successful.
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

open Method | readyState Property | send Method

Applies to: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
# XML DOM Events

This section describes the following DOM events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ondataavailable</td>
<td>Fires when new data is available.</td>
</tr>
<tr>
<td>onreadystatechange</td>
<td>Fires when the <code>readyState</code> method changes.</td>
</tr>
<tr>
<td>ontransformnode Event</td>
<td>Fires before the style sheet is applied to a node.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
ondataavailable Event

Fires when new data is available.

**Inline HTML**

```html
<ELEMENT ondataavailable = "handler" ... >
```

All platforms

Microsoft® JScript®
(compatible with ECMA 262 language specification) only

**Event property**

```
object.ondataavailable = handler
```

Microsoft Internet Explorer only

**Named script**

```javascript
<SCRIPT FOR = object EVENT = ondataavailable>
```

[Script]

**Script Syntax**

```
oXMLDOMDocument.ondataavailable;
```

[Visual Basic]

**Visual Basic Syntax**

```
oXMLDOMDocument.ondataavailable
```

[C/C++]

**C/C++ Syntax**

```
HRESULT ondataavailable(void);
```

**C/C++ Return Values**
S_OK
    The value returned if successful.

Example

    // Assumptions :
    // The sink (an object of the class defined below) has been connecte
    // to the connection point (DIID_XMLOMDocumentEvents).
    // To see an example of how to connect the connection point,
    // see the example for the IXMLODDocument::onreadystatechange Event

BEGIN_DISPATCH_MAP(CXMLDOMSamples, CCmdTarget)
    DISP_FUNCTION_ID(CXMLDOMSamples, "Data Available Event Handler",
        DISPID_XMLOMEVENT_ONDATAAVAILABLE, DOMDocOnDataAvailable, VT_EMPTY)
END_DISPATCH_MAP()

class CXMLDOMSamples : public CCmdTarget
{
public:
    void DOMDocOnDataAvailable();
};
void CXMLDOMSamples::DOMDocOnDataAvailable()
{
    // Called whenever any data is available.
    try
    {
        ::MessageBox(NULL, _T("in ondataavailable()"), _T("Error"), MB
    }
    catch(...)
    {
        DisplayErrorToUser();
    }

Remarks

When handling asynchronous data, it is useful to be able to start processing in
parallel with the download as soon as data becomes available. The readyState
property defines several states that specify the current status of the asynchronous
download.

The event handler for this event can be explicitly set using the ondataavailable
property.
This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

ondataavailable Property | readyState Property (DOMDocument)

Applies to: DOMDocument
onreadystatechange Event

Fires when the readyState method changes.

**Inline HTML**

```html
<ELEMENT onreadystatechange = "handler" ... >
```

**Event property**

```javascript
object.onreadystatechange = handler
```

**Named script**

```javascript
<SCRIPT FOR = object EVENT = onreadystatechange>
```

[Script]

**Script Syntax**

```javascript
oXMLDOMDocument.onreadystatechange;
```

**Example**

The following JScript and HTML example demonstrates the sequence of ready states when asynchronously loading a document.

```javascript
<script>
var xmldoc;
function Load()
{
  xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
  xmldoc.onreadystatechange = CheckState;
  xmldoc.load(URL.value);
}
```
function CheckState()
{
    var state = xmldoc.readyState;
    RESULTS.innerHTML += "readyState = " + state + "<BR>"
    if (state == 4)
    {
        var err = xmldoc.parseError;
        if (err.errorCode != 0)
        {
            RESULTS.innerHTML += err.reason + "<BR>"
            else RESULTS.innerHTML += "success" + "<BR>"
        }
    }
}</script>
URL: <input type=text size=60 id=URL>
<input type=button value=LOAD onclick="jscript:Load()">
<div id=RESULTS style= "color:red;font-weight:bold;"></div>

[Visual Basic]

**Visual Basic Syntax**

`oXMLDOMDocument.onreadystatechange`

**Example**

The following example shows how to use the onreadystatechange event in an application.

Dim WithEvents xmldoc As DOMDocument50

Private Sub Form_Load()
    Set xmldoc = New DOMDocument50
    xmldoc.Load("http://MyServer/Sample.xml")
End Sub

Private Sub xmldoc_onreadystatechange()
    If (xmldoc.readyState = 4) Then
        msg = "Load is finished" & Chr(13)
        If (xmldoc.parseError.errorCode <> 0) Then
            msg = msg + "Error: " & xmldoc.parseError.reason
        End If
    MsgBox msg
End If

End Sub
**C/C++ Syntax**

HRESULT onreadystatechange(void);

**C/C++ Return Values**

S_OK
The value returned if successful.

**Example**

BEGIN_DISPATCH_MAP(CXMLDOMSamples, CCmdTarget)
    DISP_FUNCTION_ID(CXMLDOMSamples, "Ready State Event Handler",
        DISPID_XMLOrdomEVENT_ONREADYSTATECHANGE, DOMDocCheckState, VT_EMPTY
    )
END_DISPATCH_MAP()

class CXMLDOMSamples : public CCmdTarget
{
private:
    HRESULT m_hr;
    IXMLDOMDocument* m_pIXMLDOMDocument;
    BSTR m_bstrXMLDocName;
    DWORD m_pdwCookie;
public:
    BOOL AdviseConnectionPoint(BOOL bAdvise = TRUE);
    void Caller();
    BOOL DOMDocLoadLocation();
    BOOL DOMDocStateChangeSetup();
    void DOMDocCheckState();
};

// Assumptions
// m_pIXMLDOMDocument has already been initialized (a DOMDocument
// object has been created).
// Automation is enabled for the class to enable it to connect
// to the Connection point (DIID_XMLOrdomDocumentEvents).

void CXMLDOMSamples::Caller()
{
    try
    {
        m_bstrXMLDocName = ::SysAllocString(_T("http://MyServer/sample.xml"));
        DOMDocStateChangeSetup();
    }
DOMDocLoadLocation();
::SysFreeString(m_bstrXMLDocName);
m_bstrXMLDocName = NULL;
}
catch(...)
{
    if(m_bstrXMLDocName)
        ::SysFreeString(m_bstrXMLDocName);
    DisplayErrorToUser();
}

BOOL CXMDOMSamples::AdviseConnectionPoint(BOOL bAdvise)
{
    BOOL bResult = FALSE;
    try
    {
        IConnectionPointContainer *pIConnectionPointContainer = NULL;
        IConnectionPoint *pCP = NULL;

        m_hr = m_pIXMLDOMDocument->QueryInterface
            (IID_IConnectionPointContainer,(void**)&pIConnectionPointContainer);
        SUCCEEDED(m_hr) ? 0 : throw m_hr;

        if(pIConnectionPointContainer)
        {
            m_hr = pIConnectionPointContainer->FindConnectionPoint
                (DIID_XMLODOMDocumentEvents, &pCP);
            SUCCEEDED(m_hr) ? 0 : throw m_hr;
            if(SUCCEEDED(m_hr) && pCP)
            {
                if(bAdvise)
                    m_hr = pCP->Advise(GetIDispatch(TRUE), &m_pdwCookie);
                else
                    m_hr = pCP->Unadvise(m_pdwCookie);
                bResult = SUCCEEDED(m_hr) ? TRUE : FALSE;
                pCP->Release();
                pCP = NULL;
            }
            pIConnectionPointContainer->Release();
            pIConnectionPointContainer = NULL;
        }
    }
    catch(...)
    {
        if(pIConnectionPointContainer)
            pIConnectionPointContainer->Release();
        if(pCP)
            pCP->Release();
        DisplayErrorToUser();
    }
return bResult;
}

BOOL CXMLDOMSamples::DOMDocStateChangeSetup()
{
    BOOL bResult = FALSE;
    try
    {
        m_hr = m_pIXMLDOMDocument->put_async(VARIANT_TRUE);
        SUCCEEDED(m_hr) ? 0 : throw m_hr;
        bResult = AdviseConnectionPoint();
    }
    catch(...)
    {
        DisplayErrorToUser();
    }
    return bResult;
}

BOOL CXMLDOMSamples::DOMDocLoadLocation()
{
    short sResult = FALSE;
    BOOL bResult = FALSE;
    try
    {
        _variant_t varString = m_bstrXMLDocName;
        m_hr = m_pIXMLDOMDocument->load(varString, &sResult);
        bResult = SUCCEEDED(m_hr) ? TRUE : FALSE;
    }
    catch(...)
    {
        DisplayErrorToUser();
    }
    return bResult;
}

void CXMLDOMSamples::DOMDocCheckState()
{
    IXMLDOMParseError *pIParseError = NULL;
    long value;
    try
    {
        m_hr = m_pIXMLDOMDocument->get_readyState(&value);
        SUCCEEDED(m_hr) ? 0 : throw m_hr;
        if(value == 4 )
        {
            m_hr = m_pIXMLDOMDocument->get_parseError(&pIParseError);
            SUCCEEDED(m_hr) ? 0 : throw m_hr;
            m_hr = pIParseError->get_errorCode(&value);
            pIParseError->Release();
            pIParseError=NULL;
        }
    }
    catch(...)
    {
        DisplayErrorToUser();
    }
}
if(!SUCCEEDED(m_hr))
{
    throw m_hr;
}

m_strOutput = (value != 0) ? _T("Load Error") : _T("Load Succeeded");

} catch(...) {
    if(pIParseError)
        pIParseError->Release();
        DisplayErrorToUser();
}

**Remarks**

This event can be explicitly set using the `onreadystatechange` property.

This member is an extension of the Worldwide Web Consortium (W3C) Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[onreadystatechange Property (DOMDocument)]

Applies to: [DOMDocument]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - DOM Refer
ontransformnode Event

Fires before the style sheet is applied to a node.

**Important** The ontransformnode event does not fire on DOMDocument for style sheets using the XSLT namespace "http://www.w3.org/1999/XSL/Transform". It does, however, fire on DOMDocument for style sheets using the XSL namespace "http://www.w3.org/XSL/Transform/1.0" or where MSXML 3.0 is installed you use "http://www.w3.org/TR/WD-xsl" to declare the older XSL namespace.

[Script]

**Script Syntax**

`boolVal = ontransformnode(nodeCode, nodeData);`

**Parameters**

`nodeCode` The indicator of the current node in the style sheet.

`nodeData` The indicator of the current node in the XML source data.

**Return Value**

True to indicate that transformation processing is to continue; False to indicate that transformation processing should stop.

[Visual Basic]

**Visual Basic Syntax**

`boolVal = ontransformnode(nodeCode, nodeData)`
Parameters

nodeCode
The indicator of the current node in the style sheet.

nodeData
The indicator of the current node in the XML source data.

Return Value

True to indicate that transformation processing is to continue; False to indicate that transformation processing should stop.

[Script]

Example

In this example, the XML data island named "target" contains the XML source data, and the XML data island named "ss" contains the XSL Transformations (XSLT) style sheet.

<XML id="target">
   <PEOPLE>
      <PERSON name="Atkinson, Teresa" />
      <PERSON name="Debroux, Steve" />
      <PERSON name="O'Hara, Robert" />
      <PERSON name="Porzuczek, Peter" />
   </PEOPLE>
</XML>

<XML id="ss">
   <xsl:template match="/" xmlns:xsl="http://www.w3.org/XSL/Transform"
      <xsl:for-each select="/PEOPLE/PERSON/@name" >
         <P><xsl:value-of /></P>
      </xsl:for-each>
   </xsl:template>
</XML>

Example

This script contains the Verify function, which sets the ontransformnode property to the event handler, onxform. The event handler is called before the transformation operation on each node so that, given the sample data provided in
the data islands, this sample calls the alert several times.

```html
<SCRIPT LANGUAGE="javascript" FOR="window" EVENT="onload">
document.expando = false;
Verify();
</SCRIPT>
<Script LANGUAGE="javascript">
function onxform (code, doc ) {
    alert("Target document node being transformed:\n\n" + doc.xml + "\n\nStylesheet node used for the transform:\n\n" + code.xml);
    if (doc.value == "O'Hara, Robert")
        return false; // Stop processing.
    else
        return true;
}
function Verify() {
    target.ontransformnode = onxform;
    var sOut = target.XMLDocument.transformNode(ss.XMLDocument);
    div.innerHTML += sOut;
}
</Script>
</DIV>
</BODY>
</HTML>

The return value from the event handler indicates whether to continue processing. In this example, the event handler checks for a particular data value and returns the value "false" to stop transformation processing.

[C/C++]

C/C++Syntax

```c
HRESULT ontransformnode(  
    IXMLDOMNode *pCode,  
    IXMLDOMNode *pData,  
    VARIANT vBool);
```  

**Parameters**

*pCode* [in]  
The address of the current node in the XSL style sheet.

*pData* [in]  
The address of the current node in the XML source data.
vBool [out, retval]
   Boolean. True if transformation processing is to continue; False to stop
   transformation processing.

C/C++ Return Values

S_OK
   The value returned if successful.

Remarks

The event handler for this event can be explicitly set using the ontransformnode
property.

This member is an extension of the Worldwide Web Consortium (W3C)
Document Object Model (DOM).

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

ontransformnode Property

Applies to: DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Refer
MSXML Helper APIs

This section contains the following topics.

- XML Helper Objects/Interfaces
- XML Helper Methods
- XML Helper Properties
- Digital Signatures

Other Resources

Web & Internet Samples
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
XML Helper Objects/Interfaces

The XML Helper objects/interfaces are used with both DOM and SAX objects and interfaces.

The following table lists the XML Helper objects/interfaces.

<table>
<thead>
<tr>
<th>Object/Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMXNamespaceManager</td>
<td>Provides an interface for C++ namespace management functionality.</td>
</tr>
<tr>
<td>IMXNamespacePrefixes</td>
<td>Provides a collection of namespace prefixes for a Microsoft® Visual Basic® implementation of IMXNamespaceManager.</td>
</tr>
<tr>
<td>IVBMXNamespaceManager</td>
<td>Provides an interface for namespace management functionality.</td>
</tr>
</tbody>
</table>

The following share properties and methods with DOM objects/interfaces.

<table>
<thead>
<tr>
<th>Object/Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXMLHTTPRequest</td>
<td>Provides client-side protocol support for communication with HTTP servers.</td>
</tr>
<tr>
<td>IXSLProcessor</td>
<td>Used for transformations with compiled style sheets.</td>
</tr>
<tr>
<td>IXSLTemplate</td>
<td>Used for caching compiled XSL Transformations (XSLT) templates.</td>
</tr>
<tr>
<td>IServerXMLHTTPRequest</td>
<td>Provides methods and properties for establishing HTTP connections between items on different Web servers.</td>
</tr>
</tbody>
</table>

Other Resources

Web & Internet Samples
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
**IMXNamespaceManager**

Provides a convenient way to manage namespaces, namespace prefixes, and namespace URIs for an XML document. With IMXNamespaceManager, namespace contexts are organized as a stack of namespace declarations. Each push operation pushes a namespace declaration or set of namespace declarations onto the stack. Each pop operation discards everything from the top of the stack. Namespace declarations on the stack have both current and local context.

**Remarks**

IMXNamespaceManager provides these duplicate methods for C++ processing to avoid the use of BSTR objects for C++. It is not necessary to use BSTR objects because string values are returned in a character buffer.

IMXNamespaceManager performs a search from the top of a stack through all of the namespace declarations, until a declaration is found, or until the bottom of the stack is reached.

For C++, all [in] BSTR parameters can be passed as null-terminated wchar_t*.

All indexes are continuous, and start from 0.

IMXNamespaceManager checks [in] arguments for prefixes to ensure that the prefix name is valid according to the W3C Namespaces in XML Recommendation. IMXNamespaceManager does not check URI arguments.

**Versioning**

MSXML 4.0 and later

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]
Header and IDL files: mxml2.h, mxml2.idl

Remarks

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IMXNamespaceManager Members | IMXNamespacePrefixes

Other Resources

W3C Namespaces in XML Recommendation
IMXNamespaceManager Members

Properties

**allowOverride**
Controls whether the declarePrefix() method allows duplicate prefixes in the local context.

Methods

**declarePrefix**
Binds the namespace prefix with the namespace URI in the current context.

**getDeclaredPrefix**
Returns the indexed prefix in the current context.

**getPrefix**
Finds the indexed prefix associated with the provided URI.

**getURI**
Returns the namespace URI for the provided prefix and DOM node.

**popContext**
Pops local context from the top of the stack.

**pushContext**
Pushes local context down the stack.

**pushNodeContext**
Pushes local context down the stack, and creates a new local context that is defined by a DOM node.

**reset**
Clears the stack of contexts, emptying the current context.

Events

None.

See Also
IMXNamespacePrefixes

Provides a collection of namespace prefixes for the Microsoft® Visual Basic® and Script implementations of IMXNamespaceManager.

Versioning

MSXML 4.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

Remarks

None.

See Also

IMXNamespacePrefixes Members | IMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
IMXNamespacePrefixes Members

Properties

item
Returns a prefix from the indicated index in the prefixes collection.

length
Length of the prefixes collection.

Methods

None.

Events

None.

See Also

IMXNamespaceManager
IVBMXNamespaceManager

Provides a convenient way to manage namespaces, namespace prefixes, and namespace URIs for an XML document. With IVBMXNamespaceManager, namespace contexts are organized as a stack of namespace declarations. Each push operation pushes a namespace declaration or set of namespace declarations onto the stack. Each pop operation discards everything from the top of the stack. Namespace declarations on the stack have both current and local context.

**Current context**
Set of all namespace declarations on the stack.

```csharp
oNSMgr.declarePrefix("a", "urn:same uri");
oNSMgr.pushContext();
oNSMgr.declarePrefix("b", "urn:new uri");
```

The current context holds the prefixes "a" and "b".

**Local context**
Set of all namespace declarations from the top of the stack to the last pushContext method.

```csharp
oNSMgr.declarePrefix("a", "urn:same uri");
oNSMgr.pushContext();
oNSMgr.declarePrefix("b", "urn:new uri");
```

The local context holds the prefix "b".

**Example**

The following example shows how to use IVBMXNamespaceManager to manage namespace declarations.

**XML file (test.xml)**

```xml
<?xml version="1.0" ?>
  <y Name="a:firstele"/>
  <y ref="a:firstele"/>
  <y Name="b:firstele"/>
```
Visual Basic form code (NSMgr.frm)

Option Explicit

Private Sub Form_Resize()
    ' Resize the text box to the size of the form.
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Private Sub Form_Load()
    ' Resize the text box to the size of the form.
    Text1.Top = 100
    Text1.Left = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
    Text1 = ""
    ' Process the attributes in the nodes selected using XPath.
    ProcessNodes "x/y[@Name]", "Name"
    ProcessNodes "x/y[@ref]", "ref"
End Sub

Private Sub ProcessNodes(strXPathQuery As String, sAttrName As String)
    Dim oNodeList As IXMLDOMSelection
    Dim curNode As IXMLDOMNode
    Dim xmldoc As New DOMDocument50
    Dim oNSMgr As New MXNamespaceManager50
    Dim n, x As Long
    Dim str, strName, strValue, strPrefix, strSuffix, strURI As String
    xmldoc.Load App.Path & "\test.xml"
    ' If the element nodes you are selecting are namespace-prefixed,
    ' be sure to set the following additional second-level
    ' DOM selection properties:
    ' xmldoc.setProperty "SelectionLanguage", "XPath"
    ' xmldoc.setProperty "SelectionNamespaces", _
    ' "xmlns:a='http://www.example.microsoft.com/a'"
    Set oNodeList = xmldoc.selectNodes(strXPathQuery)
    ' Processes the nodes that have the named attribute specified.
    For n = 0 To (oNodeList.length - 1)
        Set curNode = oNodeList.Item(n)
        For x = 0 To (curNode.Attributes.length - 1)
            If curNode.Attributes.Item(x).nodeName = sAttrName Then
                strName = curNode.Attributes.Item(x).nodeName
                strValue = curNode.Attributes.Item(x).nodeValue
                strPrefix = Left(strValue, InStr(strValue, ":") - 1)
                strSuffix = Right(strValue, _
Len(strValue) - InStr(strValue, ":")
' Uses the getURIFromNode method of MXNamespaceManager
to get the URI from the declarations of the DOMDoc
' object. This URI is used for the extracted prefix
' current DOM selection.
strURI = oNSMgr.getURIFromNode(strPrefix, curNode)
' Display the output for each namespace URI
' lookup process.
str = "Attribute: " & strName & " [" & _
     CStr(n) & "]" & vbCrLf & _
 "Attribute name: " & strName & vbCrLf & _
 "Namespace prefix: " & strPrefix & vbCrLf & _
 "Namespace URI: " & strURI & vbCrLf & vbCrLf
Text1 = Text1 & str
str = ""
End If
Next
Next
' Clear the DOM object.
Set xmldoc = Nothing
End Sub

Try It!

1. Copy the sample XML file, and paste it into a text file. Save the file as
test.xml.
2. Create a Standard EXE project in Visual Basic. Save the empty project as
NSMgr.vbp to the same directory where you saved test.xml. Save the form
file as NSMgr.frm.
3. Create a reference to MSXML 5.0. To do this, select References... from the
Project menu, and then check the box for Microsoft XML, v5.0.
4. Double click the TextBox icon from the tools menu. A TextBox control will
appear on the project's form named "Text1". In the Properties window, set
the value for the ScrollBars property to "2 – Vertical" and the value of the
MultiLine property to "True".
5. Copy the Visual Basic form code above, and paste it into the code editor to
replace whatever is already there.
6. Execute the code by selecting Start from the Run menu.
7. Verify that the output is similar to that listed in below. You can resize the
form as needed to view the output text more easily.

Output

When you build and run the previous example in Visual Basic, you will see text
output similar to the following. This demonstrates that for each XPath selected node list and named attribute, the namespace URI was looked up using MXNamespaceManager and the getURIFromNode method it provides.

Attribute: Name [0]
Attribute name: Name
Namespace prefix: a
Namespace URI: http://www.example.microsoft.com/a

Attribute: Name [1]
Attribute name: Name
Namespace prefix: b
Namespace URI: http://www.example.microsoft.com/b

Attribute: ref [0]
Attribute name: ref
Namespace prefix: a
Namespace URI: http://www.example.microsoft.com/a

Attribute: ref [1]
Attribute name: ref
Namespace prefix: b
Namespace URI: http://www.example.microsoft.com/b

Remarks

IMXNamespaceManager provides duplicate methods for C++ processing, when necessary, to avoid the use of BSTR objects for C++.

IVBMXNamespaceManager performs a search from the top of stack through all namespace declarations, until a declaration is found or until the bottom of the stack is reached.

All indexes are continuous and start from 0.

IMXNamespaceManager checks [in] arguments for prefixes to ensure a valid prefix name according to the W3C Namespaces in XML Recommendation. IMXNamespaceManager does not check URI arguments.

Versioning

MSXML 4.0 and later
Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Version-Dependent ProgID: MSXML2.MXNamespaceManager50

Version-Dependent CLSID: 88d969d6-f192-11d4-a65f-0040963251e5

Remarks

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IVBMXNamespaceManager Members | IMXNamespaceManager Members | IMXNamespacePrefixes

Other Resources

W3C Namespaces in XML Recommendation
IVBMXNamespaceManager Members

Properties

allowOverride

Controls whether the declarePrefix() method allows duplicate prefixes in the local context.

Methods

declarePrefix

Binds the namespace prefix with the namespace URI in the current context.

getDeclaredPrefixes

Returns a collection of declared namespace prefixes in the current context.

getPrefixes

Returns a collection of namespace prefixes bound to the specified URI.

getURI

Returns the namespace URI for the specified prefix.

getURIFromNode

Returns the namespace URI for the provided prefix in the context of a DOM node.

popContext

Pops local context from the top of the stack.

pushContext

Pushes local context down the stack.

pushNodeContext

Pushes local context down the stack while creating a new local context that is defined by a DOM node.

reset

Clears the stack of contexts, emptying the current context.

Events
None.

See Also

IVBMXNamespaceManager | IMXNamespaceManager | IMXNamespacePrefixes
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
IXMLHTTPRequest

Provides client-side protocol support for communication with HTTP servers.

[Script]

Example

The following Microsoft® JScript® example posts a DOMDocument containing order information to an Active Server Pages (ASP) page on a server and returns the result as a new XML document.

```script
def PostOrder(xmldoc):
    var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
                 xmlhttp.Send(xmldoc);
    return xmlhttp.responseXML;
}
</script>

The ASP page on the server loads the posted XML document, processes the order, and builds a resulting XML document.

```%@
Response.Expires = -1000;
// Load the posted XML document.
var doc = Server.CreateObject("Msxml2.DOMDocument.5.0");
doc.load(Request);
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    var result = Server.CreateObject("Msxml2.DOMDocument.5.0");
    // Now process the order and build the result document.
    Response.ContentType = "text/xml"
    result.save(Response);
}%>

[Visual Basic]
Example

The following Microsoft Visual Basic® example creates an XMLHttpRequest object and asks a server for an XML document. The server sends back an XML document that is displayed by the code snippet.

```vbasic
Private Sub Command1_Click()
    Dim HttpReq As New MSXML2.XMLHTTP5.0
    HttpReq.open "GET", "http://XMLSampleServer/CatalogServer.asp", False
    HttpReq.send
    MsgBox HttpReq.responseText
End Sub
```

[C/C++]

Example

The following C/C++ example creates an XMLHttpRequest object and asks a server for an XML document. The server sends back an XML document that is displayed by the code snippet.

```c/c++
#import "msxml5.dll"
using namespace MSXML2;

void XMLHttpRequestSample()
{
    IXMLHttpRequestPtr pIXMLHTTPRequest = NULL;
    BSTR bstrString = NULL;
    HRESULT hr;
    try {
        hr=pIXMLHTTPRequest.CreateInstance("Msxml2.XMLHTTP.5.0");
        SUCCEEDED(hr) ? 0 : throw hr;
        hr=pIXMLHTTPRequest->open("GET", "http://XMLSampleServer/CatalogServer.asp", false);
        SUCCEEDED(hr) ? 0 : throw hr;
        hr=pIXMLHTTPRequest->send();
        SUCCEEDED(hr) ? 0 : throw hr;
        bstrString=pIXMLHTTPRequest->responseText;
        MessageBox(NULL, _bstr_t(bstrString), _T("Results"), MB_OK);
    }
    catch (Exception e)
    {
        MessageBox(NULL, e.message, _T("Error"), MB_OK);
    }
}
```
if(bstrString)
{
    ::SysFreeString(bstrString);
    bstrString = NULL;
}

} catch (...) {
    MessageBox(NULL, _T("Exception occurred"), _T("Error"), MB_OK)
    if(bstrString)
    ::SysFreeString(bstrString);
}

Remarks

A client computer can use the XMLHttpRequest object (MSXML2.XMLHTTP.5.0) to send an arbitrary HTTP request, receive the response, and have the Microsoft® XML Document Object Model (DOM) parse that response.

This object is integrated with Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office to support sending the request body directly from, and parsing the response directly into, the MSXML DOM objects. When combined with the support for Extensible Stylesheet Language (XSL), the XMLHttpRequest component provides an easy way to send structured queries to HTTP servers and efficiently display the results with a variety of presentations.

The usual sequence is to call the open method, set any custom header information through the setRequestHeader method followed by the send method, and then to check one of the four different response properties.

The XMLHttpRequest object is supported in Microsoft Internet Explorer 5.0 or later.

Versioning

MSXML 2.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib
[C/C++]

**Header and IDL files**: msxml2.h, msxml2.idl

**Version-Dependent ProgID**: Msxml2.XMLHTTP.5.0

**Version-Dependent CLSID**: 88D969C5-F192-11D4-A65F-0040963251E5

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**

[IXMLHTTPRequest Members](#) | [IXMLDOMDocument/DOMDocument](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
IXMLHTTPRequest Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from the IDispatch interface.

**Properties**

- **onreadystatechange*** Specifies the event handler to be called when the readyState property changes. Read/write.
- **readyState** Represents the state of the request. Read-only.
- **responseBody** Represents only one of several forms in which the HTTP response can be returned. Read-only.
- **responseStream** Represents only one of several forms in which the HTTP response can be returned. Read-only.
- **responseText** Represents the response entity body as a string. Read-only.
- **responseXML** Represents the parsed response entity body. Read-only.
- **status** Represents the HTTP status code returned by a request. Read-only.
- **statusText** Represents the HTTP response line status. Read-only.

* Denotes an extension to the World Wide Web Consortium (W3C) DOM.

**Methods**

- **abort** Cancels the current HTTP request.
- **getAllResponseHeaders** Retrieves the values of all the HTTP headers.
  - Retrieves the value of an HTTP header from the
getResponseHeader  response body.

open

Initializes an MSXML2.XMLHTTP request, and specifies the method, URL, and authentication information for the request.

send

Sends an HTTP request to the server and receives a response.

setRequestHeader

Specifies the name of an HTTP header.

Events

None.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLHTTPRequest
IXSLProcessor

Used for transformations with compiled style sheets.

[C/C++]

IXSLProcessor is designed to handle asynchronous transformations. In this case, multiple calls to the `transform` method must be made, in which each call will transform as much as it can based on what is available from the input. In this scenario, the processor remains in READYSTATE_INTERACTIVE until the transform is complete.

[Script]

Example

```javascript
var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0")
var xslProc;
xslDoc.async = false;
xslDoc.load("sample2.xsl");
xslt.stylesheet = xslDoc;
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");
xslProc = xslt.createProcessor();
xslProc.input = xmlDoc;
xslProc.transform();
alert(xslProc.output);
```

Sample2.xsl

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/">
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">\n  </xsl:template>
</xsl:stylesheet>
```
[Visual Basic]

Example

Dim xslt As New Msxml2.XSLTemplate50
Dim xslDoc As New Msxml2.FreeThreadedDOMDocument50
Dim xmlDoc As New Msxml2.DOMDocument50
Dim xslProc As IXSLProcessor
xslDoc.async = False
xslDoc.Load "sample2.xsl"
Set xslt.stylesheet = xslDoc
xmlDoc.async = False
xmlDoc.Load "books.xml"
Set xslProc = xslt.createProcessor()
xslProc.input = xmlDoc
xslProc.Transform
MsgBox xslProc.output

Sample2.xsl

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:param name="param1"/>
  <xsl:template match="/"/>
    Hello
  </xsl:template>
  <xsl:template match="/" mode="edit">
    In Edit Mode
  </xsl:template>
  <xsl:template match="/" mode="view">
    In View Mode
  </xsl:template>
</xsl:stylesheet>

[C/C++]

Example

#import "msxml5.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface) \
if(pInterface) {
    pInterface->Release();
pInterface = NULL;
}

#define RELEASE(pInterface) 
    { 
    pInterface->Release();
pInterface = NULL;
    }

BOOL XSLProcessorDemo()
{
    BOOL bResult = FALSE;
    short sResult = FALSE;
    HRESULT hr;
    IXMLDOMDocument2 *pStyleSheet=NULL;
    IXMLDOMDocument2 *pDOMObject=NULL;
    IXSLTemplate *pIXSLTemplate=NULL;
    IXSLProcessor *pIXSLProcessor=NULL;
    VARIANT varValue;

    try
    {
        hr = CoCreateInstance(CLSID_XSLTemplate, NULL, CLSCTX_SERVER,
            IID_IXSLTemplate, (LPVOID*)(&pIXSLTemplate));
        SUCCEEDED(hr) ? 0 : throw hr;

        if(pIXSLTemplate)
        {
            hr=CoCreateInstance(CLSID_FreeThreadedDOMDocument, NULL,
                CLSCTX_SERVER, IID_IXMLDOMDocument2,
                (LPVOID*)(&pStyleSheet));
            SUCCEEDED(hr) ? 0 : throw hr;

            if(pStyleSheet)
            {
                hr=pStyleSheet->put_async(VARIANT_FALSE);
                if(SUCCEEDED(hr))
                {
                    hr=pStyleSheet->load(_variant_t
                        (_T("d:\inetpub\wwwroot\samplexsl.xml")), &sRes
                        if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
                    {
                        hr=pIXSLTemplate->putref_stylesheet(pStyleSheet);
                        if(SUCCEEDED(hr))
                        {
                            hr=pIXSLTemplate->createProcessor(&pIXSLProcessor
                        }
                    }
                }
            }
        }
    }

    catch(...) { hr=E_FAIL; }
    return hr==S_OK;
SUCCEEDED(hr) ? 0 : throw hr;
if(pIXSLProcessor)
{
    hr=CoCreateInstance(CLSID_DOMDocument, NULL, CLSCTX_SERVER, IID_IXMLDOMDocument2, (LPVOID*)&pIXMLDOMDocument);
    SUCCEEDED(hr) ? 0 : throw hr;
}
if(pIXMLDOMDocument)
{
    hr=pIXMLDOMDocument->put_async(VARIANT_FALSE);
    if(SUCCEEDED(hr))
    {
        hr=pIXMLDOMDocument->load(_variant_t(_T("d:\inetpub\wwwroot\sampleXSLWithObject.xml")));
        if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
        {
            hr=pIXSLProcessor->put_input(_variant_t(pIXMLDOMDocument));
            if(SUCCEEDED(hr))
            {
                hr=CoCreateInstance(CLSID_FreeThreadedDOMDocument, NULL, CLSCTX_SERVER, IID_IXMLDOMDocument2, (LPVOID*)&pDOMObject);
                if(SUCCEEDED(hr) && pDOMObject)
                {
                    hr=pDOMObject->put_async(VARIANT_FALSE);
                    if(SUCCEEDED(hr))
                    {
                        hr=pDOMObject->loadXML(_T("<foo>foo</foo>"), &sResult);
                        if(SUCCEEDED(hr))
                        {
                            hr=pIXSLProcessor->addObject(pDOMObject, _T("urn:object"));
                            if(SUCCEEDED(hr))
                                bResult=TRUE;
                        }
                    }
                }
            }
        }
    }
}
hr=pIXSLProcessor->transform(&sResult);
if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))

The style sheet — "d:\inetpub\wwwroot\sampleXSLWithObject.xml"

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:myObj="urn:my-object"
    version="1.0">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
        <xsl:element name="bar">
            <xsl:value-of select="myObj:get-text()"/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Output (in a message box)

```xml
<?xml version="1.0" encoding="UTF-16"?>
```
Remarks

The createProcessor method returns IXSLProcessor. The processor has a transform method that takes the input data, applies the XSL Transformations (XSLT) style sheet defined in the IXSLTemplate, and writes the result to the specified output stream.

This XSL processor is completely independent of the transformNode and transformNodeToObject methods on DOMDocument. In particular, when you transform XML using IXSLProcessor, the ontransformnode event is not fired on DOMDocument.

Versioning

MSXML 2.6 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Version-Dependent ProgID: Msxml2.XSLTemplate.

Version-Dependent CLSID: 88d969c3-f192-11d4-a65f-0040963251e5

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXSLProcessor Members
IXSLProcessor Members

The methods and properties in this table are an extension to the World Wide Web Consortium (W3C) DOM.

### Properties

- **input**: Specifies which XML input tree to transform. Read/write.
- **output**: Gets a custom output to write the result of the transformation. Read/write.
- **readyState**: Returns the current state of the processor. Read-only.
- **startMode**: Returns the base name part of the start mode, a qualified name. Read-only.
- **startModeURI**: Returns the namespace Uniform Resource Identifier (URI) part of the start mode, the qualified name. Read-only.
- **stylesheet**: Provides the Extensible Stylesheet Language (XSL) style sheet that is to be compiled into the XSL template. Read-only.
- **ownerTemplate**: Returns the style sheet template that was used to create IXSLProcessor.

### Methods

- **addObject**: Adds objects into a style sheet.
- **addParameter**: Adds parameters into an XSL Transformations (XSLT) style sheet.
- **reset**: Resets the state of the processor to the state it was in prior to calling transform.
setStartMode

Performs subsets of larger XSLT transformations by selecting the XSLT mode with which to start. This minimizes the amount of XSLT processing.

transform

Starts the transformation process or resumes a previously failed transformation.

Events

None.

See Also

IXSLProcessor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
IXSLTemplate

Used for caching compiled XSL Transformations (XSLT) templates.

[C/C++]

Example

The following C++ example demonstrates the use of a single style sheet with multiple XML files. The style sheet is loaded once and cached into the IXSLTemplate object.

```cpp
# import "msxml5.dll"
using namespace MSXML2;

#define CHECK_AND_RELEASE(pInterface)  
if(pInterface)  
{  
pInterface->Release();  
pInterface = NULL;  
}  
#define RELEASE(pInterface)  
{
  pInterface->Release();  
pInterface = NULL;
}

BOOL XSLTemplateDemo()
{
  BOOL bResult = FALSE;
  short sResult = FALSE;
  IXMLDOMDocument2 *pStyleSheet=NULL;
  IXMLDOMDocument2 *pIXMLDOMDocument2=NULL;
  IXSLTemplate *pIXSLTemplate=NULL;
  IXSLProcessor *pIXSLProcessor=NULL;
  IXMLDOMDocument2 *pIXMLDOMDocument=NULL;
  HRESULT hr;
  VARIANT varValue;

  try
  {
```
hr = CoCreateInstance(CLSID_XSLTemplate50, NULL, CLSCTX_SERVER,
IID_IXSLTemplate, (LPVOID*)(&pIXSLTemplate));
SUCCEEDED(hr) ? 0 : throw hr;

if(pIXSLTemplate)
{
    hr=CoCreateInstance(CLSID_FreeThreadedDOMDocument50, NULL,
    CLSCTX_SERVER, IID_IXMLDOMDocument2,
    (LPVOID*)(&pStyleSheet));
    SUCCEEDED(hr) ? 0 : throw hr;

    if(pStyleSheet)
    {
        hr=pStyleSheet->put_async(VARIANT_FALSE);
        if(SUCCEEDED(hr))
        {
            hr=pStyleSheet->load(_variant_t(_T("d:\inetpub\wwwroot\samplexsl.xml")), &sResult);
            if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
            {
                hr=pIXSLTemplate->putref_stylesheet(pStyleSheet);
                if(SUCCEEDED(hr))
                {
                    hr=pIXSLTemplate->createProcessor(&pIXSLProcessor);
                    SUCCEEDED(hr) ? 0 : throw hr;

                    if(pIXSLProcessor)
                    {
                        hr=CoCreateInstance(CLSID_DOMDocument50, NULL,
    CLSCTX_SERVER, IID_IXMLDOMDocument2,
    (LPVOID*)(&pIXMLDOMDocument));
                        SUCCEEDED(hr) ? 0 : throw hr;

                        if(pIXMLDOMDocument)
                        {
                            hr=pIXMLDOMDocument->put_async(VARIANT_FALSE);
                            if(SUCCEEDED(hr))
                            {
                                hr=pIXMLDOMDocument->load(_variant_t(_T("d:\inetpub\wwwroot\samplexmlldtd.xml")), &sResult);
                                if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
                                {
                                    hr=pIXSLProcessor->put_input(_variant_t(pIXMLDOMDocument));
                                    if(SUCCEEDED(hr))
                                    {
                                        hr=pIXSLProcessor->transform(_variant_t(&sResult));
                                        if(SUCCEEDED(hr)&&(sResult==VARIANT_TRUE))
pIXSLProcessor->get_output(&varValue);
::MessageBox(NULL,_bstr_t(varValue),_T("Transformed Output"), MB_OK);

RELEASE(pIXMLDOMDocument);

// Load another document.
hrc=CoCreateInstance(CLSID_DOMDocument50, NULL,
CLSCTX_SERVER, IID_IXMLDOMDocument2,
(LPVOID*)(&pIXMLDOMDocument2));
SUCCEEDED(hr) ? 0 : throw hr;

if(pIXMLDOMDocument2)
{
hr=pIXMLDOMDocument2->put_async(VARIANT_FALSE);
if(SUCCEEDED(hr))
{
// Load some XML into the document.
hr=pIXMLDOMDocument2->loadXML(_T("<COLLECTION xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <DATE dt:dt="datetime">1998-10-13T15:56:00</DATE>
  <BOOK><TITLE>Lover Bird</TITLE>
    <AUTHOR>Cynthia Randall</AUTHOR>
    <PUBLISHER>Lucerne Publishing</PUBLISHER>
  </BOOK>
</COLLECTION>"), &sResult);
if(SUCCEEDED(hr) && (sResult==VARIANT_TRUE))
{
    // Use the same processor.
    hr=pIXSLProcessor->put_input(_varia
    pIXMLDOMDocument2));
    if(SUCCEEDED(hr))
    {
        hr=pIXSLProcessor->transform( &sResult);
        if(SUCCEEDED(hr)&&(sResult==VARIANT_TRUE))
        {
            pIXSLProcessor->get_output(&varValue);
        }
    }
}
Remarks

Before MSXML 2.6, every call to the transformNode or transformNodeToObject method had to recompile the style sheet because the style sheet was simply passed as an IXMLDOMNode. You can increase performance by caching the compiled style sheet and reusing it with the IXSLTemplate object.

To cache a compiled XSLT style sheet, load an XSLT style sheet into an IXSLTemplate object. This object is free-threaded and stateless, so it can be stored in shared Active Server Pages (ASP) application state. Then to transform a given document using this template, create an IXSLProcessor object using the createProcessor method. The IXSLProcessor object stores the state for one transform call and has a rental-threading model.
**Note** In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

This object is an extension of the World Wide Web Consortium (W3C) Document Object Model (DOM).

**Versioning**

MSXML 2.6 and later

**Requirements**

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Version-Dependent ProgID:** Msxml2.XSLTemplate.5.0

**Version-Dependent CLSID:** 88D969C3-F192-11D4-A65F-0040963251E5

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

IXSLTemplate Members
IXSLTemplate Members

The methods and properties in this table are an extension to the World Wide Web Consortium (W3C) DOM.

Properties

- **stylesheet**
  Provides the Extensible Stylesheet Language (XSL) style sheet to compile into an XSL template.

Methods

- **createProcessor**
  Creates a rental-model `XSLProcessor` that will use this template.

Events

None.

See Also

- **IXSLTemplate**
IServerXMLHTTPRequest/ServerXMLHTTP

Provides methods and properties that enable you to establish an HTTP connection between files or objects on different Web servers.

The ServerXMLHTTP object offers functionality similar to that of the XMLHTTP object. Unlike XMLHTTP, however, the ServerXMLHTTP object does not rely on the WinInet control for HTTP access to remote XML documents. ServerXMLHTTP uses a new HTTP client stack. Designed for server applications, this server-safe subset of WinInet offers the following advantages:

- **Reliability** — The HTTP client stack offers longer uptimes. WinInet features that are not critical for server applications, such as URL caching, auto-discovery of proxy servers, HTTP/1.1 chunking, offline support, and support for Gopher and FTP protocols are not included in the new HTTP subset.

- **Security** — The HTTP client stack does not allow a user-specific state to be shared with another user's session. ServerXMLHTTP provides support for client certificates.

  **Note**  For more information about the limitations of the WinInet control, see [WinInet Limits Connections Per Server](#).

**Usage**

The ServerXMLHTTP object is commonly used to:

- Receive XML documents from an Active Server Pages (ASP) page on a local or remote Web server (HTTP GET).
- Post XML documents to an ASP page on a local or remote Web server (HTTP POST).
- Post and process response XML documents from an ASP page (HTTP POST).

You can use the ServerXMLHTTP object either indirectly using the setProperty method of DOMDocument or directly using the ServerXMLHTTP object itself. For examples of these two approaches, as well as examples of how to use the
ServerXMLHTTP security options and how to use ServerXMLHTTP in a multitiered environment, see [IServerXMLHTTPRequest/ServerXMLHTTP Examples](#).

**Remarks**

The ServerXMLHTTP `open` method makes the connection between servers and the `send` method sends the request.

You can read the response using one of the four following properties: `responseBody`, `responseStream`, `responseText`, and `responseXML`.

[C/C++]

With ServerXMLHTTP, the usual sequence is to call the `open` method, set any custom header information through the `setRequestHeader` method, call the `send` method, and then check one of the four response properties.

For example, an ASP, Microsoft® Visual Basic®, or C++ component on a server computer can send an HTTP request to another server and then receive the response as a stream or XML document object. The response can then be fed to downstream clients, saved to a file on the server, or combined with other XML data (potentially collected from other Web servers).

ServerXMLHTTP offers additional benefits for transporting XML data:

- **ServerXMLHTTP** maintains state, so transactional XML data can be used in business applications that require real-time responses.
- **ServerXMLHTTP** allows you to serve the response object as a stream or Document Object Model (DOM) object. As a stream, this offers considerable performance benefits when moving data through the HTTP protocol.

ServerXMLHTTP offers backward compatibility with XMLHTTP. Source code (JScript, Visual Basic Scripting Edition, Visual Basic, or C++) that uses the XMLHTTP component can be easily modified to use the new ServerXMLHTTP component.

The number of instances of ServerXMLHTTP that can exist simultaneously within a single process primarily depends upon the amount of memory available for
applications on the system. However, other factors, such as CPU processing capacity, or available socket connections can further limit the number of instances that can be active simultaneously.

The IServerXMLHTTPRequest interface inherits from IXMLHTTPRequest and extends it with the following four new methods: `getOption`, `setOption`, `waitForResponse`, and `setTimeout`.

**Versioning**

MSXML 3.0 and later

**Requirements**

Supported platforms include Microsoft Windows® 2000, or Microsoft Windows NT® 4.0 with Microsoft Internet Explorer 5.01 (or later) installed. ServerXMLHTTP fails on other platforms, such as Microsoft Windows 98.

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and IDL files:** msxml2.h, msxml2.idl

**Version-Dependent ProgID:** Msxml2.ServerXMLHTTP.5.0

**Version-Dependent CLSID:** 88D969C6-F192-11D4-A65F-0040963251E5

**See Also**

[IXMLHTTPRequest](#) | IServerXMLHTTPRequest/ServerXMLHTTP Members
IServerXMLHTTPRequest/ServerXMLHTTPRequest Members

[Visual Basic, Script]

The following tables show the properties, methods, and events.

[C/C++]

The following tables show the properties, methods, and events.

This interface inherits from IXMLHTTPRequest.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onreadystatechange</td>
<td>Specifies the event handler to be invoked when the readyState property changes. Read/write.</td>
</tr>
<tr>
<td>responseBody</td>
<td>Represents the response entity body as an array of unsigned bytes. Read-only.</td>
</tr>
<tr>
<td>readyState</td>
<td>Represents the state of the request. Read-only.</td>
</tr>
<tr>
<td>responseStream</td>
<td>Represents the response entity body as an IStream. Read-only.</td>
</tr>
<tr>
<td>responseText</td>
<td>Represents the response entity body as a string. Read-only.</td>
</tr>
<tr>
<td>responseXML</td>
<td>Represents the response entity body as parsed by Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office. Read-only.</td>
</tr>
<tr>
<td>status</td>
<td>Represents the HTTP status code returned by a request. Read-only.</td>
</tr>
<tr>
<td>statusText</td>
<td>Represents the HTTP response line status. Read-only.</td>
</tr>
</tbody>
</table>

Methods

Note  The following method names are case-sensitive when used with ServerXMLHTTP, unlike XMLHTTP method names, which are not.
abort  Cancels the current HTTP request.
getAllResponseHeaders  Retrieves the values of all the HTTP headers.

Returns the value of one of the following options:

- SXHOPTION_URL
- SXHOPTION_URL_CODEPAGE
- SXHOPTION_ESCAPE_PERCENT_IN_URL
- SXHOPTION_IGNORE_SERVER_SSL_CERT
- SXHOPTION_SELECT_CLIENT_SSL_CERT

getOption  Retrieves the value of an HTTP header from the response body.

getOption  Initializes a request and specifies the method, URL, and authentication information for the request.

send  Sends an HTTP request to the server and receives a response.

Sets one of the following options:

- SXHOPTION_URL_CODEPAGE
- SXHOPTION_ESCAPE_PERCENT_IN_URL
- SXHOPTION_IGNORE_SERVER_SSL_CERT
- SXHOPTION_SELECT_CLIENT_SSL_CERT

setOption  Specify proxy configuration.

setProxy  Specify proxy authentication credentials.

setProxyCredentials  Specifies timeout settings for resolving the domain name, establishing the connection to the server, sending the data, and receiving the response.

setTimeouts  Specifies the name of an HTTP header.

setRequestHeader  Allows the requesting server to suspend execution while asynchronous send operation to complete.

Events

None.
IServerXMLHTTPRequest/ServerXMLHTTPRequest Examples

This section contains the following topics.

- Using ServerXMLHTTP Directly
- Using the ServerXMLHTTP Security Options
- Using ServerXMLHTTP in a Multitiered Environment
- Using the WinHTTP Proxy Configuration Utility
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Refer
Using ServerXMLHTTP Directly

Using the ServerXMLHTTP object directly offers much greater procedural control than that of the setProperty method of DOMDocument. Instead of merely retrieving XML responses from a remote server, the ServerXMLHTTP object allows developers to use the HTTP methods, GET and POST, as well as the ability to handle basic security logons.
Using ServerXMLHTTP to GET XML results from an ASP page

In the following example, ServerXMLHTTP retrieves an XML response from an ASP page over HTTP. By using GET, the example is able to send a request without actually transferring any data to the Web server.

The responseXML property of the objSrvHTTP object contains the XML response from the Web server. The example program writes this response to the browser's output by first informing the browser that the response to be received is XML ("text/xml"). Next, the program passes the response directly to the Response object for displaying onscreen.

Example

```<%@language=JScript%>
<%
    var objSrvHTTP;
    objSrvHTTP = Server.CreateObject ("Msxml2.ServerXMLHTTP.5.0");
    objSrvHTTP.open ("GET","http://someotherserver/respond.asp", false);
    objSrvHTTP.send ();
    Response.ContentType = "text/xml";
    Response.Write (objSrvHTTP.responseXML.xml);
%
```
Using ServerXMLHTTP to POST XML to an ASP page

In the following example, ServerXMLHTTP uses HTTP to send XML data to an Active Server Page (ASP) page. Unlike the preceding example that used GET, this example uses POST. The POST method sends data, along with the request, to the Web server.

As in the previous example, the program writes the response to the browser's output by setting the ContentType to "text/xml" and passing the response to the Response object.

Example

<%@language=JScript%>
<%
    var objSrvHTTP;
    var objXMLDocument;
    objSrvHTTP = Server.CreateObject ("Msxml2.ServerXMLHTTP.5.0");
    objXMLDocument = Server.CreateObject ("Msxml2.DOMDocument.5.0");

    objXMLDocument.async= false;
    objXMLDocument.loadXML ("<msg><id>1</id></msg>");

    objSrvHTTP.open ("POST","http://someotherserver/respond.asp",false);
    objSrvHTTP.send (objXMLDocument);
    Response.ContentType = "text/xml";
    Response.Write (objSrvHTTP.responseXML.xml);
%>
Using ServerXMLHTTP to POST and Process XML

In the following example, ServerXMLHTTP sends an XML document to an ASP page on a Web server and passes its XML response to a waiting DOMDocument for processing.

Example

<%@language=Jscript%>
<%
    var objSrvHTTP;
    var objXMLSend;
    var objXMLReceive;
    objSrvHTTP = Server.CreateObject("Msxml2.ServerXMLHTTP.5.0");
    objXMLSend = Server.CreateObject("Msxml2.DOMDocument.5.0");
    objXMLReceive = Server.CreateObject("Msxml2.DOMDocument.5.0");

    objXMLSend.async = false;
    objXMLSend.loadXML ("<msg><id>2</id></msg>");

    objSrvHTTP.open ("POST","http://someotherserver/respond.asp",false)
    objSrvHTTP.send (objXMLSend);
    objXMLReceive = objSrvHTTP.responseXML;
    Response.ContentType = "text/xml";
    Response.Write (objXMLReceive.xml);
%>
Using the ServerXMLHTTP Security Options

The bstrUser and bstrPassword parameters of the open method of ServerXMLHTTP allow users to log onto Web servers secured with basic authentication. Both the bstrUser and bstrPassword parameters are sent in clear text (data that is transmitted as clearly readable text characters).

The following example demonstrates how to use the optional bstrUser and bstrPassword parameters of the open method of ServerXMLHTTP to retrieve an XML document from a Web server secured with basic authentication.

Example

```vbscript
<%@language=Jscript%>
<%
    var objSrvHTTP;
    objSrvHTTP = Server.CreateObject("Mxsvm2.ServerXMLHTTP.5.0");
    objSrvHTTP.open ("GET","http://someotherserver/secure.asp",false,
                    "testuser", "testpassword");

    objSrvHTTP.send ();
    Response.ContentType = "text/xml";
    Response.Write (objSrvHTTP.responseXML.xml);
%
```
Using ServerXMLHTTP in a Multitiered Environment

The following example shows how ServerXMLHTTP can be used to access a news service on a backend server and deliver it to a Web client.
Request on the Presentation Tier

A simple Web page, latestnews.asp, contains a link to the ServerXMLHTTP.asp page on the business tier server.

<html>
<head>
    <title>Latest News</title>
</head>

<body>
    <a href="http://myserver/myproject/ServerXMLHttp.asp">Latest News</a>
</body>
</html>
Request on the Business Tier

The ServerXMLHTTP.asp page resides on the business or middle tier. This page contains Microsoft® JScript® that opens a connection to a NewsService.asp page on a server at the data services tier.

```javascript
<%@language=JScript%>

<html>
<body>
<p>Top News Story</p>
<%Response.Write(newsElement.text);%>
</body>
</html>
```
Response on the Data Services Tier

The **NewsService.asp** page resides on the data services or middle tier. This page contains JScript that loads an XML document, `news.xml`, into a DOMDocument object named `Result`, and then saves the object directly to the `Response` object, which is returned to `ServerXMLHTTP.asp` on the business tier.

```jscript
<%@language=JScript%>
<%
  var Result = Server.CreateObject("Msxml2.DOMDocument.5.0");
  Result.async = false;
  Result.load(Server.MapPath("news.xml"));
  Response.ContentType = "text/xml";
  Result.save(Response);
%>

**Note**  A document comprised of the latest news stories collected by a news service can be quite large. For this example, the `news.xml` document loaded by `NewsService.asp` is small and simple.

```xml
<?xml version ="1.0"?>
<news>
  <story1>Here is the top news story…</story1>
  <story2>News is the next news story.</story2>
</news>
```
Response on the Business Tier

ServerXMLHTTP.asp handles the response on the business tier. In this case, ServerXMLHTTP.asp loads the response into the xmlResult (DOMDocument) object. The DOM selects a single node from the document using the selectSingleNode method of DOMDocument, setting the results to the newsElement object. The program then writes the text property of the newsElement object (along with HTML) to the Web page on the requesting client computer on the presentation tier.

<%@language=JScript%>
<% var srvXMLHTTP;  
srvXMLHTTP = Server.CreateObject("Msxml2.ServerXMLHTTP.5.0");  
srvXMLHTTP.open("GET", "http://dataserver/news/NewsService.asp", false);  
srvXMLHTTP.send();  
newsElement = srvXMLHTTP.responseXML.selectSingleNode("/news/story1"); %>

<html>
<body>
<p>Top News Story</p>
<%Response.Write(newsElement.text);%>
</body>
</html>
Response on the Presentation Tier

The user sees the following output in the Web browser.

Top News Story

Here is the top news story…
Using the WinHTTP Proxy Configuration Utility

The WinHTTP proxy configuration utility, `proxycfg.exe`, configures WinHTTP to access HTTP and HTTPS servers through a proxy server. Because the `ServerXMLHTTP` object depends on WinHTTP proxy settings, an administrator can use the `proxycfg.exe` utility as part of the deployment and installation process of an application that uses WinHTTP. The administrator who runs `proxycfg.exe` must have local administrator privileges so that `proxycfg.exe` can update the registry of the local computer. WinHTTP proxy settings are per-machine, not per-user.

**Note**  WinHTTP proxy settings are separate from the proxy settings in Microsoft® Internet Explorer.

Configuration of WinHTTP proxy settings must be done using the `proxycfg.exe` utility; these settings cannot be configured through Control Panel. With the `proxycfg.exe` utility, an administrator can also specify an optional list of host names to bypass the proxy server and be accessed directly. The `proxycfg.exe` tool can also configure WinHTTP to access all servers directly.

**To download the proxy configuration utility**

- To download the proxy configuration utility, go the [XML Development Center](https://msdn.microsoft.com/xmldevcenter).

**Important**  After using `proxycfg.exe` to update the registry, the previous settings cannot be restored.

**Usage**

The following examples show the syntax used for various commands in the `proxycfg.exe` utility.

`proxycfg`  
This command displays the current WinHTTP proxy settings.
proxycfg -d
This command specifies that all HTTP and HTTPS servers should be accessed directly. Use this command if there is no proxy server.

proxycfg -p proxy-server-list optional-bypass-list
This command specifies one or more proxy servers, and an optional list of hosts that should be accessed directly. If a proxy server is not specified for a given protocol and that server is not in the bypass list, the -p option specifies that the server cannot be accessed at all.

proxycfg -d -p proxy-server-list optional-bypass-list
This command specifies one or more proxy servers, and an optional list of hosts that should be accessed directly. If a proxy server is not specified for the given protocol, the -d option specifies that the server should be accessed directly instead.

proxycfg -u
This command imports the Internet Explorer proxy settings of the current user. WinHTTP does not support auto-discovery and configuration script-based proxy settings.

The following table describes how to use both the proxy-server-list and optional-bypass-list parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description of use</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy-server-list</td>
<td>Proxies are specified in a space-delimited string. The proxy listings can contain the port number used to access the proxy. Proxy servers can be listed to use a specific protocol. The valid protocols are HTTP or HTTPS. The syntax for a proxy listing that specifies a protocol is: &quot;protocol=http://proxy_name&quot; where protocol is either http or https and proxy_name is the name of the proxy server. For example, the string, &quot;http=http://http_proxy_name:80&quot;, specifies that the protocol is HTTP, the name of the proxy server is http_proxy_name, and the port number for this server is 80. An example of a proxy listing that specifies the HTTPS protocol is &quot;https=http://https_proxy_name&quot;.</td>
</tr>
</tbody>
</table>
If the proxy server uses the default port number for the protocol, the port number can be omitted.

If a proxy name is listed by itself, the `proxycfg.exe` utility uses that proxy as the default proxy for any protocols that do not have a proxy specified. For example, the listing, "http=http://http_proxy other_proxy", specifies to use the http_proxy server for any HTTP operations and the other_proxy server for any HTTPS operations.

This list contains host names or IP addresses that are locally known.

This list can contain wildcards, "*", that cause the application to bypass the proxy server for addresses that fit the specified pattern. For example, both "*.microsoft.com" and "*.org" are acceptable wildcard patterns. Wildcard characters must be the left-most characters in the list, so "myserver.*" is not supported.

optional-bypass-list

To list multiple addresses and host names, separate them with blank spaces or semicolons in the proxy bypass string. If the "<local>" macro is specified, the function bypasses any host name that does not contain a period.

**Examples**

The following is the most common command used with the `proxycfg.exe` utility. This command specifies to use the proxy server named "itgproxy" to access both HTTP and HTTPS servers, and also bypass any host names that do not contain a period. In this case, the -d option has no effect.

```
proxycfg -d -p itgproxy "<local>"
```

Compare the previous command to the following command which specifies to use the server named "itgproxy" to access both HTTP and HTTPS servers, and not to bypass any servers.
proxycfg -p itgproxy

The following is an example of a more complex command used by the proxycfg.exe utility. This command specifies to use the server named "http_proxy" to access HTTP servers and the server named "https_proxy" to access HTTPS servers. This command also specifies that local intranet sites (for example, host names that do not contain a period) and any site in the "*.microsoft.com" domain bypass the proxy.

proxycfg -p "http=http_proxy https=https_proxy" "<local>;*.microsoft

See Also

IServerXMLHTTPRequest/ServerXMLHTTP Members

Applies To: IServerXMLHTTPRequest/ServerXMLHTTP
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
XML Helper Properties

The following are properties of the XML Helper objects/interfaces.

allowOverride Property
- Controls whether the declarePrefix() method overwrites previously declared prefixes.

item Property
- Returns a prefix from the indicated index in the prefixes collection.

length Property
- Returns the length of the prefixes collection.
allowOverride Property

Controls whether the declarePrefix() method overwrites previously declared prefixes in the same local context.

[Script]

Script Syntax

var bolValue = objNSManager.allowOverride;
objNSManager.allowOverride = True;

Parameters

None.

Return Value

bolValue

A Boolean. The default value is True.

[Visual Basic]

Visual Basic Syntax

bolValue = objNSManager.allowOverride;
objNSManager.allowOverride = True;

Parameters

None.

Return Value

bolValue
A Boolean. The default value is True.

[C/C++]

C/C++ Syntax

HRESULT putAllowOverride(
    VARIANT_BOOL fOverride);
HRESULT getAllowOverride(
    VARIANT_BOOL* fOverride);

Parameters

fOverride [in]
    A Boolean.

fOverride [out, retval]
    A Boolean. The default value is True.

C/C++ Return Values

S_OK
    Always S_OK.

E_POINTER
    The value returned if the fOverride value is NULL.

Remarks

When namespaces are declared in XML document processing, each element will start from pushContext(NULL), and each namespace declaration will result in declarePrefix(prefix, URI). In this case, you might want to prevent the same prefix from being overridden in the same element.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

item Property
[Script, Visual Basic]
Applies to: IVB MXNamespaceManager

[C/C++]
Applies to: IMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
item Property

Returns a prefix from the indicated index in the prefixes collection. The index begins at 0.

[Script]

Script Syntax

var strValue = objNSPrefixes.item(intIndex);

Parameters

intIndex
A long integer. Read-only. The index of the item within the prefixes collection. The index starts at 0.

[Visual Basic]

Visual Basic Syntax

strValue = objNSPrefixes.item(intIndex)

Parameters

intIndex
A long integer. Read-only. The index of the item within the prefixes collection. The index starts at 0.

[C/C++]

C/C++ Syntax

HRESULT get_item(
    long index,
    BSTR* prefix);
Parameters

index [in]
   A long integer. The index of the item within the prefixes collection. The index starts at 0.

prefix [out, retval]
   The return value of the prefix.

C/C++ Return Values

S_OK
   The value returned if successful.

E_POINTER
   The value returned if the prefix value is NULL.

E_FAIL
   The value returned if index value is out of bounds.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

length Property

Applies to: IMXNamespacePrefixes
length Property

Returns the length of the prefixes collection. The collection starts at 0.

[Script]

Script Syntax

var intLength = objNSPrefixes.length

[Visual Basic]

Visual Basic Syntax

intLength = objNSPrefixes.length

[C/C++]

C/C++ Syntax

HRESULT get_length(
    long* length);

Parameters

length[out,retval]

The number of items in the prefix collection. The collection starts at 0.

C/C++ Return Values

S_OK
  The value returned if successful.
E_POINTER
  The value returned if the length is NULL.

Remarks
Read-only.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

item Property

 Applies to: IMXNamespacePrefixes
# XML Helper Methods

The following are methods of the XML Helper objects/interfaces.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>declarePrefix Method</code></td>
<td>Binds the namespace prefix with the namespace URI in the local context.</td>
</tr>
<tr>
<td><code>getDeclaredPrefixes Method</code></td>
<td>Returns a collection of declared namespace prefixes bound to a namespace name.</td>
</tr>
<tr>
<td><code>getPrefixes Method</code></td>
<td>Returns a collection of namespace prefixes bound to the specified URI.</td>
</tr>
<tr>
<td><code>getURI Method</code></td>
<td>Returns the namespace URI for the specified prefix.</td>
</tr>
<tr>
<td><code>getURIFromNode Method</code></td>
<td>Returns the namespace URI for the provided prefix and DOM node.</td>
</tr>
<tr>
<td><code>popContext Method</code></td>
<td>Pops the last local context from the top of the stack.</td>
</tr>
<tr>
<td><code>pushContext Method</code></td>
<td>Pushes the local context down the stack.</td>
</tr>
<tr>
<td><code>pushNodeContext Method</code></td>
<td>Pushes local context down the stack, and creates a new local context that is defined by the DOM node.</td>
</tr>
<tr>
<td><code>reset Method</code></td>
<td>Clears the stack of contexts, emptying the current context.</td>
</tr>
</tbody>
</table>

The following XML Helper methods are for C++ only.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDeclaredPrefix Method (C++)</code></td>
<td>Returns the indexed prefix in the current context.</td>
</tr>
<tr>
<td><code>getPrefix Method (C++)</code></td>
<td>Finds the indexed prefix associated with the provided URI.</td>
</tr>
<tr>
<td><code>getURI Method (C++)</code></td>
<td>Returns the namespace URI for the provided prefix in a current context or DOM node context.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
**declarePrefix Method**

Binds the namespace prefix with the namespace URI in the local context. You can set the `allowOverride` property to control whether duplicate prefixes are allowed in the local context.

[Script]

**Script Syntax**

ObjNSManager.declarePrefix(prefix, namespaceURI);

**Parameters**

- **prefix**
  The namespace prefix.
- **namespaceURI**
  The namespace URI.

**Return Value**

None.

**Example**

```javascript
var oNSMgr;
try {
  oNSMgr = new ActiveXObject("Msxml2.MXNamespaceManager.5.0");
  oNSMgr.declarePrefix("sample", "urn");
  oNSMgr.declarePrefix("a", "abc");
  alert(oNSMgr.getDeclaredPrefixes.length); // 3
  alert(oNSMgr.getDeclaredPrefixes[0]); // "sample"
  alert(oNSMgr.getURI("sample")); // "urn"
  alert(oNSMgr.getPrefixes("urn")[0]); // "sample"
} catch(e) {
```

alert("Error \
" + e);
}

[Visual Basic]

**Visual Basic Syntax**

ObjNSManager.declarePrefix(prefix, namespaceURI)

**Parameters**

*prefix*  
The namespace prefix.

*namespaceURI*  
The namespace URI.

**Return Value**

None.

**Example**

```vbnet
Dim oNSMgr As New Msxml2.MXNamespaceManager50
oNSMgr.declarePrefix "sample", "urn "
oNSMgr.declarePrefix "a", "abc"
MsgBox oNSMgr.getDeclaredPrefixes.length ' 3
MsgBox oNSMgr.getDeclaredPrefixes.Item(0) ' "sample"
MsgBox oNSMgr.getPrefixes("urn:some uri").Item(0) ' "sample"
MsgBox oNSMgr.getURI("sample") ' "urn"
```

[C/C++]

```c
HRESULT declarePrefix(
    BSTR prefix,
    BSTR namespaceURI);
```

**Parameters**

*prefix*[in]  
The namespace prefix.

*namespaceURI*[in]  

The namespace URI.

C/C++ Return Values

S_OK
The value returned if successful.

S_FALSE
The value returned if the prefix declared after the last `pushContext` and `allowOverride` property is `VARIANT_TRUE` (prefix is overridden).

E_FAIL
The value returned if the prefix declared after the last `pushContext` and `allowOverride` property is `VARIANT_FALSE` (prefix is not overridden).

E_INVALIDARG
The value returned if one or more of the following is true: if the namespace URI is empty, but the but prefix is not; if the prefix is invalid "xml" or "xmlns"; if the prefix is not a non-colon XML name (NCName), as defined by the W3C Namespaces in XML Recommendation.

Example

```cpp
void SampleMXNameSpaceMgr_Prefixs()
{
    MSXML2::IMXNamespaceManagerPtr PtrIMXNamespaceManager;
    HRESULT hres;
    TCHAR Buffer[100];
    unsigned short WideCharBuffer[100];
    int BufferLength = 100;

    try
    {
        PtrIMXNamespaceManager.CreateInstance(__uuidof(MSXML2::MXNamespaceManager50));
        PtrIMXNamespaceManager->declarePrefix(_T("sample"),_T("urn"));
        PtrIMXNamespaceManager->declarePrefix(_T("a"),_T("abc"));

        _itot(PtrIMXNamespaceManager->getDeclaredPrefixes()->length, _
             Buffer, _
             BufferLength);
        DisplayMessageToUser(Buffer);

        BufferLength = 100;
        PtrIMXNamespaceManager->_getDeclaredPrefix(2, WideCharBuffer,
```
```c
BufferLength = 100;
PtrIMXNamespaceManager->_getPrefix(L"urn", 1, WideCharBuffer, &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));

BufferLength = 100;
hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL, _
WideCharBuffer, _
&BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));
}
catch(...)
{
    DisplayMessageToUser("Error");
}
}

void DisplayMessageToUser(char *Msg)
{
    ::MessageBox(NULL, Msg, _T("Message"), MB_OK);
}

Remarks

The `declarePrefix` method replaces the previous declaration if a prefix has
already been declared in the local context and `allowOverride` is enabled. The
NamespaceURI parameter should not be empty, unless it is used to redefine the
default namespace with a namespace that contains an empty prefix. For example,
the following is permitted:

```
nm.declarePrefix " "," 
```

But the following is not:

```
nm.declarePrefix "a"," 
```

The invalid prefixes "xml" and "xmlns" cannot be passed to this method.

An error is raised if the prefix is invalid, or if the namespace URI is empty, but
its prefix is not.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

allowOverride property

[Script, Visual Basic]

Applies to: IVBMXNamespaceManager

[C/C++]

Applies to: IMXNamespaceManager

Other Resources

W3C Namespaces in XML Recommendation
getDeclaredPrefixes Method

Returns a collection of declared namespace prefixes bound to a namespace name.

[Script]

Script Syntax

```javascript
var objNSPrefixes = objNSManager.getDeclaredPrefixes();
```

Parameters

None.

Return Values

An object. Returns the collection of prefixes.

Example

```javascript
var oNSMgr;

try
{
    oNSMgr = new ActiveXObject("Msxml2.MXNamespaceManager.5.0");
    oNSMgr.declarePrefix( "sample", "urn");
    oNSMgr.declarePrefix( "a", "abc");
    alert(oNSMgr.getDeclaredPrefixes.length); // 3
    alert(oNSMgr.getDeclaredPrefixes[0]); // "sample"
    alert(oNSMgr.getURI("sample")); // "urn"
    alert(oNSMgr.getPrefixes("urn")[0]); // "sample"
}
catch(e)
{
    alert("Error \n" + e);
}
```

Remarks
The returned collection is never empty. If no prefixes have been declared, the returned collection contains the prefix "xml" (permanently bound to "http://www.w3.org/XML/1998/namespace").

If the URI reference in a default namespace declaration is empty, unprefixed elements in the scope of the declaration are not considered to be in any namespace. This causes the prefix xmlns to be used only for namespace bindings. This prefix is not itself bound to any namespace name.

[Visual Basic]

Visual Basic Syntax

```
set objNSPrefixes = objNSManager.getDeclaredPrefixes
```

Parameters

None.

Return Values

An object. Returns the collection of prefixes.

Example

```
Dim oNSMgr As New Msxml2.MXNamespaceManager50
oNSMgr.declarePrefix "sample", "urn"
oNSMgr.declarePrefix "a", "abc"
MsgBox oNSMgr.getDeclaredPrefixes.length ' 3
MsgBox oNSMgr.getDeclaredPrefixes.Item(0) ' "sample"
MsgBox oNSMgr.getPrefixes("urn").Item(0) ' "sample"
MsgBox oNSMgr.getURI("sample") ' "urn"
```

Remarks

The returned collection is never empty. If no prefixes have been declared, the returned collection contains the prefix "xml" (permanently bound to "http://www.w3.org/XML/1998/namespace"). If the URI reference in a default namespace declaration is empty, unprefixed elements in the scope of the
declaration are not considered to be in any namespace. This causes the prefix xmlns to be used only for namespace bindings. This prefix is not itself bound to any namespace name.

[C/C++]

HRESULT getDeclaredPrefixes(
    IMXNamespacePrefixes** prefixes);

Parameters

prefixes[out,retval]
The prefixes collection.

C/C++ Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the last parameter is NULL.
E_OUTOFMEMORY
The value returned if there is no memory for IMXNamespacePrefixes.

Remarks

The returned collection is never empty. If no prefixes have been declared, the returned collection contains the prefix 'xml' (permanently bound to 'http://www.w3.org/XML/1998/namespace'). If the URI reference in a default namespace declaration is empty, unprefixed elements in the scope of the declaration are not considered to be in any namespace. This causes the prefix xmlns to be used only for namespace bindings. This prefix is not itself bound to any namespace name.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
getPrefixes Method | getDeclaredPrefix Method (C++)

Applies to: IVBMXNamespaceManager
getPrefixes Method

Returns a collection of namespace prefixes bound to the specified URI.

[Script]

Script Syntax

```javascript
var objNSPrefixCol = obnNSManager.getPrefixes(namespaceURI)
```

Parameters

namespaceURI
A string. The namespace URI bound to the prefix.

Return Values

An object. Returns the collection of prefixes.

Example

```javascript
var oNSMgr;
try
{
    oNSMgr = new ActiveXObject("Msxml2.MXNamespaceManager.5.0");
oNSMgr.declarePrefix("sample", "urn");
oNSMgr.declarePrefix("a", "abc");
alert(oNSMgr.getDeclaredPrefixes.length); // 3
alert(oNSMgr.getDeclaredPrefixes[0]); // "sample"
alert(oNSMgr.getURI("sample")); // "urn"
alert(oNSMgr.getPrefixes("urn")[0]); // "sample"
} catch(e) {
    alert("Error \n" + e);
}
```

[Visual Basic]
Visual Basic Syntax

set objNSPrefixCol = obnNSManager.getPrefixes(strNamespaceURI)

Parameters

strNamespaceURI
   The namespace URI.

Return Values

An object. Returns the collection of prefixes.

Example

Dim oNSMgr As New Msxml2.MXNamespaceManager50
oNSMgr.declarePrefix "sample", "urn"
oNSMgr.declarePrefix "a", "abc"
MsgBox oNSMgr.getDeclaredPrefixes.length ' 3
MsgBox oNSMgr.getDeclaredPrefixes.Item(0) ' "sample"
MsgBox oNSMgr.getPrefixes("urn").Item(0) ' "sample"
MsgBox oNSMgr.getURI("sample") ' "urn"

[C/C++]

C/C++ Syntax

HRESULT getPrefixes(
   BSTR namespaceURI,
   IMXNamespacePrefixes** prefixes);

Parameters

namespaceURI[in]
   The namespace URI.
prefixes[out, retval]
   The prefixes collection.

C/C++ Return Values
S_OK
   The value returned if successful.
E_POINTER
   The value returned if the last parameter is NULL
E_INVALIDARG
   The value returned if the namespaceURI argument is an empty string. A
   prefix cannot be bound to an empty string.
E_OUTOFMEMORY
   The value returned if there is no memory for IMXNamespacePrefixes.

Remarks

When there are no prefixes bound to the specified namespace URI, the returned
collection of prefixes contains no items.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button [ ] in the upper-left corner of the page.

See Also

getDeclaredPrefixes Method

Applies to: IVBMXNamespaceManager
getURI Method

Returns the namespace URI for the specified prefix.

[Script]

Script Syntax

var strURI = objNSManager.getURI(prefix);

Parameters

prefix

The prefix bound to the specified namespace URI.

Return Values

A string. The namespace URI for the provided prefix.

Example

    var oNSMgr;
    try
    {
        oNSMgr = new ActiveXObject("Msxml2.MXNamespaceManager.5.0");
        oNSMgr.declarePrefix("sample", "urn:same uri");
        alert(oNSMgr.getURI("sample")); // "urn:same uri"
        oNSMgr.popContext();
        oNSMgr.declarePrefix("sample", "urn:new uri");
        alert(oNSMgr.getURI("sample")); // "urn:new uri"
        oNSMgr.popContext();
        alert(oNSMgr.getURI("sample")); // "urn:same uri"
    } catch(e)
    {
        alert("Error \n" + e);
    }

[Visual Basic]
Visual Basic Syntax

strURI = objNSManager.getURI(prefix)

Parameters

prefix
The prefix bound to the specified namespace URI.

Return Values

A variant. The namespace URI for the provided prefix.

Example

Dim oNSEarly As New Msxml2.MXNamespaceManager50
oNSEarly.declarePrefix "sample", "urn:same uri"
MsgBox oNSEarly.getURI("sample") ' "urn:same uri"
oNSEarly.pushContext
oNSEarly.declarePrefix "sample", "urn:new uri"
MsgBox oNSEarly.getURI("sample") ' "urn:new uri"
oNSEarly.popContext

MsgBox oNSEarly.getURI("sample") ' "urn:same uri"
If strURI <> Null Then
    MsgBox strURI
End If

[C/C++]

C/C++ Syntax

HRESULT getURI(
    BSTR prefix,
    VARIANT* uri);

Parameters

prefix[in]
The prefix bound to the specified namespace URI.
uri[out,retval]
The namespace URI for the provided prefix.

**C/C++ Return Values**

S_OK
   The value returned if successful.

E_FAIL
   The value returned if the prefix is not bound to the current context.

E_POINTER
   The value returned if the last argument is NULL.

**Remarks**

An error is raised if a prefix is not declared, or is invalid. This method does not change the current context or stack.

The prefix "xml" is always considered declared, and is permanently bound to "http://www.w3.org/XML/1998/namespace".

The prefix "xmlns" is used only for namespace bindings, and is not itself bound to any namespace name. For prefix "xmlns", getURI() returns S_FALSE, because it is not bound to any namespace.

In the following table, the first column shows the sequence of namespace declarations in an XML document. The second and third columns show the requesting prefixes and return values of the getURI method.

<table>
<thead>
<tr>
<th>Declarations</th>
<th>Requesting prefixes</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of the file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getURI &quot;xml&quot;</td>
<td></td>
<td>S_OK, VT_BSTR(&quot;http://…&quot;)</td>
</tr>
<tr>
<td>getURI &quot;abc&quot;</td>
<td></td>
<td>S_FALSE, VT_NULL</td>
</tr>
<tr>
<td>getURI &quot;&quot;</td>
<td></td>
<td>S_FALSE, VT_NULL</td>
</tr>
<tr>
<td>getURI &quot;xmlns&quot;</td>
<td></td>
<td>S_FALSE, VT_NULL</td>
</tr>
<tr>
<td>getURI &quot;cdf&quot;</td>
<td></td>
<td>S_FALSE, VT_NULL</td>
</tr>
<tr>
<td>getDeclaredPrefixes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmlns=&quot;urn://1&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
xmlns:abc="urn://2"

getURI "xml" S_OK, VT_BSTR("http://…")
getURI "abc" S_OK, VT_BSTR("urn://2")
getURI "" S_OK, VT_BSTR("urn://1")
getURI "xmlns" S_FALSE, VT_NULL
getURI "cdf" S_FALSE, VT_NULL
getDeclaredPrefixes ("abc", "", "xml")

xmlns=""

getURI "xml" S_OK, VT_BSTR("http://…")
getURI "abc" S_OK, VT_BSTR("urn://2")
getURI "" S_FALSE, VT_NULL
getURI "xmlns" S_FALSE, VT_NULL
getURI "cdf" S_FALSE, VT_NULL
getDeclaredPrefixes ("abc", "xml")

Applies to: IVBMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
**getURIFromNode Method**

Returns the namespace URI for the provided prefix and DOM node.

**Script Syntax**

```javascript
var strURI = objNSManager.getURIFromNode(strPrefix, objIXMLDOMNode);
```

**Parameters**

- `strPrefix`  
  The namespace prefix.

- `objIXMLDOMNode`  
  The provided DOM node.

**Return Values**

A variant. The namespace URI.

**Visual Basic Syntax**

```visualbasic
strURI = objNSManager.getURIFromNode(strPrefix, objIXMLDOMNode)
```

**Parameters**

- `strPrefix`  
  The namespace prefix.

- `objIXMLDOMNode`  
  The provided DOM node.

**Return Values**
A variant. The namespace URI.

[C/C++]

C/C++ Syntax

HRESULT getURIFromNode(
    BSTR strPrefix,
    IXMLDOMNode* contextNode,
    VARIANT* uri);

Parameters

strPrefix[in]  
The namespace prefix.
contextNode[in]  
The provided DOM node.
uri[out,retval]  
The namespace URI.

C/C++ Return Values

S_OK  
The value returned if successful.
S_FALSE  
The value returned if the prefix is not bound to the current context.
E_INVALIDARG  
The value returned if the contextNode parameter is NULL.
E_POINTER  
The value returned if the passed object or pointer to the return string is NULL.

Remarks

Current context on the stack is not available for this method. The stack and current context on the stack are not affected.

To use the context of the same node several times, use a separate instance of the NamespaceManager object, or reset the current one. Resetting the current instance
of NamespaceManager will cause the loss of the current stack.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

getURI Method

Applies to: IVBMXNamespaceManager
popContext Method

Pops the last local context from the top of the stack.

[Script]

Script Syntax

objNSManager.popContext();

Parameters

None.

Example

```javascript
var oNSMgr;

try
{
    oNSMgr = new ActiveXObject("Msxml2.MXNamespaceManager.5.0");
oNSMgr.declarePrefix("sample", "urn:same uri");
alert(oNSMgr.getURI("sample")); // "urn:same uri"
oNSMgr.pushContext();
oNSMgr.declarePrefix("sample", "urn:new uri");
alert(oNSMgr.getURI("sample")); // "urn:new uri"
oNSMgr.popContext();
alert(oNSMgr.getURI("sample")); // "urn:same uri"
}
catch(e)
{
    alert("Error \n" + e);
}
```

[Visual Basic]

Visual Basic Syntax

objNSManager.popContext
Parameters

None.

Example

```vbnet
Dim oNSMgr As New Msxml2.MXNamespaceManager50
oNSMgr.declarePrefix "sample", "urn:same uri"
MsgBox oNSMgr.getURI("sample") ' "urn:same uri"
oNSMgr.pushContext
oNSMgr.declarePrefix "sample", "urn:new uri"
MsgBox oNSMgr.getURI("sample") ' "urn:new uri"
oNSMgr.popContext
    MsgBox oNSMgr.getURI("sample") ' "urn:same uri"
```

[C/C++]

C/C++ Syntax

```c
HRESULT popContext();
```

Parameters

None.

C/C++ Return Values

S_OK
    The value returned if successful.
E_FAIL
    The value returned if no context is available to pop.

Example

```c
MSXML2::IMXNamespaceManagerPtrPtrIMXNamespaceManager;
HRESULT hres;
unsigned short WideCharBuffer[100];
int BufferLength = 100;

try
{
```
PtrIMXNamespaceManager.CreateInstance(_uuidof(MSXML2::MXNamespaceManager));
PtrIMXNamespaceManager->declarePrefix(_T("sample"),_T("urn"));

hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL,
WideCharBuffer, &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));
PtrIMXNamespaceManager->pushContext();
PtrIMXNamespaceManager->declarePrefix(_T("sample"),_T("urn2"));

BufferLength = 100;

hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL,
WideCharBuffer, &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));
PtrIMXNamespaceManager->popContext();

BufferLength = 100;

hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL,
WideCharBuffer, &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));
}
catch(...) {
    DisplayMessageToUser("Error");
}

void DisplayMessageToUser(char *Msg) {
    ::MessageBox(NULL, Msg, _T("Message"), MB_OK);
}

Remarks

The popContext method removes all declarations from the top of the stack down to the declaration prior to the last pushContext method call. If no context is available to pop, popContext returns an error.

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also
[Script, Visual Basic]

Applies to: **IVBMXNamespaceManager**

[C/C++]

Applies to: **IMXNamespaceManager**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
pushContext Method

Pushes the local context down the stack.

[Script]

Script Syntax

objNSManager.pushContext();

Parameters

None.

Example

var oNSMgr;

try {
   oNSMgr = new ActiveXObject("Msxml2.MXNamespaceManager.5.0");
   oNSMgr.declarePrefix("sample", "urn:same uri");
   alert(oNSMgr.getURI("sample"));
   oNSMgr.pushContext();
   oNSMgr.declarePrefix("sample", "urn:new uri");
   alert(oNSMgr.getURI("sample"));
   oNSMgr.popContext();
   alert(oNSMgr.getURI("sample"));
} catch(e) {
   alert("Error \n" + e);
}

[Visual Basic]

Visual Basic Syntax

objNSManager.pushContext
**Parameters**

None.

**Example**

```vbnet
Dim oNSMgr As New Msxml2.MXNamespaceManager50
oNSMgr.declarePrefix "sample", "urn:sample uri"
MsgBox oNSMgr.getURI("sample")
oNSMgr.pushContext
oNSMgr.declarePrefix "sample", "urn:new uri"
MsgBox oNSMgr.getURI("sample")
oNSMgr.popContext

MsgBox oNSMgr.getURI("sample")
```

[C/C++]

**C/C++ Syntax**

```c
HRESULT pushContext();
```

**Parameters**

None.

**C/C++ Return Values**

S_OK

  The value returned if successful.

**Example**

```c
MSXML2::IMXNamespaceManagerPtr PtrIMXNamespaceManager;
HRESULT hres;
unsigned short WideCharBuffer[100];
int BufferLength = 100;

try
{
    PtrIMXNamespaceManager.CreateInstance( _
```
__uuidof(MSXML2::XMLNamespaceManager));
PtrIMXNamespaceManager->declarePrefix(_T("sample"),_T("urn"));

hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL, _
    WideCharBuffer, _
    &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));

PtrIMXNamespaceManager->pushContext();

PtrIMXNamespaceManager->declarePrefix(_T("sample"),_T("urn2"))
BufferLength = 100;
hres = PtrIMXNamespaceManager->_getURI(L"sample", _
    NULL, _
    WideCharBuffer, _
    &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));

PtrIMXNamespaceManager->popContext();

BufferLength = 100;
hres = PtrIMXNamespaceManager->_getURI(L"sample", _
    NULL, _
    WideCharBuffer, _
    &BufferLength);
DisplayMessageToUser(_bstr_t(WideCharBuffer));
}

} catch(...) {
    DisplayMessageToUser("Error");
}

void DisplayMessageToUser(char *Msg)
{
    ::MessageBox(NULL, Msg, _T("Message"), MB_OK);
}

Remarks

The pushContext method pushes the local context down the stack, and creates a
new local context.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.
See Also

[Script, Visual Basic]
Applies to: IVBMXNamespaceManager

[C/C++]
Applies to: IMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
pushNodeContext Method

Pushes local context down the stack, and creates a new local context that is defined by the DOM node.

[Script]

**Script Syntax**

```javascript
var ObjNSManager.pushNodeContext(oContextNode,[fDeep as Boolean=True]);
```

**Parameters**

- **oContextNode**
  An IXMLDOMNode object. A DOM node that specifies the context to use.
- **fDeep**
  Boolean. The default value is True. If True, pushes namespace declarations of parent nodes. If False, includes only namespace declarations in the provided node.

[Visual Basic]

**Visual Basic Syntax**

```vbscript
ObjNSManager.pushNodeContext(oIXMLDOMNode,[fDeep as Boolean=True]);
```

**Parameters**

- **oIXMLDOMNode**
  An IXMLDOMNode object. A DOM node that specifies the context to use.
- **fDeep**
  Boolean. The default value is True. If True, pushes namespace declarations of parent nodes. If False, includes only namespace declarations in the provided node.

[C/C++]
C/C++ Syntax

HRESULT pushNodeContext(
         IXMLDOMNode * oContextNode,
         VARIANT_BOOL fDeep);

Parameters

oContextNode[in]
   The DOM node with context use.

fDeep[in,optional]
   Boolean. The default value is True. If True, pushes namespace declarations
   of parent nodes. If False, includes only namespace declarations in the
   provided node.

C/C++ Return Values

S_OK
   The value returned if successful.

E_INVALIDARG
   The value returned if the passed object is NULL.

E_FAIL
   The value returned if an error occurs.

Remarks

The new local context is populated with namespace declarations from the
context of the DOM node. The second parameter indicates whether deep context
should be used. Deep context includes all namespace declarations from the node,
and all parents of the node. Pushed declarations are still visible in the current
context, unless they are replaced by new declarations.

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

[Script, Visual Basic]
Applies to: IVBMXNamespaceManager

[C/C++]

Applies to: IMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
reset Method

Clears the stack of contexts, emptying the current context.

[Script]

Script Syntax

objNSManager.reset();

Parameters

None.

[Visual Basic]

Visual Basic Syntax

objNSManager.reset

Parameters

None.

[C/C++]

C/C++ Syntax

HRESULT reset();

Parameters

None.

C/C++ Return Values
S_OK
The value returned if successful.

E_FAIL
The value returned if an error occurs.

Remarks

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

[Script, Visual Basic]
Applies to: IVBMXNamespaceManager

[C/C++]
Applies to: IMXNamespaceManager
getDeclaredPrefix Method (C++)

Returns the indexed prefix in the current context. Used for enumeration, this method requests all available prefixes in the current context. The set of prefixes begins at 0. The index value for a prefix can change as the current context changes.

C/C++ Syntax

HRESULT _getDeclaredPrefix
    [in] long nIndex,
    [in, out] wchar_t* pwchPrefix,
    [in, out] int* ppchPrefix);

Parameters

nIndex[in]
    A long index value, starting from 0.
pwchPrefix[in,out]
    The returned prefix value. The buffer length on input contains the maximum length of the buffer.
ppchPrefix[in,out]
    The length of the prefix. On output (for successful calls), the returned buffer contains the length of the prefix string. If the buffer is NULL, only the required buffer size is returned.

C/C++ Return Values

S_OK
    The value returned if successful.
E_FAIL
    The value returned if the index is out of bounds.
E_XML_BUFFERTOOSMALL
    The MSXML error code for insufficient return buffer length.
E_POINTER
    The value returned if ppchPrefix is NULL.
Example

MSXML2::IMXNamespaceManagerPtr PtrIMXNamespaceManager;
HRESULT hres;
TCHAR Buffer[100];
unsigned short WideCharBuffer[100];
int BufferLength = 100;

try
{
    PtrIMXNamespaceManager.CreateInstance(_uuidof(MSXML2::MXNamespaceManager));
    PtrIMXNamespaceManager->declarePrefix(_T("sample"), _T("urn"));
    PtrIMXNamespaceManager->declarePrefix(_T("a"), _T("abc"));

    _itot(PtrIMXNamespaceManager->getDeclaredPrefixes()->length, Buffer, BufferLength);
    DisplayMessageToUser(Buffer);

    BufferLength = 100;
    PtrIMXNamespaceManager->_getDeclaredPrefix(2, WideCharBuffer, BufferLength);
    DisplayMessageToUser(_bstr_t(WideCharBuffer));

    BufferLength = 100;
    PtrIMXNamespaceManager->_getPrefix(L"urn", 1, WideCharBuffer, BufferLength);
    DisplayMessageToUser(_bstr_t(WideCharBuffer));

    BufferLength = 100;
    hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL, _WideCharBuffer, BufferLength);
    DisplayMessageToUser(_bstr_t(WideCharBuffer));
}
catch(...)
{
    DisplayMessageToUser("Error");
}

void DisplayMessageToUser(char *Msg)
{
    ::MessageBox(NULL, Msg, _T("Message"), MB_OK);

Remarks
Returns a prefix value stored in a character buffer that is a null-terminated wide string accompanied by the `length` parameter. The length, `ppchPrefix`, parameter contains the size of the buffer on input and the length, not including the null character, of a result on output. If the pointer to the buffer is NULL, `ppchPrefix` contains the required buffer size.

**See Also**

Applies to: [IMXNamespaceManager](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
getPrefix Method (C++)

Finds the indexed prefix associated with the provided URI. Returns the indexed prefix in the current context. Used for enumeration, this method requests all available prefixes in the current context. The set of prefixes begins at 0. Note that the index value for a prefix can change as the current context changes.

C/C++ Syntax

```c
HRESULT _getPrefix
    [in] const wchar_t* pwszNamespaceURI,
    [in] long nIndex,
    [in, out] wchar_t* pwchPrefix,
    [in, out] int* pcchPrefix);
```

Parameters

*pwszNamespaceURI* [in]
- The namespace URI associated with the prefix.

*nIndex* [in]
- A long index value starting from 0.

*pwchPrefix* [in,out]
- The returned prefix value. The buffer length on input contains the maximum length of the buffer. On output (for successful calls), the returned buffer contains the length of the prefix string. If the buffer is NULL, only the length is returned.

*pcchPrefix* [in,out]
- The length of the prefix.

C/C++ Return Values

*S_OK*
- The value returned if successful.

*E_INVALIDARG*
- The value returned if the *pwszNamespaceURI* parameter is an empty string or NULL. No prefix can be bound to an empty string.
E_FAIL
   The value returned if the index is out of bounds.
E_XML_BUFFERTOOSMALL
   The MSXML error code for insufficient return buffer length.
E_POINTER
   The value returned if ppchPrefix is NULL.

Remarks

Return values stored in a character buffer are null-terminated wide strings that are accompanied by the pcchPrefix parameter. The pcchPrefix, parameter specifies the length of the buffer on input, and returns the length of the result on output. The length of the result is returned as a null-terminated string in characters. If the pointer to the buffer is NULL, the getPrefix method returns only the length.

Example

MSXML2::IMXNamespaceManagerPtr PtrIMXNamespaceManager;
HRESULT hres;
TCHAR Buffer[100];
unsigned short WideCharBuffer[100];
int BufferLength = 100;

try
{
   PtrIMXNamespaceManager.CreateInstance(_
      __uuidof(MSXML2::MXNamespaceManager));
   PtrIMXNamespaceManager->declarePrefix(_T("sample"), _T("urn"));
   PtrIMXNamespaceManager->declarePrefix(_T("a"), _T("abc"));
   _itot(PtrIMXNamespaceManager->getDeclaredPrefixes()->length, _
      Buffer, BufferLength);
   DisplayMessageToUser(Buffer);

   BufferLength = 100;
   PtrIMXNamespaceManager->_getDeclaredPrefix(2, WideCharBuffer, &BufferLength);
   DisplayMessageToUser(_bstr_t(WideCharBuffer));

   BufferLength = 100;
   PtrIMXNamespaceManager->_getPrefix(L"urn", 0, WideCharBuffer, &BufferLength);
   DisplayMessageToUser(_bstr_t(WideCharBuffer));
}
BufferLength = 100;
hres = PtrIMXNamespaceManager->_getURI(L"sample", NULL, _
WideCharBuffer, _
&BufferLength);

DisplayMessageToUser(_bstr_t(WideCharBuffer));
}
catch(...)
{
    DisplayMessageToUser("Error");
}

void DisplayMessageToUser(char *Msg)
{
    ::MessageBox(NULL, Msg, _T("Message"), MB_OK);

Remarks

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button ☐ in the upper-left corner of the page.

See Also

Applies to: IMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Reference
getURI Method (C++)

Returns the namespace URI for the provided prefix in a current context or DOM node context. If the DOM node is Null, it uses the current context.

C/C++ Syntax

HRESULT _getURI
    [in] const wchar_t* pwchPrefix,
    [in] IXMLDOMNode* pContextNode,
    [in, out] wchar_t* pwchUri,
    [in, out] int* pcchUri);

Parameters

pwchPrefix[in]
    The prefix associated with the namespace URI.

pContextNode[in]
    The provided DOM node.

pwchUri[in,out]
    The returned namespace URI value. The buffer length on input contains the maximum length of the buffer. On output (for successful calls), the returned buffer contains the length of the namespace URI string. If the buffer is NULL, only the length is returned.

pcchUri[in,out]
    The length of the namespace URI.

C/C++ Return Values

S_OK
    The value returned if successful.

E_INVALIDARG
    The pwchUri parameter is an empty string. No prefix can be bound to an empty string.

E_FAIL
    The value returned if the index is out of bounds.
E_XML_BUFFERTOOSMALL
   The MSXML error code for insufficient return buffer length.
E_POINTER
   The value returned if pcchURI is NULL.

Remarks

Return values stored in a character buffer are null-terminated wide strings that are accompanied by the pcchUri parameter. The pcchUri, parameter specifies the length of the buffer on input and returns the length of the result on output. The length of the result is returned as a null-terminated string in characters. If the pointer to the buffer is NULL, the getURI method returns only length.

See Also

Applies to: IMXNamespaceManager
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature

[This feature was first implemented for MSXML 5.0.]

Beginning with Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office, application developers can sign and verify XML data in the form of a DOM document, DOM nodes, or a SAX stream.

Working with XML Digital Signature (XMLDSig) is similar to working with DOM: You call methods and properties on the COM interfaces. This is because XMLDSig is built as a set of COM objects, following the same programming model as other features in MSXML.

The implementation of the XMLDSig feature complies with the "XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002" specification.

This documentation contains the following parts:

- XML Digital Signature Developer's Guide
- XML Digital Signature Reference

Applies To

Version: MSXML 5.0 and later.

Library: MSXML5.dll, MSXML5a.dll, MSXML5r.dll

Other Resources

XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Developer's Guide

[This feature was first implemented for MSXML 5.0.]

This guide contains the following topic.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter Kit</td>
<td>Provides introductory information and FAQ about XML Digital Signature.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
This starter kit is meant to provide you with a quick introduction to the XML Digital Signature (XMLDSig) feature in MSXML 5.0 and later. It briefly discusses some of the important underlying technologies, such as Public Key Infrastructure (PKI), encryption, and digital signatures.

This section contains the following topics.

- What Is XML Digital Signature?
- Getting Started with XML Digital Signatures

See Also

XML Digital Signatures Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
What is XML Digital Signature?

[This feature was first implemented for MSXML 5.0.]

In a paper document, a signature ensures the integrity of the document and the authenticity of the author or signer. This is because a signature is unique to the signer. In essence, signature serves to ensure data integrity, authenticity, and non-repudiation in the process of data exchange.

Similarly, digital signature ensures authenticity, integrity, and non-repudiation for an electronic document. Digital signature relies on the public key infrastructure (PKI), in which an electronic document is uniquely bound to the signer by means of the encryption and decryption of a thumbprint (digest or hash) of the document. This encryption and decryption is accomplished using a pair of public and private keys.

XML Digital Signature is an XML representation of digital signature applied over arbitrary documents. The syntax and semantics of XML digital signature follows the W3C standard as specified in "XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002".

This section contains the following topics.

- Public and Private Key Pairs
- Certificates and Certificate Authority
- Encryption and Decryption
- Digest or Hash
- Digital Signature
- XML Digital Signature
- CAPICOM and CryptoAPI
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Public and Private Key Pairs

[This feature was first implemented for MSXML 5.0.]

In the public key infrastructure (PKI), a subject (that is, an entity) whose identity is of significant value is assigned a pair of cryptographic keys. The public key is published to the general public. The private key is secret, known only to the assigned. The relationship between these two keys is such that data encrypted by one key can be decrypted only by the corresponding key. It is almost impossible to obtain the private key from the public key. Certificates are used to ensure the validity of public keys.

PKI can be used to maintain data confidentiality as well as data integrity, authenticity, and non-repudiation. The former amounts to encryption and decryption. The latter gives rise to digital signature.

MSXML uses CAPICOM or CryptoAPI for the underlying cryptographic operations, including the key management, that are required for XML digital signature. In CAPICOM or CryptoAPI, keys are opaque and can be accessed or passed around through key handles. In MSXML, keys are encapsulated in objects that implement the IXMLDsigKey interface or the IXMLDsigKeyEx interface. To use a key, you must first create the key object by calling one of the createKeyFrom* methods on a signature object that implements IXMLDigitalSignature or IXMLDigitalSignatureEx.

See Also

XML Digital Signature Methods
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Certificates and Certificate Authority

[This feature was first implemented for MSXML 5.0.]

A certificate is a proof of validity for a public key issued by a certificate authority (CA). In addition to the key itself, a certificate contains information about the subject the certificate is issued to, when the certificate is valid, and the organization that issued the certificate.

When a signature is verified using a public key, it might be a good idea to check the certificate of the key to ensure that the certificate has not expired or been revoked, and that the issuer is a trusted certificate authority.

In MSXML, you can call IXMLDSigKey::getVerifyingCertificate and IXMLDSigKeyEx::getVerifyingCertificateContext to retrieve the available certificate of the key used in signature verification. You can verify the validity of the certificate by using the appropriate methods or functions exposed by CAPICOM or CryptoAPI.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Encryption and Decryption

[This feature was first implemented for MSXML 5.0.]

Encryption translates data from an intelligible format to an unintelligible one. Decryption is the process in reverse. Confidentiality is achieved if encrypted data can be decrypted only by the intended reader.

There are two ways to encrypt data. Symmetric encryption uses a shared key that is used in both encryption and decryption. Confidentiality is maintained as long as the key is known only to the sender and the recipient. It is nearly impossible to keep the shared key secret when the key must be distributed frequently and widely through different means.

Asymmetric encryption uses public and private key pairs. Data becomes confidential when the sender encrypts it with the public key of the intended recipient and the receiver decrypts it with a private key. Key management is simple as long as the public key is certifiable by a reputable certificate authority (CA).

Encryption and decryption are used in digital signature as well. In this case, however, the role of public and private keys are reversed, and the transformations are usually applied against a digest of the data.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Digital Signature

[This feature was first implemented for MSXML 5.0.]

Digital signature is data digest encrypted with the private key of the signer. The private key ensures that the digital signature is unique to its bearer.

To sign data, the signer digests the data, encrypts it with a private key, and attaches the encrypted digest value to the data. To verify the signature, the verifier uses the signer's public key to decrypt the encrypted digest value attached to the data. The verifier then compares this decrypted digest value with the digest value computed on the companion data. It is important that both the signer and the verifier use the same hash function to digest the data.

Commonly used methods to produce digital signatures include RSA with SHA-1 and DSA with SHA-1 (also known as DSS).

For XML digital signature, MSXML supports RSA with SHA-1 and DSA with SHA-1, in addition to HMAC with SHA-1.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Digest or Hash

[This feature was first implemented for MSXML 5.0.]

The digest of data is the result of some fixed bit length of a one-way hash function that takes the data as input. A digest is sensitive to changes in the corresponding data. Therefore, it serves as a unique and in most cases compacted representation of the original data.

The strength of a hash algorithm is measured by the unlikelihood of collision. A collision is when two pieces of data produce the same digest. The strength increases with the bit length of the hash value. Digests are useful because computations involved in encrypting or decrypting a digest are usually much less demanding than encrypting or decrypting the data itself.

Commonly used hash functions include Secure Hash Algorithm 1 (SHA-1) whose output is of 160 bit in length.

In MSXML, supported methods for digest including SHA-1.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature

[This feature was first implemented for MSXML 5.0.]

XML digital signature is an XML representation of digital signature as defined in the W3C standard: "XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002". This means that XML digital signature is itself an XML document or an XML fragment.

MSXML uses Microsoft CAPICOM and CryptoAPI as the underlying technologies perform the cryptographic operations required for XML digital signature. MSXML encapsulates CAPICOM and CryptoAPI operations, such as key management and the encryption/decryption and computations of digests. MSXML also simplifies the signing or verification of XML data into a simple method call on a COM interface. Experienced CAPICOM/CryptoAPI users can further examine and manipulate other security-related information using methods or functions exposed in CAPICOM/CryptoAPI.

XML digital signature can be used to sign any data, including an XML document or fragment, an HTML page, plain text, or binary-encoded data such as a JPEG file.

Based on the relationship between the XML digital signature and the signed data, there are three types of XML digital signatures. The following topics describe these types.

- **Enveloping Signatures**
- **Enveloped Signatures**
- **Detached Signatures**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Enveloping Signatures

The `<ds:Signature>` element contains the signed or to-be-signed data as the content of a `<ds:Object>` element. For example, the following signature template is for an enveloping signature in which the plain text "Hello, World!" is embedded.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="obj">Hello, World!</ds:Object>
</ds:Signature>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Enveloped Signatures

In an enveloped signature, the signed or to-be-signed data is an XML document and contains the `<ds:Signature>` element as its child element.

**Note** The content of `<ds:Signature>` element is excluded from the calculations of the signature value (that is, the content of `<ds:SignatureValue>`) of an enveloped signature.

The following is an example of enveloped signatures.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Envelope [
  <!ENTITY ds "http://www.w3.org/2000/09/xmldsig#"/>
  <!ENTITY c14n "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <!ENTITY xpath "http://www.w3.org/TR/1999/REC-xpath-19991116"/>
  <!ENTITY xslt "http://www.w3.org/TR/1999/REC-xslt-19991116"/>
  <!ENTITY digest "http://www.w3.org/2000/09/xmldsig#sha1"/>
]>
<Letter>
  <Return-address>...</Return-address>
  <To>...</To>
  <Message>...</Message>
  <From>
    <ds:Signature xmlns:ds="&ds;">
      <ds:SignedInfo>
        ...
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform Algorithm="&xpath;">
            <XPath xmlns:dsig="&ds;">
              not(ancestor-or-self::dsig:Signature)
            </XPath>
          </ds:Transform>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="&digest;"/>
        <ds:DigestValue></ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
  </From>
  <Attach>...</Attach>
</Letter>
```
The data referenced in the <ds:Signature> element and specified by the "URI='"" attribute of the <ds:Reference> descendant element is the entire <Letter> element, including <ds:Signature> itself.
Detached Signatures

A detached signature is neither enveloping nor enveloped. The signature and data can be separate objects or files, or both can be contained in an XML document as sibling elements.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
CAPICOM and CryptoAPI

[This feature was first implemented for MSXML 5.0.]

CAPICOM and CryptoAPI are frameworks supported by Microsoft to enable applications to offer cryptographic and other security-related services. CryptoAPI is C-based. CAPICOM is COM-based and can be used in scripting. For more information, see Cryptography under the Security section in the MSDN Library.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Getting Started with XML Digital Signatures

Before you use IXMLDigitalSignature or IXMLDigitalSignatureEx interface to sign an XML document, you must have a digital certificate installed on your machine. To obtain a digital certificate, you must either create or have access to a certificate authority (CA). You also need to install CAPICOM 2.0 on your machine.

The following topics describe how perform the necessary steps to get started with XML Digital Signatures.

- Setting Up a Certificate Authority
- Requesting a Digital Certificate
- Issuing a Certificate for a Pending Request
- Installing a Digital Certificate
- Installing Platform SDK and Configuring Visual C++
- Downloading and Installing CAPICOM 2.0
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Setting Up a Certificate Authority

To request a digital certificate, you must either create a certificate authority (CA) or have access to one. For testing purposes, you might want to set up a private certificate authority to issue certificates for code signing. The following steps outline the procedure for doing this on a Windows 2000 Server or .NET Server machine.

To set up a certificate authority (CA)

1. Select a Windows 2000 Server or .Net Server machine to host the CA.
2. From the CA host, open Control Panel.
3. Double click Add/Remove Programs.
4. Click Add/Remove Windows Components.
5. Check Certificate Services and then click Next.
6. On the Certification Authority Types page of the wizard, select Stand-alone root CA. Also check the Advanced options box, and then click Next.
7. On the Public and Private Key Pair page, highlight "Microsoft Enhanced Cryptographic Provider v1.0". You might want to set "1024" as the value in the Key length drop-down box. Click Next.
8. On the CA Identifying Information page, fill out the blanks as appropriate. Click Next.
9. On the Data Storage Location page, use the default locations. Click Next.
10. Click Finish.

Note  CA requires IIS to be running. The Setup creates a "CertSrv" virtual directory under the default Web site under IIS. Requests for certificates should be addressed to this site via the URL, such as:
"http://theServer/CertSrv", where "theServer" is the URL of the Web server hosting the CA.

After you install Certificate Services, the computer cannot be renamed and cannot join or be removed from a domain.

If IIS is running on the server computer when you attempt to install Certificate Services, you will be prompted to stop IIS to complete the
installation.

If your Windows 2000 Server computer is running under a Service Pack update (such as SP1, SP2, or SP3), you should reapply the service packs after you install Certificate Services. You might also need to reinstall other services, such as IIS or Terminal Services.

After you have set up your CA, or if you choose to access an existing CA, you can request a digital certificate.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Requesting a Digital Certificate

Before you use IXMLDigitalSignature or IXMLDigitalSignatureEx interface to sign an XML document, you must have a digital certificate installed on your machine. To obtain a digital certificate, you must request one from a certificate authority (CA). You can either access a CA or set up your own. For instructions on setting up a private CA, see Setting Up a Certificate Authority.

The following steps describe how to request a certificate from a CA running Microsoft Certificate Services, supported on a Windows server machine.

To request a certificate from a certificate authority

1. Go to the CA by pointing your Web browser to the hosting web server—for example http://myCAServer/CertSrv, where myCAServer is the name of the Web server that also hosts the CA. "CertSrv" is the name of the virtual directory offering the certificate service.
2. Under Select a task, check Request a certificate. Click Next.
3. On the Choose Request Type page, select Advanced request. Click Next.
4. On the Advanced Certificate Requests page, select Submit a certificate request to this CA using a form. Click Next.
5. On the Advanced Certificate Request page, perform the following tasks:
   1. Fill in Identifying Information as appropriate.
   2. For Intended Purpose, select Code Signing Certificate.
   3. For the CSP entry under Key Options, select "Microsoft Enhanced Cryptographic Provider v1.0" for use with the PROV_RSA_FULL type.
   4. For Key Usage, select Signature.
   5. Select Create new key set, and check the Set the container name checkbox.
   6. Type a name, such as "MyRSAFullKeys", for the Container name field.

   Note All the sample files in the Digital Signatures Reference section use "MyRSAFullKeys" as the container name. Therefore, it is recommended that you use this name for testing purposes. If you choose a different name, you will need to supply it to some
key handling methods, such as `createKeyFromCSP`, for the samples to run.

7. Leave the remaining items with their default settings.
6. Click **Submit** to finish the request.

Now your request is pending. Depending on the CA you use, you might need to check the status of the request periodically. If you created your own CA, you will have to issue a certificate for the pending request.

After the certificate has been issued, you can install the certificate.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Issuing a Certificate for a Pending Request

If you set up your own certificate authority (CA), you will have to issue a certificate from the CA server computer to your local machine. The following procedure describes how to do this. If you chose to access an existing CA rather than create your own, an administrator on the CA computer must follow these steps.

To issue a certificate for a pending request

1. On the CA server computer, open Control Panel.
2. Double-click Administrative Tools, then Certification Authority.
3. In the console tree that appears, under Certification Authority (Local), locate the name of the CA that was used in the certificate request process. Expand its contents.
4. In the expanded console tree, click Pending Requests.
5. In the right-hand details pane, locate the pending certificate request that you've just submitted. Click it.
6. From the Action menu, point to All Tasks, and then click Issue.

Now the requested certificate has been issued, and you can install it for use in signing and verifying XML.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Installing a Digital Certificate

After you have requested and been issued a digital certificate, you must install it on your machine.

To install a certificate to a machine

1. Verify that the certificate has been issued by visiting the certificate authority (CA) server, such as "http://myCAServer/certsrv", to which you have previously submitted the request.

   Note  If your certificate request is still pending, you must wait until an administrator on the CA server machine issues the certificate before you attempt this procedure. See Issuing a Certificate for a Pending Request for more information.

2. Under Select a task, select Check on a pending certificate and click Next.
3. On the Check on a Pending Certificate Request page, select the appropriate certificate (such as "Code Signing Certificate"), and click Next.
4. Click the Install this certificate link.
5. When prompted, click Yes to confirm installation of the certificate.

Next, you need to install the latest version of the Platform SDK and configure Visual C++ to use it.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Installing the Platform SDK and Configuring Visual C++

Before using the XML Digital Signatures samples in this SDK, you need to do the following:

- Download and install a recent version of the Platform SDK.
- Verify that you have the latest versions of Wincrypt.h and Crypt32.lib provided by the Platform SDK.
- Configure Visual C++ project settings to use the Platform SDK subdirectories for include and library files.

The following procedures describe how to complete these setup tasks.

**To download and install the latest Platform SDK**

1. Go to Platform SDK Update on the Microsoft.com Download Center.
2. In the list of SDKs, select Core SDK.
3. Click Install this SDK.
4. Follow the instructions to install the Core SDK components.

   At a minimum, you need the build environment files (approximately 25 MB).

**To verify you have current versions of wincrypt.h and crypt32.lib**

1. In either Windows Explorer or at a command prompt, browse or change to the folder where you installed the Platform SDK.

   By default, the folder will be C:\Program Files\Microsoft SDK.

2. In the \Include subdirectory, verify that you have a copy of Wincrypt.h dated 6/21/2002 or later.
3. In the \Lib subdirectory, verify that you have a copy of Crypt32.lib dated 6/21/2002 or later.
To configure Visual C++ to use Platform SDK include and library files

1. Open or create a new project for each sample in Visual C++.
2. From the Tools menu, click Options....
3. In the Options dialog box select the Directories tab.
4. Add a new directory entry that points to the Platform SDK Include files folder, such as "C:\Program Files\Microsoft SDK\Include" if you installed the Platform SDK in the default path. To add a new directory entry, either click the New button or double click at the bottom of the list box, following the last entry in the list.
5. Select and drag the entry to the top of the list.
6. In Show directories for, select Library files.
7. Add a new directory entry that points to the Platform SDK Lib files folder, such as "C:\Program Files\Microsoft SDK\Lib" if you installed the Platform SDK in the default path.
8. Select and drag the entry to the top of the list.
9. Click OK.
Downloading and Installing CAPICOM 2.0

To use XML Digital Signatures, you must install CAPICOM 2.0.

To download and install CAPICOM 2.0

1. Go to Platform SDK Redistributable: CAPICOM 2.0 on the Microsoft.com Download Center.
2. Download the file by clicking the CC2RINST.EXE link.
3. When prompted, click Save and specify a folder on your local drive.
4. Run the self-extracting executable file. This will create a directory structure containing the redistributable, samples, readme text, and so on.
5. Navigate this directory to CAPICOM\x86\capicom.cab.
6. Right-click the capicom.dll file. Select Extract.
7. In the Select a Destination box, navigate to your system32 directory and click Extract.
8. Open a command prompt and type the following command:
   
   regsvr32.exe capicom.dll

   A message box should appear with the following message:

   DllRegisterServer in capicom.dll succeeded.

Note   If you first install the latest version of the Platform SDK and configure Visual C++ to use it, CAPICOM 2.0 might already exist in updated form on your system. To verify, check the date and version information on capicom.dll in your Windows system directory (i.e. C:\WINDOWS\system32).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Reference

[This feature was first implemented for MSXML 5.0.]

MSXML Digital Signature supports a COM-based API, which is documented in the following topics:

- XML Digital Signature Interfaces
- XML Digital Signature Properties
- XML Digital Signature Methods
- XML Digital Signature Enumerated Types
- XML Digital Signature Elements
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Interfaces

[This feature was first implemented for MSXML 5.0.]

XML Digital Signature supports the following COM interfaces to work with digital signatures and cryptographic keys.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXMLDigitalSignature</td>
<td>Dispatchable interface that handles signing and verification, as well as key manipulations.</td>
</tr>
<tr>
<td>IXMLDigitalSignatureEx</td>
<td>Non-dispatchable interface that handles signing and verification, as well as key manipulations.</td>
</tr>
<tr>
<td>IXMLDSigKey</td>
<td>Dispatchable interface that provides information about the key used for signing or verification.</td>
</tr>
<tr>
<td>IXMLDSigKeyEx</td>
<td>Non-dispatchable interface that provides information about the key used for signing or verification.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
IXMLDigitalSignature

[This feature was first implemented for MSXML 5.0.]

The IXMLDigitalSignature interface supports the following properties and methods to allow application developers to sign XML documents or verify signatures.

**Properties**

- **signature**
  
  This read/write property allows you to get or set a signature template or object to be processed by the IXMLDigitalSignature object.

- **store**
  
  This read/write property allows you to get or set a certificate store object.

**Methods**

- **createKeyFromCSP**
  
  Retrieves a key handle from a key container within a specified cryptographic service provider (CSP). This key handle is used to access the private key used for signing or the public key for signature verification.

- **createKeyFromHMACSecret**
  
  Extracts and returns a key handle from a base-64 encoded secret value used in the HMAC algorithm. This key handle is used to access a key that can be used for both signing and signature verification.

- **createKeyFromNode**
  
  Creates and returns a key handle based on the information contained in the `<ds:KeyInfo>` element and its descendant elements (`<ds:KeyInfo>` is
a child of a `<ds:Signature>` element). The key handle is used to access the key that is used for signature verification.

Creates and returns a SAX proxy object. The `setReferenceData` method uses this object to assign a SAX stream as the new data source of the signature object or template.

Assigns a new data source to a specified `<ds:Reference>` element in a signature object.

Signs data referenced in a `<ds:Signature>` element.

Verifies the signature of data referenced in the `<ds:Signature>` element.

### Events

None.

### Remarks

Using the `IXMDDigitalSignature` interface typically involves the following tasks.

1. Create an XML DOM object, and load it with a digital signature as represented by a `<ds:Signature>` element belonging to the `http://www.w3.org/2000/09/xmldsig#` namespace.
2. Create an `IXMDDigitalSignature` object, and assign the XML DOM object to its signature property.
3. Optionally call `setReferenceData`, and possibly `createSAXProxy`, to set or replace the data source referenced in the signature object.
4. Create a cryptographic key using one of the key creation methods. See [XML Digital Signature Methods](#) for more information.
5. Call `sign` or `verify` on the `IXMDDigitalSignature` object to process the signature on the XML document or fragment.
6. Optionally call `setReferenceData` to assign a different data source on the signature object, and repeat Step 5 to sign or verify the new data with the existing signature information.

The implementation of the `IXMLDigitalSignature` interface conforms to the [XML-Signature Syntax and Processing W3C Recommendation 12 February 2002](http://www.w3.org/TR/2002/REC-xmlsig-core-20020212).

**Requirements**

**Implementation:** `msxml5.dll`

[C/C++]

**Header and IDL files:** `msxml2.h, msxml2.idl`

**Version-Dependent ProgID:** `Msxml2.MXDigitalSignature.5.0`

**Version-Dependent CLSID:** `88d969e5-f192-11d4-a65f-0040963251e5, "MXDigitalSignature50"`

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**

[IXMLDSigKey](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
The IXMLDigitalSignatureEx interface extends the IXMLDigitalSignature interface to enable C/C++ applications to interact with various cryptographic objects, such as a certificate context or a certificate store handle, that are required to sign or verify a signature.

Properties

None

Methods

createKeyFromCertContext
Creates and returns a key handle from a certificate context as defined in CryptoAPI. Both the private and public keys are accessible through the resultant key handle.

createKeyFromHMACSecretBinary
Creates and returns a key handle from an unencoded binary secret value used in an HMAC algorithm. The key accessible through this handle may be used for signing and verifying an XML document or fragment.

getStoreHandle
Gets a handle to the certificate store from which the certificate can be retrieved and validated.

setDebugOutput
Sets the target for debug output.

setStoreHandle
Sets a certificate store handle on the signature object, so that certificate information can be inserted into the <ds:KeyInfo> element.
Events

None.

Remarks

The methods of this interface can only be called from C/C++ applications.

Requirements

Implementation: msxml5.dll

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Version-Dependent ProgID: Msxml2.MXDigitalSignature.5.0

Version-Dependent CLSID: 88d969e5-f192-11d4-a65f-0040963251e5, "MXDigitalSignature 5.0"

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDSigKey
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
IXMLDSigKey

[This feature was first implemented for MSXML 5.0.]

The IXMLDSigKey interface supports a method to enable an XML application to access the certificate used in signature verification. The application can then call the appropriate CAPICOM objects or methods to access entries in the certificate. This is useful for checking the validity of a certificate.

Properties

None.

Methods

getVerifyingCertificate

Retrieves the certificate used in signature verification.

Events

None.

Remarks

This is a dispatchable interface, and its method can be called from applications written in C/C++, Visual Basic, and many scripting languages.

Requirements

Implementation: msxml5.dll

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl
Version-Dependent ProgID: Msxml2.DOMDocument.5.0, Msxml2.FreeThreadedDOMDocument.5.0

Version-Dependent CLSID: 88d969e5-f192-11d4-a6f-0040963251e5

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☐ in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
The IXMLDSigKeyEx interface enables an XML application to access cryptographic information used in signature signing or verification. In addition to obtaining the certificate used in signature verification, an application can also retrieve the certificate context and the handle to the cryptographic service provider (CSP) created by the underlying process. Other cryptographic information can be obtained using the resultant certificate, certificate context, or CSP provider by calling the appropriate CryptoAPI functions.

**Properties**

None.

**Methods**

- **getCSPHandle**
  
  Retrieves the handle to the cryptographic service provider resulted from calling the IXMLDigitalSignature::createKeyFromCSP method.

- **getVerifyingCertificateContext**
  
  Retrieves the certificate context associated with this key.

**Events**

None.

**Remarks**

This is a non-dispatchable interface and can only be called in C/C++ applications.

**Requirements**
Implementation: msxml5.dll

[C/C++]

Header and IDL files: msxml2.h, msxml2.idl

Version-Dependent ProgID: Msxml2.DOMDocument.5.0,
Msxml2.FreeThreadedDOMDocument.5.0

Version-Dependent CLSID: 88d969e5-f192-11d4-a65f-0040963251e5

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Properties

[This feature was first implemented for MSXML 5.0.]

This section describes the following properties for XML digital signatures.

- **signature**: Allows you to get or set a `<ds:Signature>` element of an XML document. Read/write.
- **store**: Exposes the certificate store to an XML application. Read/write.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
signature Property

[This feature was first implemented for MSXML 5.0.]

This is a read/write property for getting or setting a signature to be processed.

The signature property takes an IXMLDOMNode object as its value. The DOM object holds a signature template or signed signature document as represented by a <ds:Signature> element that belongs to the http://www.w3.org/2000/09/xmldsig# namespace. The ds alias is used for this namespace throughout this document.

[JScript]

JScript Syntax

```javascript
var objSigNode = objXMLDigitalSignature.signature;
objXMLDigitalSignature.signature = objSigNode;
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objSigNode = objXMLDigitalSignature.signature
Set objXMLDigitalSignature.signature = objSigNode
```

[C/C++]

C/C++ Syntax Using Smart Pointers

```c
IXMLDOMNodePtr objSigNode = objXMLDigitalSignature.signature;
objXMLDigitalSignature.signature = objSigNode;
```

C/C++ Syntax

```c
HRESULT get_signature (IXMLDOMNode** objSigNode);
HRESULT putref_signature (IXMLDOMNode* objSigNode);
```
Parameters

$objSigNode

An XML DOM node corresponding to the <ds:Signature> element of the XML document.

Return Values

S_OK

Getting or setting signature succeeded.

E_FAIL

Getting or setting signature failed.

Remarks

Setting this property resets all the data previously assigned using the setReferenceData method, but does not have any side effects on cryptographic keys already created by the selected key creation method or by the createSAXProxy method.

After this property is set, you must not modify the XML data referenced to or embedded in the corresponding <ds:Signature> element until the data has been signed or verified. Otherwise, the outcome is not defined.

For both signing and verification, the signature property must meet certain requirements. The following topics provide tables that describe these requirements.

- Signature Template Requirements for Signing
- Signature Object Requirements for Verification

Example

This example illustrates how to get and set the signature property. The sample code performs the following tasks.

1. Loads an XML signature template file, signature_template.xml, to a DOM object.
2. Assigns the `<ds:signature>` element selected from the loaded DOM object to the signature property of an MXDigitalSignature50 object.

3. Signs the data referenced in the signature template. This data is the text "Hello, World!" contained in a `<ds:Object Id="objData">` element.

The example uses a resource file, `signature_template.xml`. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same from each language.

- **Resource Files**
  - JScript Code (signature.js)
  - Visual Basic Code (signature.frm)
  - C/C++ Code (signature.cpp)
  - Output for the signature Example

**Applies To**

IXMLDigitalSignature

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☐ in the upper-left corner of the page.

**See Also**

IXMLDigitalSignatureEx | sign Method | verify Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
# Signature Template Requirements for Signing

For signing, the `signature` property must be set with a signature template that meets the following requirements.

<table>
<thead>
<tr>
<th>Required elements of signature template for signing</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ds:Signature&gt;</code></td>
<td>Must be present and signify the starting point for the signature processing</td>
</tr>
</tbody>
</table>
| `<ds:SignedInfo>`                                  | Must contain the following child elements: `<ds:CanonicalizationMethod>`, `<ds:SignatureMethod>`, and `<ds:Reference>`.
| `<ds:CanonicalizationMethod>`                      | Must contain `Algorithm` attribute with a valid value. |
| `<ds:SignatureMethod>`                             | Must be present, and may contain an `Algorithm` attribute. Upon signing, an `Algorithm` attribute will be created, if not already present. This attribute value is replaced or set with the signing algorithm determined by the key used in signing. |
| `<ds:Reference>`                                   | At least one `<ds:Reference>` element must be present. This element may contain a `URI` attribute indicating the data source for the document to be signed and the value of this attribute must be resolved to XML data. When the `URI` attribute is absent, an application can set the data source by calling the `setReferenceData` method. There can be at most one |

<ds:Reference> element without the URI attribute.
This is an optional element. If present, it must contain at least one
<ds:Transform> child element.
Multiple children can be in any order.
<ds:Transform>
Must contain an Algorithm attribute with a known algorithm name as its value.
<xslt:stylesheet>
When the transform algorithm is XSLT, this element must be present and must contain a valid XSLT template.
When the transform algorithm is XPath, this element must be present and must contain the filtering predicate text.
<ds:XPath>
Must be present, and must contain the Algorithm attribute with the SHA1 algorithm name.
<ds:DigestMethod>
Must be present, but may contain a text value. For signing, this value will be replaced with the computed digest value.
<ds:DigestValue>
Must be present, but may contain a text value. For signing, this value will be replaced with the computed signature value.
<ds:SignatureValue>
Must be present if data is to be signed with fWriteKeyInfo set to KEYVALUE or CERTIFICATES. These enumerated constants are defined in XMLDSIG_WRITEKEYINFO enum.
In such cases, any existing content of this element will be replaced with appropriate key information.
May be present and contain an Id attribute whose value can be referenced
<ds:KeyInfo>
<ds:Object>
<ds:Manifest>

by the URI attribute of a <ds:Reference> element.
May be present, and may contain an Id attribute whose value can be referenced by the URI attribute of a <ds:Reference> element.

An example of a simple signature template, signature_template.xml, is listed as the Resource File for the signature example.

Return to signature property reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Signature Document Requirements for Verification

For verification, the signature property must be set with a valid signature object meeting the following requirements:

<table>
<thead>
<tr>
<th>Required elements of signature object for verification</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">ds:Signature</a></td>
<td>Must be present, and must signify the starting point for the signature processing</td>
</tr>
<tr>
<td><a href="">ds:SignedInfo</a></td>
<td>Must be present, and must contain the following child elements:</td>
</tr>
<tr>
<td><a href="">ds:CanonicalizationMethod</a></td>
<td><a href="">ds:CanonicalizationMethod</a>, <a href="">ds:SignatureMethod</a>, and <a href="">ds:Reference</a></td>
</tr>
<tr>
<td><a href="">ds:SignatureMethod</a></td>
<td>Must contain an Algorithm attribute with a valid value.</td>
</tr>
<tr>
<td><a href="">ds:Reference</a></td>
<td>Must be present, and must contain an Algorithm attribute with a valid signing algorithm name. At least one such element must be present. This element may contain a URI attribute indicating the data source for the document to be verified. The value of this attribute must be resolved to a data source. When the URI attribute is absent, an application must set the data source by calling the setReferenceData method. There can be at most one <a href="">ds:Reference</a> element without the URI attribute. This is an optional element. If present, it must contain at least one</td>
</tr>
</tbody>
</table>
<ds:Transform> child element. Multiple children can be in any order. Must contain an Algorithm attribute with a known algorithm name as its value.

When the transform algorithm is XSLT, this element must be present and must contain a valid XSLT template.

When the transform algorithm is XPath, this element must be present and must contain the filtering predicate text.

Must be present, and must contain an Algorithm attribute with the SHA1 algorithm name.

Must be present, and must contain a valid digest value. For verification, this value will be compared with the computed digest value.

Must be present, and must contain a valid signature value. For verification, this value will be compared with the computed signature value.

Must be present, and must contain valid key information if data is to be verified with user-supplied key information. When present, the <ds:KeyValue> child element will be used for verification. Otherwise, a <ds:X509Data> element containing a certificate will be searched.

May be present, and may contain an Id attribute whose value can be referenced by the URI attribute of a <ds:Reference> element.

May be present, and may contain a Id attribute whose value can be referenced
by the URI attribute of a <ds:Reference> element.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Resource: signature_template.xml

Use this resource file for the signature example.

This simple signature template references embedded text data held in the <ds:Object Id="#objData"> element. It also specifies how the referenced data is to be signed. Notice that the required <ds:SignatureValue> and <ds:DigestValue> elements are empty.

Signature Template (signature_template.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#objData">
            <ds:DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue/>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue/>
    <ds:Object Id="objData">Hello, World!</ds:Object>
</ds:Signature>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig, dsigKey
var szResult = ""
KEYVALUE = 1;
DSIGNSS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"
PROV_RSA_FULL = 1;
// Define constant for Diffie-Hillman or other cryptographic
// provider if not using RSA full provider.
// PROV_DSS_DH = 13;

if (loadSignature("signature_template.xml")) {
  // Change this key container name to your own if necessary.
  if (signWithKey(PROV_RSA_FULL, "MyRSAFullKeys")) {
    alert(szResult);
  }
}

///////// Helper functions: //////////
function alert(str) { WScript.echo(str); }

///////// Set signature for signing. //////////
function loadSignature(file)
{
  try {
    xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmldsig = new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
  }
  catch (e) {
    alert("Installation of mxsml5 is required to run this app.\n");
    return false;
  }
  xmldoc.async = false;
  xmldoc.preserveWhiteSpace = true;
  xmldoc.validateOnParse = false;
  xmldoc.resolveExternals = false;

  if (!xmldoc.load(file)) {
    alert("Can't load " + file + "\n");
    return false;
  }
  szResult += "\nInput signature template:\n\n" + xmldoc.xml;
  xmldoc.setProperty("SelectionNamespaces", DSIGNSS);
Set the signature property value to the first
<ds:Signature> element.
xmldsig.signature = xmldoc.selectSingleNode("./ds:Signature");

return true;
}

function signWithKey(dwCspType, szKeyContainer)
{
// Read the signature property to verify that it's been set.
if (!xmldsig.signature) {
  alert("Must set signature template before signing.\n");
  return false;
}
var oKey = xmldsig.createKeyFromCSP(dwCspType, "", szKeyContainer
var oSignedKey = xmldsig.sign(oKey, KEYVALUE);
if (oSignedKey == null) {
  alert("Signing failed.\n");
}
else {
  szResult += ("\nThe data referenced in the signature template
  "was signed successfully.\n" +
  "Resultant signature:\n\n" +
  xmldoc.xml");
}
return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template, and paste it into a text file. Save the file as signature_template.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as signature.js, in the same directory where you saved signature_template.xml.
4. From a command prompt, navigate to this directory, then type "cscript signature.js".

Note  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe or cscript.exe), if it is not already installed.
5. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim DispStr, file, provType, keyContainer
Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
Const PROV_RSA_FULL = 1

Private Function LoadXML()
    ' Read the input XML file, and display the content in
    ' the Text1 text box.
    DispStr = ""
    If xmldoc.xml = "" Then
        If file = "" Then
            DispStr = "invalid input xml file name"
            LoadXML = False
            Exit Function
        End If
    End If
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.resolveExternals = False
    If xmldoc.Load(Path) = False Then
        DispStr = DispStr + vbCrLf + 
                "can't load " + Path + vbCrLf + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    DispStr = "Input signature template:" + vbCrLf + vbCrLf
             + xmldoc.xml + vbCrLf
    LoadXML = True
End Function

Private Function SignXML()
    If xmldoc.xml = "" Then
        DispStr = "signature template is empty."
        SignXML = False
        Exit Function
    End If
    If keyContainer = "" Then
        DispStr = DispStr + "invalid keyContainer."
        SignXML = False
    End If
    Exit Function
End If

xmldoc.async = False
xmldoc.preserveWhiteSpace = True
xmldoc.validateOnParse = False
xmldoc.setProperty "SelectionNamespaces", DSIGNS

Set signature = xmldoc.selectSingleNode("./ds:Signature")
Set xmlDsSig.signature = signature

Set oKey = xmlDsSig.createKeyFromCSP(provType, ",", keyContainer,
Set oSignedKey = xmlDsSig.sign(oKey, 1)

If oSignedKey Is Nothing Then
    DispStr = "sign failed." + vbNewLine
End If

DispStr = DispStr + "The data referenced in the signature templa 
+ "was signed successfully." + vbNewLine 
+ "Resultant signature:" + vbNewLine + vbNewLine

DispStr = DispStr + xmldoc.xml
SignXML = True
End Function

Private Sub Form_Load()
    'Set text box to use form to determine its width 
    'and height when form is loaded
    Text1.Left = 100
    Text1.Top = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750

    provType = PROV_RSA_FULL
    file = "signature_template.xml"
    ' Change this key container name to your own if necessary.
    keyContainer = "MyRSAFullKeys"

    If LoadXML = False Then
        MsgBox DispStr
        Exit Sub
    End If

    If SignXML = False Then
        MsgBox DispStr
        Exit Sub
    End If
    Text1.Text = DispStr
End Sub
Private Sub Form_Resize()

    'Set text box to use form in determining its width and height
    'when form is resized
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750

End Sub

Try It!

1. Ensure that you have completed all the procedures in [Getting Started with XML Digital Signatures](#).
2. Copy the [XML signature template](#), and paste it into a text file. Save the file as `signature_template.xml`.
3. Create a Standard EXE project in Visual Basic. Save the empty project as `signature.vbp` to the same directory where you saved `signature_template.xml`. Name the form file `signature.frm`.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v5.0.
5. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named Text1. In the properties for Text1, set the ScrollBars property to use "2 - Vertical" as the value and set the MultiLine property to True.
6. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor to replace whatever code is already there.
7. Execute the code by selecting Start from the Run menu.
8. Verify that your output is similar to that listed in the Output topic. To view the contents of the output more easily, resize the application form as needed.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: signature.cpp

#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;
#define PROV_RSA_FULL 1

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL loadSignature(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    printf("\nInput signature template:\n\n%s\n", (LPCSTR)xmldoc->xml);
    xmldoc->setProperty("SelectionNamespaces", "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode(".//ds:Signature");
    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }
    return VARIANT_TRUE;
}

VARIANT_BOOL signWithKey(long CspType, _bstr_t keyContainer)
{
    if (!objectsAreInitialized) {
        printf("Must initialize object for signing.\n");
        return VARIANT_FALSE;
    }
// Get the value of the signature property and see if it has been set.
if (xmldsig->signature == NULL) {
    printf("Must set signature template before signing.\n");
    return VARIANT_FALSE;
}

IXMLDSigKeyPtr dsigKey = xmldsig->createKeyFromCSP(CspType, "", k
if (dsigKey == NULL) {
    printf("Failed to create key from CSP\n");
    return VARIANT_FALSE;
}

IXMLDSigKeyPtr signedKey = xmldsig->sign(dsigKey, KEYVALUE);
if (signedKey != NULL) {
    printf("\nThe data referenced in the signature template ");
    printf("was signed successfully.\nResultant signature:\n\n");
    printf("%s\n", (LPCSTR)xmldoc->xml);
} else {
    printf("Signing failed.\n");
    if (dsigKey) dsigKey.Release();
    return VARIANT_FALSE;
}

if (dsigKey) dsigKey.Release();
if (signedKey) signedKey.Release();

return VARIANT_TRUE;

VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.\n");
        return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50))) ) {
        printf("Installation of msxml5 is required to run this app.\n");
        return VARIANT_FALSE;
    }

    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}
void cleanObjects()
{
    if (xml_doc) xml_doc.Release();
    if (xml_dsig) xml_dsig.Release();
}

void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }
    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }
    if (VARIANT_TRUE == loadSignature("signature_template.xml")) {
        // Change this key container name to your own if necessary.
        if (VARIANT_TRUE != signWithKey(PROV_RSA_FULL, "MyRSAFullKeys")
            printf("exit with failure.\n");
    }
    cleanObjects();
    CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "signatureProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight signatureProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then
type "signature_template.xml" in the **File name** text box.

7. Click **OK**.

8. Copy the **XML signature template**, and paste it into the text file you just created.

9. Repeat steps 5-8 for the C++ file above (**signature.cpp**).

10. From the **Project** menu, click **Settings...**, then click the **Link** tab.

11. In **Object/library modules**, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click **OK**.

12. Build the sample by selecting **Build signatureProj.exe** from the **Build** menu.

13. Execute the sample application by selecting **!Execute signatureProj.exe** from the **Build** menu.

14. Verify that your output is similar to that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
When you build and run the signature example, you should get output similar to the following. Some of your keyInfo values will be different if you used a different key for signing the document.

Input signature template:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#objData">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#">
        <ds:DigestValue/+cFiIGIjF2PrUbAra+vbkOD4ICs=</ds:DigestValue>
      </ds:DigestMethod>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="objData">Hello, World!</ds:Object>
</ds:Signature>
```

The data referenced in the signature template was signed successfully.

Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#objData">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#">
        <ds:DigestValue>AwP8vztsLoCfXPCp0UfAavtFCxKok+3a16dU81UCPB5dbzZWupT+0tb8YPmkBoXRS5LgyJM0BFnak12pq+8PfyxSxz5/UVUK8n6X++4QAAXfStkSS+gkSCQuslad6swu9QmohaVaWAmKX/zOsfyvfXormrI=</ds:DigestValue>
      </ds:DigestMethod>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>AwnP8vztsLoCfXPCp0UfAavtFCxKok+3a16dU81UCPB5dbzZWupT+0tb8YPmkBoXRS5LgyJM0BFnak12pq+8PfyxSxz5/UVUK8n6X++4QAAXfStkSS+gkSCQuslad6swu9QmohaVaWAmKX/zOsfyvfXormrI=
  </ds:SignatureValue>
  <ds:Object Id="objData">Hello, World!</ds:Object>
</ds:Signature>
```
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KeyValue>
        <RSAKeyValue>
            <Modulus>
                pjLYA6wZ0Sw83nZiCnYmWRhuerStQ60kvL21qzpKcwZFvHCx21qucXwHiLLaHBY0KAhu9Fd6Fo1S+I+w/pQ6ZNfMmx7rh8yNgER9rcJXzPQeH1xHDz3Hm/Qiko8WkPajD9SNh79QTb/usLhjjE=
            </Modulus>
            <Exponent>AQAB</Exponent>
        </RSAKeyValue>
    </KeyValue>
</KeyInfo>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
store Property

[This feature was first implemented for MSXML 5.0.]

This read/write property makes a certificate store available for use in an XML application. The certificate store is a CAPICOM 2.0 object of the Store class.

JScript Syntax

```javascript
var objCertStore = objXMLDigitalSignature.store;
objXMLDigitalSignature.store = objCertStore;
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objCertStore = objXMLDigitalSignature.store
Set objXMLDigitalSignature.store = objCertStore
```

[C/C++]

C/C++ Syntax Using Smart Pointers

```c
IUnknownPtr objCertStore = objXMLDigitalSignature.store;
objXMLDigitalSignature.store = objCertStore;
```

C/C++ Syntax

```c
HRESULT get_store (IUnknown** objCertStore);
HRESULT putref_store (IUnknown* objCertStore);
```

Parameters

$objCertStore$

A certificate store object created as a CAPICOM 2.0 object of the Store class.
Return Values

S_OK
Getting or setting the store object succeeded.

E_FAIL
Getting or setting the store object failed. The \texttt{objCertStore} parameter returns NULL.

Remarks

You can only set the store property if the \texttt{CERTIFICATES} flag has been passed via the \texttt{fWriteKeyInfo} parameter during the \texttt{sign} method call. If this flag has been passed, and the \texttt{signature} property has been set, an application can then set the \texttt{store} property to specify the certificate store to use. From this store, certificates are retrieved and written to the \texttt{<ds:KeyInfo>} element in the signed signature.

You might want your application to verify the authenticity of the certificate used to sign or verify. To do this, you can use the \texttt{store} property to obtain the certificate store, enumerate the certificates there, and build a certificate chain to see if the certificate used can be trusted.

To create the CAPICOM 2.0 \texttt{Store} object, use the following ProgID:

"CAPICOM 2.0 Store"

For more information, see the following example.

Example

This example illustrates the effect of setting the \texttt{store} property on an \texttt{IXMLDigitalSignature} object. When a signature is signed without setting this property and the \texttt{CERTIFICATES} flag is set in the call to the \texttt{sign} method, the resultant signature document contains the certificate of the key used in \texttt{sign}. However, when the signature is signed after this property is set to the certificate store, the resultant signature document contains all the certificates in the store, in addition to the certificate of the key used in \texttt{sign}.

In this example, the certificate store has three certificates. So the signed signature document contains four \texttt{<ds:Certificate>} entries, two of which are
identical.

The example uses a resource file, *signature_template.store.rsa.xml*. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same from each language.

- Resource Files
- JScript Code (storeProp.js)
- Visual Basic Code (storeProp.frm)
- C/C++ Code (storeProp.cpp)
- Output for the store Property Example

### Applies To

IXMLDigitalSignature

### Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

### See Also

IXMLDigitalSignatureEx | sign Method | verify Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Resource:
signature_template.store.rsa.xml

The following resource is used as input for the store example in JScript, Visual Basic, and C/C++.

<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="obj" Mime-Type="text/plain">Hello, World!</ds:Object>
  <ds:KeyInfo/>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig, certStore, infile;

NOKEYINFO = 0;
KEYVALUE = 1;
CERTIFICATES = 2;
PURGE = 4;

CAPICOM_MEMORY_STORE = 0
CAPICOM_LOCAL_MACHINE_STORE = 1
CAPICOM_CURRENT_USER_STORE = 2
CAPICOM_ACTIVE_DIRECTORY_USER_STORE = 3

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
PROV_RSA_FULL = 1;
// Change this key container name to your own if necessary.
RSA_KEY = "MyRSAFullKeys";

csp = PROV_RSA_FULL;
key = RSA_KEY;
infile = "signature_template.store.rsa.xml";

if (InitXML())
{
    if (LoadXML(infile)) {
        alert("Sign with WriteKeyInfo = CERTIFICATES:";
        SignXML(CERTIFICATES|PURGE);

        if (certStore) {
            xmldsig.store = certStore;
            alert("certStore set on xmldsig.");
            alert("number of certifcates in the store = "
                +certStore.CERTIFICATES.count+"\n");
        }
        alert("Sign with WriteKeyInfo = CERTIFICATES:";
        SignXML(CERTIFICATES|PURGE);
    }
}

/////////  Helper functions: ///////////
function alert(str) { WScript.echo(str); }

////////// Set signature for signing. //////////
function InitXML()
{
  try {
    xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmldsig = new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
  }
  catch (e) {
    alert("Installation of mxsml5 is required to run this app.
");
    return false;
  }

  xmldoc.async = false;
  xmldoc.preserveWhiteSpace = true;
  xmldoc.validateOnParse = false;
  xmldoc.resolveExternals = false;

  // Create and open a CAPICOM store object.
  certStore = new ActiveXObject("CAPICOM.Store.2");
  certStore.Open(CAPICOM_CURRENT_USER_STORE, "MY", 0);

  return true;
}

function LoadXML(file)
{
  if (xmldoc == null) {
    alert("must instantiate xml dom\n");
    return false;
  }

  if (!xmldoc.load(file)) {
    alert("Can't load " + file + "\n");
    return false;
  }

  xmldoc.setProperty("SelectionNamespaces", DSIGNS);
  xmldsig.signature = xmldoc.selectSingleNode("//ds:Signature");

  return true;
}

function SignXML(fwWriteKeyInfo)
{
  if (!xmldsig.signature)
  {
    alert("Invalid signature template\n");
    return false;
  }

  // Sign the XML document with the key information.
  // ...
}
var oKey = xmldsig.createKeyFromCSP(csp, "", key, 0);
if (!oKey)
{
    alert("Invalid key.\n");
    return false;
}

var oSignedKey = xmldsig.sign(oKey, fwWriteKeyInfo);
if (!oSignedKey)
{
    alert("sign failed.\n");
    return false;
}

alert("The specified data was signed successfully.\n"+
        "Resultant signature:\n"+
        xmldoc.xml + "\n");
return true;

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the signature_template.store.rsa.xml and paste it into a text file. Save the file as signature_template.store.rsa.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as storeProp.js, in the same directory where you saved signature_template.store.rsa.xml.
4. From a command prompt, navigate to this directory, then type "cscript storeProp.js".

    Note   Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe or cscript.exe), if it is not already installed.

5. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source: storeProp.frm

' Be sure to select "CAPICOM v2.0 Type Library"
' and "Microsoft XML v5.0" options from the
' "Project->References..." menu item in
' Visual Studio.
'
Dim xmldoc As New DOMDocument50
Dim xmlDsigt As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim dataObj As IXMLDOMNode
Dim infile, provType, keyContainer
Dim certStore As New store

Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
Const PROV_RSA_FULL = 1
' Change this key container name to your own if necessary.
Const RSA_KEY = "MyRSAFullKeys"

Private Function WriteLine(ByVal str As String)
  Text1.Text = Text1.Text + str + vbNewLine
End Function

Private Function writeClear()
  Text1.Text = ""
End Function

Private Function LoadXML(ByVal file As String)
' Read input xml file and display the content in the text3.
Path = App.Path + "\" + file
xmldoc.async = False
xmldoc.preserveWhiteSpace = True
xmldoc.validateOnParse = False
xmldoc.resolveExternals = False
If xmldoc.Load(Path) = False Then
  WriteLine "Can't load " + Path
  WriteLine "Reason: " + xmldoc.parseError.reason
  LoadXML = False
  Exit Function
End If
xmldoc.setProperty "SelectionNamespaces", DSIGNS
Set xmlDsigt.signature = xmldoc.selectSingleNode(".//ds:Signature"
LoadXML = True
End Function

Private Function SignXML(fwWriteKeyInfo As Long)
  If xmlDsigt.signature.Is Nothing Then
    WriteLine "Invalid signature template."
SignXML = False
Exit Function
End If

If keyContainer = "" Then
    WriteLine "Invalid key container."
    SignXML = False
    Exit Function
End If

Set oKey = xmldsig.createKeyFromCSP(provType, "", keyContainer, 
If oKey Is Nothing Then
    WriteLine "Invalid key"
    SignXML = False
    Exit Function
End If

Set oSignedKey = xmldsig.sign(oKey, fwWriteKeyInfo)
If oSignedKey Is Nothing Then
    WriteLine "sign failed."
    SignXML = False
    Exit Function
End If

WriteLine "The specified data was signed successfully."
WriteLine "Resultant signature: "
WriteLine xmlldoc.xml

SignXML = True
End Function

Private Sub Form_Load()
    'Set text box to use form to determine its width
    'and height when form is loaded
    Text1.Left = 100
    Text1.Top = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
    provType = PROV_RSA_FULL
    keyContainer = RSA_KEY
    infile = "signature_template.store.rsa.xml"
    writeClear
    certStore.open CAPICOM_CURRENT_USER_STORE, "MY", CAPICOM_STORE_0
    If LoadXML(infile) = True Then
        WriteLine "Sign with WriteKeyInfo = CERTIFICATES:"
        SignXML CERTIFICATES + PURGE
        Set xmlldsig.store = certStore
        WriteLine "certStore set on xmlldsig."
WriteLine "number of certificates in the store = " & certStore.CERTIFICATES.Count
WriteLine ""
WriteLine "Sign with WriteKeyInfo = CERTIFICATES:
SignXML CERTIFICATES + PURGE
End If
End Sub

Private Sub Form_Resize()
    'Set text box to use form in determining its width and height when form is resized
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template from Resource Files and paste it into a text file. Save the file as signature_template.store.rsa.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as storeProp.vbp to the same directory where you saved signature_template.store.rsa.xml. Name the form file storeProp.frm.
4. Create references to MSXML 5.0 and CAPICOM 2.0. To do this, select References... from the Project menu, and then check the boxes for Microsoft XML, v5.0 and CAPICOM v2.0 Type Library.
5. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named "Text1". In the Properties window, set the value for the ScrollBars property to "2 – Vertical" and the value of the MultiLine property to True.
6. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor to replace whatever code is already there.
7. Execute the code by selecting Start from the Run menu.
8. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
```c++
#include <stdio.h>

#import <msxml5.dll>
using namespace MSXML2;
#define PROV_RSA_FULL 1

#import <CAPICOM.dll>
using namespace CAPICOM;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
// Change this key container name to your own if necessary.
#define RSA_KEY "MyRSAFullKeys"
#define INFILE "signature_template.store.rsa.xml"

_bstr_t infile, outfile;
IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;
IStore2Ptr certStore = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}
```
VARIANT_BOOL SignXML(XMLDSIG_WRITEKEYINFO fwWriteKeyInfo) {
    if (xmlsig->signature == NULL) {
        printf("Invalid signature template\n");
        return false;
    }

    IXMLDSigKeyPtr pKey = xmlsig->createKeyFromCSP(
        PROV_RSA_FULL, ",", RSA_KEY, 0);
    if (pKey == NULL) {
        printf("Invalid key\n");
        return false;
    }

    IXMLDSigKeyPtr pKeyOut = xmlsig->sign(pKey, fwWriteKeyInfo);
    if (NULL == pKeyOut) {
        printf("sign failed.\n");
        return false;
    }

    printf("The specified data was signed successfully.\n");
    printf("Resultant signature:\n");
    printf((LPCSTR)xmldoc->xml);
    printf("\n");

    return true;
}

VARIANT_BOOL initObjects() {
    HRESULT hr;
    if (FAILED(xmlsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    if (FAILED(xmlldoc.CreateInstance(__uuidof(DOMDocument50))) ) {
        printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    xmlldoc->async = VARIANT_FALSE;
    xmlldoc->validateOnParse = VARIANT_FALSE;
    xmlldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmlldoc->resolveExternals = VARIANT_FALSE;
hr = certStore.CreateInstance(__uuidof(Store));
if (FAILED(hr)) {
    printf("Installation of CAPICOM Store 2 failed.\n");
    return VARIANT_FALSE;
}

hr = certStore->Open(CAPICOM_CURRENT_USER_STORE, "MY", CAPICOM_STORE_OPEN_READ_ONLY);
if (FAILED(hr)) {
    printf("certStore->Open failed.\n");
    return VARIANT_FALSE;
}

objectsAreInitialized = VARIANT_TRUE;
return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
    if (certStore) certStore.Release();
}

void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    infile = INFILE;

    if (VARIANT_TRUE == LoadXML(infile)) {
        printf("Sign with fwWriteKeyInfo = CERTIFICATES:\n");
        SignXML(CERTIFICATES);

        if (certStore) {
            xmldsig->store = certStore;
            printf("store set on xmldsig.\n");
            printf("number of certificates in the store = %d\n\n",
                   certStore->Certificates->Count);
        } else {
            
        }
printf("invalid certstore\n");
}

printf("Sign with fwWriteKeyInfo = CERTIFICATES:\n");
SignXML(XMLDSIG_WRITEKEYINFO(CERTIFICATES|PURGE));
}
cleanObjects();
CoUninitialize();

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "storeProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight storeProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_template.store.rsa.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature template file from Resource Files, and paste it into the text file you just created.
9. Repeat steps 5-8 for the C++ file above (store.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting Build storeProj.exe from the Build menu.
13. Execute the sample application by selecting !Execute storeProj.exe from the Build menu.
14. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

The output below corresponds to the results from two calls to `sign`. The first call is made without setting the certificate store on the XML Digital Signature object (certStore). The second call is made after setting the certificate store. The result from the first call contains only one `<X509Data>` element. This is because when CERTIFICATES flag is passed in, `sign` tries to get the copy of the certificate of the key used to sign and, when the certificate is found as the key is created using `createKeyFromCSP`, it is inserted in the signed signature document.

The result from the second call contains four `<X509Data>` elements. This is because the `sign` method first tries to get the certificate of the key used and inserts any found certificate into the resultant signature document, as shown above. Because the certificate store is set, the `sign` method then gets all the certificates from that store and inserts them into the signed signature. In this case, there are three certificates in the store.

Output from first call

**Sign with WriteKeyInfo = CERTIFICATES:**
The specified data was signed successfully.

Resultant signature:
```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nh4==</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>IFILa99v5A6V1v37H+OiqOObmGyRaprUzlV3dPbx88YzDdIu9GooyMcsGvpBshBbHQoMso+/VctXUA==</ds:SignatureValue>
  <ds:Object Id="obj" MimeTypetext/plain>Hello, World!</ds:Object>
  <ds:KeyInfo><X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
    MIIE7jCCBK2gAwIBAgIKJMLTcgABAAAACDAJBgqhkj0AQQDMIGYMwIwYJKoZIhvcN
    FrpbnZvQHnLY3VvZw15eGisLmNvbTELMAgGA1UEBlMCCVVMxExDeMBgNVBAgTBV
    CwYDVQQHEwRDaXR5MRYwFAYDVQQKEw1TZWJN1cmUgTXkgWGlsmRUwEwYDVQQLEx
    ```
Output from second call

certStore.set on xmlmdsig.
number of certificates in the store = 3

Sign with WriteKeyInfo = CERTIFICATES:
The specified data was signed successfully.

Resultant signature:
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#;Object">
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
  </ds:Reference>
  <ds:SignatureValue>
    IFILa99v5A6V1v37H+OiqO0bmGyRaprUzlV3dPbx88YzDdIu9GooyMcsGvpBshBbHQ0Mso+/VctXUA==
  </ds:SignatureValue>
  <ds:Object Id="obj" MimeType="text/plain">Hello, World!</ds:Object>
</ds:Signature>
"/x509Certificate
"/x509Data
</ds:KeyInfo>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Methods

[This feature was first implemented for MSXML 5.0.]

This section describes the following methods for XML digital signatures.

- **createKeyFromCertContext**
  Creates and returns a key handle from a certificate context as defined in CryptoAPI. Both the private and public keys are accessible through the resultant key handle.

- **createKeyFromCSP**
  Retrieves a key handle from a key container within a specified Cryptographic Service Provider (CSP). This key handle is used to access the private key used for signing or the public key for signature verification.

- **createKeyFromHMACSecret**
  Extracts and returns a key handle from a base-64 encoded secret value used in the HMAC algorithm. This key handle is used to access a key that can be used for both signing and signature verification.

- **createKeyFromHMACSecretBinary**
  Creates and returns a key handle from an unencoded binary secret value used in an HMAC algorithm. The key accessible through this handle may be used for signing and verifying an XML document or fragment.

- **createKeyFromNode**
  Creates and returns a key handle based on the information contained in the <ds:KeyInfo> element and its descendant elements (<ds:KeyInfo> is a child of a <ds:Signature> element). The key handle is used to access the key that is used for signature verification.

- **createSAXProxy**
  Creates and returns a SAX proxy object. The setReferenceData method uses this object.
to assign a SAX stream as the new data source of the signature object or template.

getCSPHandle
Retrieves the handle to the cryptographic service provider resulted from calling the IXMLDigitalSignature::createKeyFromCSP method.

getStoreHandle
Gets a handle to the certificate store from which the certificate can be retrieved and validated.

getVerifyingCertificate
Retrieves the certificate associated with this key that is used in signature verification.

getVerifyingCertificateContext
Retrieves the certificate context associated with this key.

setDebugOutput
Sets the target for debug output.

setReferenceData
Assigns a new data source to a specified <ds:Reference> element in a signature object.

setStoreHandle
Sets a certificate store handle on the signature object, so that certificate information can be inserted into the <ds:KeyInfo> element.

sign
Signs data referenced in a <ds:Signature> element.

verify
Verifies the signature of a data referenced in the <ds:Signature> element.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
createKeyFromCertContext Method

[This feature was first implemented for MSXML 5.0.]

Extracts a key from a specified certificate context obtainable from any of the CryptoAPI methods returning the CERT_CONTEXT structure.

C/C++ Syntax Using Smart Pointers

```c
IXMLDSigKeyPtr keyRef =
    objIXMLDigitalSignatureEx->createKeyFromCertContext(pCertContext);
```

C/C++ Syntax

```c
HRESULT createKeyFromCertContext(
    void* pCertContext,
    IXMLDSigKey** keyRef);
```

Parameters

- **pCertContext [in]**
  Pointer to a data structure that contains a certificate context that can be obtained from a CryptoAPI function returning a CERT_CONTEXT structure.

- **keyRef [out, retval]**
  In C/C++ this is a reference to the resultant key object. In C/C++ using smart pointer classes this is the key object.

Return Values

- **S_OK**
  The method succeeded and the resultant keyRef parameter points to a valid XMLDSigKey object.

- **E_FAIL**
  The method failed and the resultant keyRef parameter is NULL.
Remarks

This method can be used to access a private key for signing and is available for C/C++ applications only.

Example

This C/C++ example illustrates how to sign an XML document with a key generated from a certificate context stored on your local machine. Generation of the key is encapsulated in the function GetCertContext, which uses three CryptoAPI functions: CertOpenSystemStore, CertFindCertificateInstore, and CertClosestore.

This example uses one resource file. You should get output similar to that provided at the end of the sample.

- Resource File (signature_template.rsa.xml)
- C/C++ Code (KeyFromCertContext.cpp)
- Output for the KeyFromCertContext Example

Applies To

IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

createKeyFromCSP Method | createKeyFromHMACSecret Method | createKeyFromHMACSecretBinary Method | createKeyFromNode Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
The resource used in this example is an XML signature template. The data to be signed is an embedded text string, "Hello, World!".

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="obj">Hello, World!</ds:Object>
</ds:Signature>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source:
KeyFromCertContext.cpp

#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>
#import <msxml5.dll>
using namespace MSXML2;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define SIG_IN "signature_template.rsa.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignatureExPtr xmlsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

// Get the first certificate context from the "MY" system store.
PCERT_CONTEXT GetCertContext()
{
    HCERTSTORE hCertStore=NULL;
    PCERT_CONTEXT  pCert=NULL;
    #define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING

    hCertStore = CertOpenSystemStore(0, "MY");
    if (!hCertStore) {
        printf("failed to open store\n");
        return NULL;
    }

    pCert = CertFindCertificateInStore(
        hCertStore,
        MY_ENCODING_TYPE,
        0,
        CERT_FIND_ANY,
        NULL,
        NULL);
    if (!pCert) {
        printf("Failed to find certificate in store\n");
        return NULL;
    }

    CertCloseStore(hCertStore, CERT_CLOSE_STORE_CHECK_FLAG);
    return pCert;
}
// Load an XML file (a signature template), sigFile, as a DOM object
VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

// Sign the XML data embedded in the signature template.
VARIANT_BOOL SignXML()
{
    IXMLDSigKeyPtr pKey, pKeyOut;

    if (xmldsig->signature == NULL) {
        printf("Invalid signature template.\n");
        return VARIANT_FALSE;
    }

    PCCERT_CONTEXT pCertContext = GetCertContext();
    pKey = xmldsig->createKeyFromCertContext((void*)pCertContext);

    if (pKey == NULL) {
        printf("Invalid key from CertContext\n");
        return VARIANT_FALSE;
    }

    pKeyOut = xmldsig->sign(pKey, KEYVALUE);
    if (pKeyOut == NULL) {
        printf("Invalid signature.\n");
        return VARIANT_FALSE;
    }
}

CertFreeCertificateContext(pCertContext);
printf("The data referenced in the signature template was signed successfully. Resultant signature: \n\n%s\n", (LPCSTR)xmldoc->xml);
return VARIANT_TRUE;
}

// Helper function that creates and initiates required DOM objects:
VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.\n"
            return VARIANT_FALSE;
    }
    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50))) ) {
        printf("Installation of msxml5 is required to run this app.\n"
            return VARIANT_FALSE;
    }
    xmldoc->async = VARIANT_FALSE;
xmldoc->validateOnParse = VARIANT_FALSE;
xmldoc->preserveWhiteSpace = VARIANT_TRUE;
xmldoc->resolveExternals = VARIANT_FALSE;
    objectsAreInitialized = VARIANT_TRUE;
    return VARIANT_TRUE;
}

// Helper function that releases objects of the application scope:
void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
}

void main()
{
    if ( CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }
    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }
    if(VARIANT_TRUE == LoadXML(SIG_IN)) {
        if (VARIANT_TRUE != SignXML()) {
            printf("Signature failed\n");
        exit(-1);
    }
}
printf("exit with failure.\n");
}
}
cleanObjects();
CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "keyFromCertContextProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight keyFromCertContextProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_template.rsa.xml" in the File name text box.
7. Click OK.
8. Copy the XML data file from Resource: signature_template.rsa.xml, and paste it into the text file you just created.

Note You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 5-8 for the C++ file above (keyFromCertContext.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting Build keyFromCertContextProj.exe from the Build menu.
13. Execute the sample application by selecting !Execute keyFromCertContextProj.exe from the Build menu.
14. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
When you build and run the **KeyFromCertContext** code, you should get output in the following format. The likely differences are due to the different certificate contexts (or keys) used.

The data referenced in the signature template was signed successfully.

Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:DigestValue>q7jECW4dAYAz6sgkuPIeyzz/vdw="</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>vqpqpa+/tqW4dJTv3F3K022UI6Di00EI8kud5BcKSRXofc3Hb4Ihm1om8NTDPRToBPtsQ6oCA==</ds:SignatureValue>
  <ds:Object Id="obj">Hello, World!</ds:Object>
<ds:KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#"><KeyValue><RSAKeyValue><Modulus>wylu+tpOV2x0LM1Up0iQH5v4gI/0f44sPKZxgfzw9BKWiJ2QNhfcSibRw5LJj1u2yOIf3LXu+7Rd6ow="</Modulus><Exponent>AQABAA==</Exponent></RSAKeyValue></KeyInfo></ds:Signature>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
createKeyFromCSP Method

[This feature was first implemented for MSXML 5.0.]

Extracts a key from a specified cryptographic service provider (CSP) and a specified key container. The resultant key can be used to sign or verify a signature.

[JScript]

**Script Syntax**

```javascript
var objKey = objXMLDigitalSignature.createKeyFromCSP(
    cspType,
    cspName,
    keyContainerName,
    flag);
```

[Visual Basic]

**Visual Basic Syntax**

```vbscript
var objKey = objXMLDigitalSignature.createKeyFromCSP(
    _cspType, _
    _cspName, _
    _keyContainerName, _
    _flag)
```

[C/C++]

**C/C++ Syntax Using Smart Pointers**

```c
IXMLDSigKeyPtr objKey =
    objXMLDigitalSignature->createKeyFromCSP(
        long cspType,
        _bstr_t cspName,
        _bstr_t keyContainerName,
        DWORD flag
    );
```
C/C++ Syntax

HRESULT createKeyFromCSP(
    long cspType,
    BSTR cspName,
    BSTR keyContainerName,
    long flag,
    IXMLDSigKey** objKey);

Parameters

cspType
A cryptographic service provider (CSP) type as defined in CryptoAPI. Only
PROV_RSA_FULL and PROV_DSS_DH are supported.

cspName
The name of the CSP. To use the default provider, set this to an empty string
(""").

keyContainerName
The name of the key container within the specified CSP. The container
holds the key or keys issued by a certificate authority.

flag
Reserved. Must be zero.

objKey [out, retval]
In C/C++ this points to a key object implementing the IXMLDSigKey
interface. In JScript, Visual Basic, and C/C++ with smart pointer class
wrappers, this is the key object itself. The resultant key object can then be
used in the call to the sign method or the verify method.

Return Values

This methods returns the standard CryptoAPI return values, including the
following:

S_OK
The value returned if successful.

E_FAIL
The value returned if there was insufficient information, if the method call
was not authorize, etc.

E_ACCESSDENIED
The value returned if the user does not have access to the security item.

Remarks

A digital certificate must have been installed locally for the `createKeyFromCSP` method to work properly.

Example

This example illustrates how to use the `createKeyFromCSP` method to generate a cryptographic key from the installed CSP, and uses this key to sign a document.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource File (`signature_template.rsa.xml`)
- JScript Code (`KeyFromCSP.js`)
- Visual Basic Code (`KeyFromCSP.frm`)
- C/C++ Code (`KeyFromCSP.cpp`)
- Output for the KeyFromCSP Example

Applies To

IXMLDigitalSignature | IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXMLDigitalSignatureEx | `createKeyFromCertContext Method` | `createKeyFromHMACSecret Method` | `createKeyFromHMACSecretBinary Method` | `createKeyFromNode Method` | `sign Method` | `verify Method`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
The **KeyFromCSP** example uses a signature template file that contains internally referenced text data to be signed. This file has the following characteristics:

- The signed text data, "Hello, World!", is held as the content of a `<ds:Object Id="HelloWorld">` element.
- The algorithm used to generate the signature is [http://www.w3.org/2000/09/xmldsig#dsa-sha1](http://www.w3.org/2000/09/xmldsig#dsa-sha1).
- The signature information, `<ds:KeyInfo>`, is missing and will be added when the data is signed.

### Signature Template (signature_template.rsa.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#HelloWorld" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue />
  <ds:Object Id="HelloWorld"
    MimeType="text/plain">Hello, World!</ds:Object>
</ds:Signature>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
JScript Source: KeyFromCSP.js

var xmldoc, xmlsig, dsigKey
var szResult = "";
KEYVALUE = 1;
DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
PROV_RSA_FULL = 1;
// Change this key container name to your own if necessary.
RSA_KEY = "MyRSAFullKeys";

csp = PROV_RSA_FULL;
key = RSA_KEY;
sig = "signature_template.rsa.xml"

if (loadSignature(sig)) {
    if (signWithKey(csp, key)) {
        alert(szResult);
    }
}

///////// Helper functions: ///////////
function alert(str) { WScript.echo(str); }

///////// Set signature for signing. /////////
function loadSignature(file)
{
    try {
        xmldoc = new ActiveXObject("Msxml2.DOMDOcument.5.0");
        xmlsig = new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
    } catch (e) {
        alert("Installation of mxsml5 is required to run this app.\n");
        return false;
    }

    xmldoc.async = false;
    xmldoc.preserveWhiteSpace = true;
    xmldoc.validateOnParse = false;

    if (!xmldoc.load(file)) {
        alert("Can't load "+ file + "\n");
        return false;
    }
    szResult += "\nInput signature template:\n\" + xmldoc.xml;

xmldoc.setProperty("SelectionNamespaces", DSIGNS);
  xmldsig.signature = xmldoc.selectSingleNode(".//ds:Signature");

  return true;
}

function signWithKey(dwCspType, szKeyContainer) {
  if (!xmldsig.signature) {

    alert("Must set signature template before attempting signing.\n")
    return false;
}  
  var oKey = xmldsig.createKeyFromCSP(dwCspType,"",szKeyContainer,0)
  var oSignedKey = xmldsig.sign(oKey, KEYVALUE);
  if (oSignedKey == null) {
    alert("Signing failed.\n");
  }  
  else {
    szResult += ("\nThe data referenced in the signature template
" + "was signed successfully.\n" + "Resultant signature:\n\n" + xmldoc.xml );
  }
  return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template from Resource: signature_template.itext.dsa.xml and paste it into a text file. Save the file as signature_template.rsa.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as signature.js, in the same directory where you saved signature_template.rsa.xml.
4. From a command prompt, navigate to this directory, then type "cscript signingLetter.js".
**Note** Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe and cscript.exe), if it is not already installed.

5. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source:
KeyFromCSP.frm

Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim DispStr, file
Dim outfile, provType, keyContainer

Const NOKEYINFO = 0
Const KEYVALUE = 1
Const CERTIFICATES = 2
Const PURGE = 4
Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
Const PROV_RSA_FULL = 1

Private Function LoadXML()
    ' Read the input xml file and display the content in the text3.
    DispStr = ""
    If xmldoc.xml = "" Then
        If file = "" Then
            DispStr = "invalid input xml file name"
            LoadXML = False
            Exit Function
        End If
    End If
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    If xmldoc.Load(Path) = False Then
        DispStr = DispStr + vbNewLine + _
        "can't load " + Path + vbNewLine + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    DispStr = "Input signature element:" + vbNewLine + vbNewLine _
    + xmldoc.xml + vbNewLine
    LoadXML = True
End Function
Private Function SignXML()
    If xmldoc.xml = "" Then
        DispStr = "signature template is empty."
        SignXML = False
        Exit Function
    End If

    If keyContainer = "" Then
        DispStr = DispStr + "invalid keyContainer."
        SignXML = False
        Exit Function
    End If

    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.setProperty "SelectionNamespaces", DSIGNS

    Set signature = xmldoc.selectSingleNode("//ds:Signature")
    Set xmldsig.signature = signature

    Set oKey = xmldsig.createKeyFromCSP(provType, "", keyContainer,
    Set oSignedKey = xmldsig.sign(oKey, KEYVALUE)

    If oSignedKey Is Nothing Then
        DispStr = "sign failed." + vbCrLf
    End If

    DispStr = DispStr + "The data referenced in the signature template"
    + "was signed successfully." + vbCrLf + "Resultant signature:" + vbCrLf + vbCrLf

    DispStr = DispStr + xmldoc.xml + vbCrLf
    output = App.Path + "" + outfile
    xmldoc.save (output)
    SignXML = True
End Function

Private Sub Form_Load()
    'Set text box to use form to determine its width
    'and height when form is loaded
    Text1.Left = 100
    Text1.Top = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750

    provType = PROV_RSA_FULL
    ' Change this key container name to your own if necessary.
    keyContainer = "MyRSAFullKeys"
file = "signature_template.rsa.xml"
outfile = "signature_document.rsa.xml"

If LoadXML = False Then
    Text1.Text = DispStr
    Exit Sub
End If

If SignXML = False Then
    Text1.Text = DispStr
    Exit Sub
End If

Text1.Text = DispStr
End Sub

Private Sub Form_Resize()
    'Set text box to use form in determining its width and height
    'when form is resized
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template Resource: signature_template.rsa.xml, and paste it into a text file. Save the file as signature_template.rsa.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as KeyFromCSP.vbp to the same directory where you saved signature_template.rsa.xml. Name the form file KeyFromCSP.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v5.0.
5. Double click on the TextBox icon from the tools panel. This results in a TextBox control being placed on the project's form named "Text1". In the Properties window, modify the following properties for "Text1": select "2 - Vertical" as the value for the ScrollBar property and set the value of the MultiLine property to True.
6. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor to replace whatever code is already there.
7. Execute the code by selecting Start from the Run menu.
8. Verify that your output is similar to that listed in the Output topic. To view the output more easily, resize the form at run time. You can also verify that
the output file containing the XML digital signature file (signature_document.rsa.xml) exists in the same directory where you created and ran your project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: KeyFromCSP.cpp

#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL loadSignature(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    printf("\nInput signature template:\n%s\n", (LPCSTR)xmldoc->xml

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode(".//ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL signWithKey(long CspType, _bstr_t keyContainer)
{
    if (!objectsAreInitialized) {
        printf("Must initialize object for signing.\n");
        return VARIANT_FALSE;
    }
// Get the value of the signature property, and see if
// it has been set.
if (xmlDsig->signature == NULL) {
    printf("Must set signature template before signing.\n");
    return VARIANT_FALSE;
}

IXMLDSigKeyPtr dsigKey = xmlDsig->createKeyFromCSP(CspType, ",", k
if (dsigKey == NULL) {
    printf("Failed to create key from CSP\n");
    return VARIANT_FALSE;
}

IXMLDSigKeyPtr signedKey = xmlDsig->sign(dsigKey, KEYVALUE);
if (signedKey != NULL) {
    printf("The data referenced in the signature template ");
    printf("was signed successfully.\nResultant signature: \n\n");
    printf("%s\n", (LPCSTR)xmlDoc->xml);
} else {
    printf("Signing failed.\n");
    if (dsigKey) dsigKey.Release();
    return VARIANT_FALSE;
}

if (dsigKey) dsigKey.Release();
if (signedKey) signedKey.Release();

return VARIANT_TRUE;
}

VARIANT_BOOL initObjects()
{
    if (FAILED(xmlDsig.CreateInstance(__uuidof(MXDigitalSignature50))
        printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    if (FAILED(xmlDoc.CreateInstance(__uuidof(DOMDocument50)))
        return VARIANT_FALSE;
    }

    xmlDoc->async = VARIANT_FALSE;
    xmlDoc->validateOnParse = VARIANT_FALSE;
    xmlDoc->preserveWhiteSpace = VARIANT_TRUE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}
```cpp
void cleanObjects()
{
    if (xml_doc) xml_doc.Release();
    if (xml_dsig) xml_dsig.Release();
}

void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    if (VARIANT_TRUE == loadSignature("signature_template.rsa.xml"))
    {
        if (VARIANT_TRUE != signWithKey(
            // Change this key container name to your own if necessary.
            PROV_RSA_FULL, "MyRSAFullKeys"))
        {
            printf("exit with failure.\n");
        }
    }

    cleanObjects();
    CoUninitialize();
}
```

**Try It!**

1. Ensure that you have completed all the procedures in [Getting Started with XML Digital Signatures](#).
3. From the **File** menu, select **New**. On the **Projects** tab of the **New** dialog box that appears, select **Win32 Console Application** in the left pane. Then type "KeyFromCSPProj" in the **Project name** field. For the project **Location** field, either accept the default setting or choose another location. Click **OK**.
4. The **Win32 Console Application** property page will appear. For the type of Console Application, select **An empty project** and click **Finish**. When the **New Project Information** box displays, click **OK**.
5. Select **File View** on the project browser, and highlight **KeyFromCSPProj files**. From the **File** menu, select **New**.

6. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "signature_template.rsa.xml" in the **File name** text box.

7. Click **OK**.

8. Copy the XML signature template file from **Resource: signature_template.rsa.xml**, and paste it into the text file you just created.

9. Repeat steps 5-8 for the C++ file above (**KeyFromCSP.cpp**).

10. From the **Project** menu, click **Settings...**, then click the **Link** tab.

11. In **Object/library** modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click **OK**.

12. Build the sample by selecting **Build KeyFromCSPProj.exe** from the **Build** menu.

13. Execute the sample application by selecting **!Execute signatureProj.exe** from the **Build** menu.

14. Verify that your output is similar to that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

When you build and run the KeyFromCSP example, you should get output similar to the following. Some of your KeyInfo values will be different if you use a different key for signing the document. Additionally, the KeyInfo items here have been rearranged for better readability.

Input signature template:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#HelloWorld" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="HelloWorld" MimeType="text/plain">Hello, World!</ds:Object>
</ds:Signature>
```

The data referenced in the signature template was signed successfully.

Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#HelloWorld" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>BQKNAJjTw98dp7Uvw4oAmJU8LSU=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    nPHDVveS1kEaxiYl/W0hq4aeN4niqGcfvZ4ULaDqmXCSRVhAkIq49bbMI/EeIoMI +YV8DsRFlaNMmNZsnEorc0eefLWwULG0aRcd4ipnxp2MKRRaCLq4t3bq5pt1LKUk u7AMY9SAjEdD1RA0m0WTcsing0o=
  </ds:SignatureValue>
  <ds:Object Id="HelloWorld" MimeType="text/plain">Hello, World!</ds:Object>
</ds:Signature>
```
<KeyValue>
  <RSAKeyValue>
  <Modulus>
  xnajBJA0aCVfy7q3wgSni1ZPWes1c8YdcoJdgqLw+RVQliykwcVb4Xgv9UB1SwMAiNfzA+dXsSeXcdcA7NJ15zmJbs3Xd1DSfEf4cboyCv/ZX9V7CvdG9ScEwaf0I1QNTueK8a/3WRNcqkE=
  </Modulus>
  <Exponent>AQAB</Exponent>
  </RSAKeyValue>
  </KeyValue>
</KeyInfo>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign...
createKeyFromHMACSecret Method

[This feature was first implemented for MSXML 5.0.]

This method generates a symmetric key based on a Base64-encoded value of an HMAC secret. The resultant key can be used in sign and verify.

[JS/Script]

**JS/Script Syntax**

```javascript
var objKey = objXMLDigitalSignature.createKeyFromHMACSecret(
    secret,
    length
);
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
Set objKey = objXMLDigitalSignature.createKeyFromHMACSecret(
    secret, _
    length _
)
```

[C/C++]

**C/C++ Syntax Using Smart Pointers**

```c++
IXMLDSigKeyPtr objKey = objXMLDigitalSignature->createKeyFromHMACSecret(  
    (BSTR)secret,  
    (LONG)length  
);
```

**C/C++ Syntax**

```c++
HRESULT createKeyFromHMACSecret(
    BSTR secret,  
    LONG length,
```
Parameters

secret [in]
A Base64 encoded HMAC secret value.

length [in]
Number of bits used to represent the HMAC secret value for signing or verification. The value of this parameter corresponds to the content of the <ds:HMACOutputLength> element in <ds:Signature>. It can be a positive number in multiples of eight (8), but no greater than 128, or a negative one (-1). The positive value is used by sign or verify as the bit length of the secret. For -1, the default length defined in CryptoAPI or the content of <ds:HMACOutputLength> element is used instead.

The actual length used in sign or verify depends on the combination of the value of the length parameter and the content of <ds:HMACOutputLength> element in the signature template or document. The following table shows the possible combinations, the length used to verify, and the values returned.

<table>
<thead>
<tr>
<th>length value</th>
<th><a href="">ds:HMACOutputLength</a> content</th>
<th>Actual length used in sign or verify</th>
<th>Return createKeyFromHMACSecret createKeyFromHMACSecret</th>
<th>E_INVALIDARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>8*n, where n=1, 2, ..., 16</td>
<td>Any value</td>
<td>The value of the length parameter</td>
<td>S_OK</td>
<td></td>
</tr>
<tr>
<td>-1 8*n, where n=1, 2, ..., 16</td>
<td>Any value</td>
<td>The value of <a href="">ds:HMACOutputLength</a></td>
<td>S_OK</td>
<td></td>
</tr>
<tr>
<td>-1 Any value</td>
<td>Any value</td>
<td>The sign or verify method will fail.</td>
<td>S_OK</td>
<td></td>
</tr>
<tr>
<td>-1 Absent</td>
<td>Any value</td>
<td>CryptoAPI default length (128)</td>
<td>S_OK</td>
<td></td>
</tr>
<tr>
<td>Any value</td>
<td>Any value</td>
<td>No key will be created. The objKey parameter is returned</td>
<td>E_INVALIDARG</td>
<td></td>
</tr>
</tbody>
</table>

```C
IXMLDSigKey** objKey);
```
as NULL.

$objKey [out, retval]
In C/C++, this is a reference to the resultant key object implementing the IXMLDSigKey interface. In Visual Basic, scripting languages, and C/C++ with smart pointers, this is the key object itself.

Return Values

The method returns the standard HRESULT values defined in CryptoAPI, including the following:

S_OK
The value returned if successful.
E_FAIL
The value returned if there was insufficient information, if the method call was not authorize, etc.
E_ACCESSDENIED
The value returned if the user does not have access to the security item.
E_INVALIDARG
An invalid argument was used as input.

Remarks

The HMAC secret value is shared by the signer and the signature verifier, and is not to be transmitted with the signature document.

The key generated by this method does not have any corresponding certificate. When this key is used in signing, the $<ds:KeyInfo>$ element and its children will be purged.

When the length parameter value is -1, it is important that the $<ds:HMACOutputLength>$ element contains a positive number that is multiple of eight (8). An empty content or zero value would mean that the signature information will not be used.

Example
This examples illustrates how to sign and verify a document using an HMAC secret value, "secret". The Base64 encoding of this value is "c2VjcmV0", and is used by the createKeyFromHMACSecret method to generate a key for both signing and verifying a piece of data. The data for this example is the text string "Hello, World!".

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- **Resource File (signature-template-enveloping-hmac-sha1.xml)**
- **JScript Code (KeyFromHMACSecret.js)**
- **Visual Basic Code (KeyFromHMACSecret.frm)**
- **C/C++ Code (KeyFromHMACSecret.cpp)**
- **Output for the KeyFromHMACSecret Example**

**Applies To**

IXMLDigitalSignature | IXMLDigitalSignatureEx

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

CreateKeyFromHMACSecretBinary Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<?xml version="1.0" encoding="UTF-8"?>
<SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"
    >
    <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"
    >
        <HMACOutputLength>40</HMACOutputLength>
    </SignatureMethod>
    <Reference URI="#object">
        <DigestMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
        >
            <DigestValue></DigestValue>
        </DigestMethod>
    </Reference>
</SignedInfo>
<SignatureValue></SignatureValue>
<Object Id="object">Hello, World!</Object>
</Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig
var szResult = "";

NOKEYINFO = 0;
KEYVALUE = 1;
CERTIFICATES = 2;
PURGE = 4;

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
sigTmp = "signature-template-enveloping-hmac-sha1.xml";
sigOut = "signature-enveloping-hmac-sha1.xml";
hmacSecret = "c2VjcmV0";
hmacLength = -1;

if (InitXML())
{
  if (LoadXML(sigTmp)) {
    SignXML();
  }

  if (LoadXML(sigOut)) {
    VerifyXML();
  }
}

///// Helper functions: //////////////
function alert(str) { WScript.echo(str); }

///// Set signature for signing. //////////////
function InitXML()
{
  try {
    xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmldsig= new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
  }
  catch (e) {
    alert("Installation of mxsml5 is required to run this app.\n");
    return false;
  }

  xmldoc.async = false;
function LoadXML(file)
{
    if (xml_doc == null) {
        alert("must instantiate xml dom\n");
        return false;
    }

    if (!xml_doc.load(file)) {
        alert("Can't load " + file + "\n");
        return false;
    }
    xml_doc.setProperty("SelectionNamespaces", DSIGNS);
    xml_dsig.signature = xml_doc.selectSingleNode(".//ds:Signature");
    return true;
}

function SignXML()
{
    if (!xml_dsig.signature)
    {
        alert("Invalid signature template\n");
        return false;
    }

    var oKey = xml_dsig.createKeyFromHMACSecret(hmacSecret, hmacLength);
    if (!oKey)
    {
        alert("Failed to create key from HMAC secret value\n");
        return false;
    }

    var oSignedKey = xml_dsig.sign(oKey, KEYVALUE);
    if (!oSignedKey)
    {
        alert("signing failed.\n");
        return false;
    }
    xml_doc.save(sigOut);
    alert("The data referenced in the signature template was "+
        "signed successfully.\n\nResultant signature:\n\n" +
    xml_doc.xml + "\n");
    return true;
}

function VerifyXML()
```javascript
if (!xmlldsig.signature) {
    alert("Invalid signature.
";
    return false;
}
var oKey = xmlldsig.createKeyFromHMACSecret(hmacSecret, hmacLength
if (!oKey)
{
    alert("Failed to create key from HMAC secret value
";
    return false;
}

var oVerifiedKey = xmlldsig.verify(oKey);
if (oVerifiedKey == null) {
    alert("Signature Verification failed.
";
}
alert("\nThe data referenced in the signature object " + 
"was verified successfully.\n");
return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the signature template resource file, and paste it into a text editor. Save the file as signature-template-enveloping-hmac-sha1.xml.
3. Copy the JScript listing above, and paste it into a code editor. Save the file as KeyFromHMACSecret.js, in the same directory where you saved signature-template-enveloping-hmac-sha1.xml.
4. From a command prompt, navigate to this directory, then type "cscript KeyFromHMACSecret.js".

   Note   Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe or cscript.exe), if it is not already installed.

5. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source:
KeyFromHMACSecret.frm

Dim xmldoc As New DOMDocument50
Dim xmlsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim DSIGNS, sigOut, sigTmp As String
Const KEYVALUE = 1

Private Function LoadXML(ByVal file As String)
    ' Read the input xml file and display the content in the text3.
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    If xmldoc.Load(Path) = False Then
        Text1.Text = Text1.Text + "can't load " + Path + vbCrLf + _
        + xmldoc.parseError.reason + vbCrLf
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    Set xmlsig.signature = xmldoc.selectSingleNode("./ds:Signature")
    LoadXML = True
End Function

Private Function SignXML(ByVal secret As String, ByVal length As Long)
    If xmlsig.signature Is Nothing Then
        Text1.Text = Text1.Text + "Invalid signature template."
        SignXML = False
        Exit Function
    End If
    Set oKey = xmlsig.createKeyFromHMACSecret(secret, length)
    If oKey Is Nothing Then
        Text1.Text = Text1.Text + "Failed to create key from HMAC secret value
" + vbCrLf
        SignXML = False
        Exit Function
    End If

    Set oSignedKey = xmlsig.sign(oKey, KEYVALUE)
    If oSignedKey Is Nothing Then
        Text1.Text = Text1.Text + "sign failed." + vbCrLf
    End If
End Function
SignXML = False
Exit Function
End If

xmlDoc.Save SigOut

Text1.Text = Text1.Text + "The data referenced in the signature template " + "was signed successfully." + vbCrLf + "Resultant signature:" + vbCrLf + vbCrLf + xmlDoc.xml + vbCrLf
SignXML = True
End Function

Private Function VerifyXML(ByVal secret As String, ByVal length As Long)
If xmlDoc.signature Is Nothing Then
    Text1.Text = Text1.Text + "Invalid signature object."
    VerifyXML = False
    Exit Function
End If

Set oKey = xmlDoc.createKeyFromHMACSecret(secret, length)
If oKey Is Nothing Then
    Text1.Text = Text1.Text + _
        "Failed to create key from HMAC secret value\n"
    VerifyXML = False
    Exit Function
End If

Set oVerifyKey = xmlDoc.verify(oKey)
If oVerifyKey Is Nothing Then
    Text1.Text = Text1.Text + "verify failed." + vbCrLf
    VerifyXML = False
    Exit Function
End If

Text1.Text = Text1.Text + _
    "The data referenced in the signature object " + _
    "was verified successfully." + vbCrLf + vbCrLf

VerifyXML = True
End Function

Private Sub Form_Load()
    ' Resize the text box control to the size of the form
    Text1.Top = 100
    Text1.Left = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"
sigTmp = "signature-template-enveloping-hmac-sha1.xml"
sigOut = "signature-enveloping-hmac-sha1.xml"
hmacSecret = "c2VjcmV0"
hmacLength = -1
Text1.Text = ""

If LoadXML(sigTmp) = False Then
    Exit Sub
End If

If SignXML(hmacSecret, hmacLength) = False Then
    Exit Sub
End If

If VerifyXML(hmacSecret, hmacLength) = False Then
    Exit Sub
End If

End Sub

Private Sub Form_Resize()
    ' Resize the text box control to the size of the form
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the signature template resource file, and paste it into a text editor. Save the file as signature-template-enveloping-hmac-sha1.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as KeyFromHMACSecret.vbp to the same directory where you saved signature-template-enveloping-hmac-sha1.xml. Name the form file KeyFromHMACSecret.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, and then check the box for Microsoft XML, v5.0.
5. Copy the Visual Basic code listing above, and paste it into the code editor to replace whatever code is already there.
6. Place a TextBox control onto the form. It should be named as Text1 (the default name). Click it to select it in the Properties window, and then modify the following properties for Text1: set the ScrollBars property to "2 – Vertical" and the MultiLine Property to True.
7. Execute the code by selecting **Start** from the **Run** menu.
8. Verify that the output is similar to that listed in the **Output** topic. You can resize the form to more easily view the output.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

#define HMACSECRET "c2VjcmV0"
#define HMACLENGTH -1
#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define SIG_IN "signature-template-enveloping-hmac-sha1.xml"
#define SIG_OUT "signature-enveloping-hmac-sha1.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL SignXML(char *secret, long length)
{
    IXMLDSigKeyPtr pKey, pKeyOut;
    /* Create the key.
    */
    /* Sign the signature block.
    */
    /* Finally, convert the signature object to a DER string.
    */
    return VARIANT_TRUE;
}
if (xmlsig->signature == NULL) {
    printf("Invalid signature template.\n");
    return VARIANT_FALSE;
}

pKey = xmlsig->createKeyFromHMACSecret(_bstr_t(secret), length);
if (pKey == NULL) {
    printf("Invalid key from HMACSecret\n");
    return VARIANT_FALSE;
}

// KEYVALUE is defined in XMLDSIG_WRITEKEYINFO enum.
pKeyOut = xmlsig->sign(pKey, KEYVALUE);
if (pKeyOut == NULL) {
    printf("Invalid signature.\n");
    return VARIANT_FALSE;
}

xmldoc->save(SIG_OUT);
printf("The data referenced in the signature template was signed\n");
return VARIANT_FALSE;
}

if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)))) {
    printf("Installation of msxml5 is required to run this app.\n"
    return VARIANT_FALSE;
}
xmldoc->async = VARIANT_FALSE;
xmldoc->validateOnParse = VARIANT_FALSE;
xmldoc->preserveWhiteSpace = VARIANT_TRUE;
objectsAreInitialized = VARIANT_TRUE;

return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmlsig) xmlsig.Release();
}

void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    if(VARIANT_TRUE == LoadXML(SIG_IN)) {
        if (VARIANT_TRUE != SignXML(HMACSECRET, HMACLENGTH)) {
            printf("exit with failure.\n");
            goto clean;
        }
    }

    if(VARIANT_TRUE == LoadXML(SIG_OUT)) {
        if (VARIANT_TRUE != VerifyXML(HMACSECRET, HMACLENGTH)) {
            printf("exit with failure.\n");
            goto clean;
        }
    }

clean:
    cleanObjects();
CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type " KeyFromHMACSecretProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight KeyFromHMACSecretProj files. From the File menu, select New.
7. Click OK.
8. Copy the XML signature template file from Resource File, and paste it into the text file you just created.
9. Repeat steps 5-8 for the C++ file above (KeyFromHMACSecret.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting Build KeyFromHMACSecretProj.exe from the Build menu.
13. Execute the sample application by selecting !Execute KeyFromHMACSecretProj.exe from the Build menu.
14. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

When you build and run the **KeyFromHMACSecret** code, you should get output similar to the following.

The data referenced in the signature template was signed successfully.

Resultant signature:

```xml
<?xml version="1.0"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
      <HMACOutputLength>40</HMACOutputLength>
    </SignatureMethod>
    <Reference URI="#object">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>gr9ao1nOSq6zgOFaaiCjYGHxB50=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>W5op7/I=</SignatureValue>
  <Object Id="object">Hello, World!</Object>
</Signature>
```

The data referenced in the signature object was verified successfully.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
createKeyFromHMACSecretBinary Method

[This feature was first implemented for MSXML 5.0.]

Extracts a key from a binary HMAC secret value for use by C/C++ applications.

C/C++ Syntax Using Smart Pointers

```c
IXMLDSigKeyPtr objKey=
    oXMLDigitalSignature->createKeyFromHMACSecretBinary(
        (byte*)secret,
        (LONG)lSecretLen,
        (LONG)lHMACOutputLength);
```

C/C++ Syntax

```c
HRESULT createKeyFromHMACSecretBinary(
    CONST byte* secret,
    LONG lSecretLen,
    LONG lHMACOutputLength,
    IXMLDSigKey** keyObj);
```

Parameters

- **secret** [in]
  Binary HMAC secret value.
- **lSecretLen** [in]
  Length of the binary HMAC secret value in bytes.
- **lHMACOutputLength** [in]
  Number of bits used to represent the secret value for signing or verify. The value of this parameter corresponds to the content of the `<ds:HMACOutputLength>` element in `<ds:Signature>`. It can be a positive number in multiples of eight (8), or negative one (-1).

  The following table explains the meanings of the different possible values of `lHMACOutputLength`. 
<table>
<thead>
<tr>
<th>value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive number</td>
<td>Bit length of the secret value. Must be in multiples of eight (8). If</td>
</tr>
<tr>
<td></td>
<td>the length of the given secret value is longer than this number, it</td>
</tr>
<tr>
<td></td>
<td>will be truncated to this number when signing.</td>
</tr>
<tr>
<td>-1</td>
<td>Use the content of <code>&lt;ds:HMACOutputLength&gt;</code> element from the containing</td>
</tr>
<tr>
<td></td>
<td><code>&lt;ds:SignatureMethod&gt;</code> element.</td>
</tr>
</tbody>
</table>

**objKey [out, retval]**

In C/C++ this is the reference to the resultant key object that implements the IXMLDSigKey interface. In C/C++ with smart pointer class wrappers, this is the resultant key object.

**Return Values**

The method returns the standard HRESULT values defined in the CryptoAPI, including the following:

- **S_OK**
  - The value returned if successful.
- **E_FAIL**
  - The value returned if there was insufficient information, if the method call was not authorize, etc.

**Remarks**

This method has the same behavior as the `createKeyFromHMACSecret` method, except that it takes a binary as the HMAC secret value and it can only called from a C/C++ application.

**Example**

This example illustrates how to use the `createKeyFromHMACSecretBinary` method to generate a key for signing and verifying data referenced in an XML digital signature. This example is similar to the [createKeyFromHMACSecretBinary Method](#) example. The difference is that the
binary value of the HMAC secret ("secret") is used, instead of the base64 encoded value ("c2VjcmV0"). The two examples use the same resource file, and their output is identical.

- Resource File (signature-template-enveloping-hmac-sha1.xml)
- C/C++ Code (KeyFromHMACSecretBinary.cpp)
- Output for the KeyFromHMACSecretBinary Example

Applies To

IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

createKeyFromHMACSecret Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"
    ><SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
        <HMACOutputLength>40</HMACOutputLength>
    </SignatureMethod>
    <Reference URI="#object">
      <DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue></DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue></SignatureValue>
  <Object Id="object">Hello, World!</Object>
</Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source:
KeyFromHMACSecretBinary.cpp

#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

// Notice that the Base64 encoded value of "secret" is "c2VjcmV0".
unsigned char HMACSECRET[] = "secret";
#define HMACLENGTH -1
#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define SIG_IN "signature-template-enveloping-hmac-sha1.xml"
#define SIG_OUT "signature-enveloping-hmac-sha1.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignatureExPtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");
    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL SignXML(unsigned char *secret, long length)
IXMLDSigKeyPtr pKey, pKeyOut;

if (xmlsig->signature == NULL) {
    printf("Invalid signature template.\n");
    return VARIANT_FALSE;
}

pKey = xmlsig->createKeyFromHMACSecretBinary( 
    secret, 
    strlen((const char*)secret), 
    length); 
if (pKey == NULL) { 
    printf("Invalid key from HMACsecret\n"); 
    return VARIANT_FALSE;
}

// KEYVALUE is defined in XMLDSIG_WRITEKEYINFO enum. 
pKeyOut = xmlsig->sign(pKey, KEYVALUE); 
if (pKeyOut == NULL) { 
    printf("Invalid signature.\n"); 
    return VARIANT_FALSE;
}

xmldoc->save(SIG_OUT); 
printf("The data referenced in the signature template was signed successfully.\nResultant signature:\n\n%s\n", (LPCSTR)xml)
return VARIANT_TRUE;
}

VARIANT_BOOL VerifyXML(unsigned char* secret, long length) 
{
    IXMLDSigKeyPtr pKey, pKeyOut;

    pKey = xmlsig->createKeyFromHMACSecretBinary( 
        secret, 
        strlen((const char*)secret), 
        length); 
if (pKey == NULL) { 
    printf("Invalid key from HMACsecretBinary\n"); 
    return VARIANT_FALSE;
}

    pKeyOut = xmlsig->verify(pKey);
if (pKeyOut == NULL) { 
    printf("Invalid signature.\n"); 
    return VARIANT_FALSE;
}
printf("The data referenced in the signature object was verified
return VARIANT_TRUE;
"
);

VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.\n"
                return VARIANT_FALSE;
    }
    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50))) ) {
        printf("Installation of msxml5 is required to run this app.\n"
                return VARIANT_FALSE;
    }
    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;

    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
}

void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n"");
        exit(-1);
    }
    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    if(VARIANT_TRUE == LoadXML(SIG_IN)) {
        if (VARIANT_TRUE != SignXML(HMACSECRET, HMACLENGTH)) {
            printf("exit with failure.\n");
            goto clean;
        }
    }
Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "KeyFromHMACSecretBinaryProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight KeyFromHMACSecretBinaryProj. From the File menu, select New.
7. Click OK.
8. Copy the XML data resource file, and paste it into the text file you just created.
9. Repeat steps 5-8 for the C++ file above (KeyFromHMACSecretBinary.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting Build.
KeyFromHMACSecretBinaryProj.exe from the Build menu.

13. Execute the sample application by selecting **Execute**
    KeyFromHMACSecretBinaryProj.exe from the Build menu.

14. Verify that your output is the same as that listed in the Output topic. You can also verify output has been saved to signature-enveloping-hmac-sha1.xml in the same directory as your project.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
When you build and run the KeyFromHMACSecretBinary.cpp code, you should get the following output.

The data referenced in the signature template was signed successfully.

Resultant signature:

```xml
<?xml version="1.0"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
      <HMACOutputLength>40</HMACOutputLength>
    </SignatureMethod>
    <Reference URI="#object">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
        <DigestValue>gr9ao1nOSq6zgOFaaiCjY6GHxB50=</DigestValue>
      </DigestMethod>
    </Reference>
  </SignedInfo>
  <SignatureValue>h2y7yFQ=</SignatureValue>
  <Object Id="object">Hello, World!</Object>
</Signature>
```

The data referenced in the signature object was verified successfully.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
createKeyFromNode Method

[This feature was first implemented for MSXML 5.0.]

Generates a key object from the DOM node corresponding to the `<ds:KeyInfo>` and one of its descendant elements. The `createKeyFromNode` method is used to acquire the public key for signature verification.

[JScript]

Script Syntax

```javascript
var objKey = objXMLDigitalSignature.createKeyFromNode(node);
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set objKey = objXMLDigitalSignature.createKeyFromNode(node)
```

[C/C++]

C/C++ Syntax Using Smart Pointers

```c
IXMLDSigKeyPtr objKey=IXMLDigitalSignature->createKeyFromNode(node);
```

C/C++ Syntax

```c
HRESULT createKeyFromNode (  
    IXMLDOMNode* node,  
    IXMLDSigKey** objKey);
```

Parameters

`node` [in]

An XML DOM node corresponding to a child element of the `<ds:KeyInfo>` element. For more information, see the Remarks section below.
**objKey [out]**

In C/C++, this is a reference to the resultant key object implementing the IXMLDSigKey interface. In Visual Basic, scripting languages, and C/C++ with smart pointers, this is the key object itself.

**Return Values**

This methods returns the standard CryptoAPI return values, including the following:

**S_OK**
The key object was created successfully.

**E_FAIL**
The key object could not be created. Possible reasons include that the <ds:KeyInfo> node contained insufficient information, or the caller was not authorized.

**E_ACCESSDENIED**
The key object could not be created because the user does not have access to the security item.

**Remarks**

The actions performed by `createKeyFromNode` during key generation depend on the input node. This method can use any of the following types of nodes to create the key.

<table>
<thead>
<tr>
<th>Input node</th>
<th>Actions performed</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">ds:KeyValue</a></td>
<td>Searches for the <a href="">ds:DSAKeyValue</a> or <a href="">ds:RSAKeyValue</a> node.</td>
</tr>
<tr>
<td><a href="">ds:DSAKeyValue</a></td>
<td>Loads the DSA key.</td>
</tr>
<tr>
<td><a href="">ds:RSAKeyValue</a></td>
<td>Loads the RSA key.</td>
</tr>
<tr>
<td><a href="">ds:X509Data</a></td>
<td>Searches for the <a href="">ds:X509Certificate</a> node.</td>
</tr>
<tr>
<td><a href="">ds:X509Certificate</a></td>
<td>Loads the X509 certificate, extracts the embedded key, and</td>
</tr>
<tr>
<td></td>
<td>loads it into the calling application.</td>
</tr>
<tr>
<td><a href="">ds:RetrievalMethod</a></td>
<td>Obtains additional key information based, on the URI and</td>
</tr>
<tr>
<td></td>
<td>the transformation specified.</td>
</tr>
</tbody>
</table>
The resultant key object does not contain any certificate information. Calling IXMLDSigKey::getStore or IXMLDSigKeyEx::getStoreHandle on this key object will result in NULL.

An error occurs when the input node is a <ds:KeyInfo> element containing any X509 certificates, either explicitly (by way of <ds:X509Certificate>), or implicitly (by way of <ds:RetrievalMethod>). In this case, calling IXMLDSigKey::getStore or IXMLDSigKeyEx::getStoreHandle on the resultant key object will return an IStore object or a memory store object containing the certificates.

Example

This example illustrates how to verify a signature on signed data. The example uses the createKeyFromNode method to obtain the signature. We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- [Resource File (signature.dsa.xml)](#)
- [JScript Code (KeyFromNode.js)](#)
- [Visual Basic Code (KeyFromNode.frm)](#)
- [C/C++ Code (KeyFromNode.cpp)](#)
- [Output for the KeyFromNode Example](#)

Applies To

IXMLDigitalSignature | IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

verify Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
The example uses a signature file, which contains internally referenced text data that has been signed. This file has the following characteristics:

- The signed text data is a string, "Hello, World!". This string is held as the content of a `<ds:Object Id="HelloWorld">` element.
- The signature method used is http://www.w3.org/2000/09/xmldsig#dsa-sha1.
- The key information is given in an `<ds:KeyInfo>` element, which will be retrieved by the `createKeyFromNode` method before the signature is verified.

**Signature File (signature.dsa.xml)**

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <ds:Reference URI="#HelloWorld" Type="http://www.w3.org/2000/09/xmldsig#;Object">
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DIGESTValue>BQKNAJjTw98dp7Uvw4oAmJU8LSU=</ds:DIGESTValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>SxbzV35amsuf4mtKgMHps0Fdm6U98fF6vF81CUpmXVd9ULLxeXH5Ew==</ds:SignatureValue>
    <ds:Object Id="HelloWorld" MimeTypetext/plain>Hello, World!</ds:Object>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyValue>
            <DSAKeyValue>
                <P>wLhwVoUYz2txHkKnKUQoN/u1ftCXqw61edboPC1wPyv6EV6FBX10E8vGHHjfqm2ZLm4Gu/j8v
                    sNXyD1i8zGFXmhqO05Me2sUpoFgbUzZpeZ7ZN0YMaFXWzgeKGdg/gb9DrK/USj4BGueB3HAzmFQT8Qu6SM/YBGZKph7JTddYXbodU=
                </P>
                <Q>nkI1onvHNR4tMSvPMc0rhOBHgME=</Q>
            </DSAKeyValue>
        </KeyValue>
    </KeyInfo>
</ds:Signature>
```
<G>QD6OU0Ieg64nvtVbp1Rblw+ehmpPCIjG1Aq+
w67C3PHqRfor9fBeKf6J6Ap111QsBBSacu6S00
LLKhvR2jwsqyAdYMYSuUYoTJZ66pGkMy2U712
1/oKrlkF8MrEZh3esA2JfvF+h396K2Wa/Nv19
s8xTcDhxLepcndAewKhZ5nY=
</G>

<y>rv1G+1/y99y0xGrdaJAN1m7YjJ5/j7famSv6
cHVwtY79WXFFBDlFGNVOKGisRG6BXX00G
4yHL0ndIsywsMYDejB+TuUfvkGcEPGDZErty1F
/xgJ48jhPttrv8Wv3k279+ufB27NpBC/
vw2Rxf3bh22L7V0KNa0Vutwie2g=
</y>

<p>AAAAZQ==</p>

<seed>soBYpnG6yy/B40r2y1zqB2mZr/U==</seed>

</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig
var szResult = "";

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
sigfile = "signature.dsa.xml"

if (LoadXML(sigfile)) {
    if (VerifyXML()) {
        alert(szResult);
    }
}

///////// Helper functions: ///////////
function alert(str) { WScript.echo(str); }

///////// Set signature for signing: //////////
function LoadXML(file)
{
    try {
        xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
        xmldsig= new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
    }
    catch (e) {
        alert("Installation of mxsml5 is required to run this app.\n");
        return false;
    }

    xmldoc.async = false;
    xmldoc.preserveWhiteSpace = true;
    xmldoc.validateOnParse = false;

    if (!xmldoc.load(file)) {
        alert("Can't load " + file + "\n");
        return false;
    }

    szResult += "\nInput signature element:\n\n" + xmldoc.xml;

    xmldoc.setProperty("SelectionNamespaces", DSIGNS);
    xmldsig.signature = xmldoc.selectSingleNode(".//ds:Signature");

    return true;
}
function VerifyXML()
{
    if (!xmldsig.signature) {
        alert("Invalid signature.\n");
        return false;
    }
    var oKeyInfo = xmldoc.selectSingleNode("./ds:KeyInfo/ds:KeyValue");
    var oKey = xmldsig.createKeyFromNode(oKeyInfo);
    var oVerifiedKey = xmldsig.verify(oKey);
    if (oVerifiedKey == null) {
        alert("Signature Verification failed.\n");
    } else {
        szResult += ("\nThe signature has been verified.\n");
    }
    return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature resource file, and paste it into a text file. Save the file as signature.dsa.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as KeyFromNode.js, in the same directory where you saved signature.dsa.xml.
4. From a command prompt, navigate to this directory, then type "cscript KeyFromNode.js".

   Note Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe or cscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source:
KeyFromNode.frm

Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim DSIGNS As String
Dim DispStr, file

Private Function LoadXML()
    ' Read the input xml file and display the content in the Text1.
    DispStr = ""
    If xmldoc.xml = "" Then
        If file = "" Then
            DispStr = "invalid input xml file name"
            LoadXML = False
            Exit Function
        End If
    End If
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    If xmldoc.Load(Path) = False Then
        DispStr = DispStr + vbNewLine + _
        "can't load " + Path + vbNewLine + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS

    DispStr = "Input signature element:" + vbNewLine + vbNewLine _
    + xmldoc.xml + vbNewLine
    LoadXML = True
End Function

Private Function VerifyXML()
    If xmldoc.xml = "" Then
        DispStr = "signature doc is empty."
        VerifyXML = False
        Exit Function
    End If
End Function
DispStr = DispStr + vbNewLine
Set signature = xmldoc.selectSingleNode(".//ds:Signature")
Set xmldsig.signature = signature

Set oKeyInfo = xmldoc.selectSingleNode(".//ds:KeyInfo/ds:KeyValue")
Set oPubKey = xmldsig.createKeyFromNode(oKeyInfo)
Set oVerifiedKey = xmldsig.verify(oPubKey)

If oVerifiedKey Is Nothing Then
    DispStr = "verify failed." + vbNewLine
End If

DispStr = DispStr + "The signature has been verified." _
    + vbNewLine + vbNewLine

VerifyXML = True
End Function

Private Sub Form_Load()
    ' Resize the text box control to the size of the form
    Text1.Top = 100
    Text1.Left = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750

    DSIGN = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
    file = "signature.dsa.xml"

    If LoadXML = False Then
        Text1.Text = DispStr
        Exit Sub
    End If

    If VerifyXML = False Then
        Text1.Text = DispStr
        Exit Sub
    End If

    Text1.Text = DispStr
End Sub

Private Sub Form_Resize()
    ' Resize the text box control to the size of the form
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub
Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature resource file, and paste it into a text file. Save the file as signature.dsa.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as KeyFromNode.vbp to the same directory where you saved signature.dsa.xml. Name the form file KeyFromNode.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, and then check the box for Microsoft XML, v5.0.
5. Place a TextBox control onto the form. It should be named as Text1 (the default name). Click it to select it. From the Properties window, modify the following properties for Text1: set the ScrollBars property to "2 – Vertical" and the MultiLine Property to True.
6. Copy the Visual Basic code listing above, and paste it into the code editor to replace whatever code is already there.
7. Execute the code by selecting Start from the Run menu.
8. Verify that your output is the same as that listed in the Output. Resize the form as needed to view the output.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: KeyFromNode.cpp

```c
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

////////////////////////////////////////////////////////////////////////////////
// Load a signature document into dom and assign it to the xmldsig object.
////////////////////////////////////////////////////////////////////////////////
VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    printf("\nInput signature element: %s\n", (LPCSTR)xmldoc->xml);
    xmldoc->setProperty("SelectionNamespaces", DSIGNS);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode(".//ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}
```

////////////////////////////////////////////////////////////////////////////////
// Verify signature with a key created from createKeyFromNode method
VARIANT_BOOL VerifyXML() {
    IXMLDOMNodePtr pKeyInfo;
    IXMLDSigKeyPtr pKey, pKeyOut;
    pKeyInfo = xmldoc->selectSingleNode("./ds:KeyInfo/ds:KeyValue");
    if (pKeyInfo == NULL) {
        printf("Invalid <ds:KeyInfo/KeyVal
ue>
        return VARIANT_FALSE;
    }
    pKey = xmldsig->createKeyFromNode(pKeyInfo);
    if (pKey == NULL) {
        printf("Invalid key from <ds:KeyInfo>
        return VARIANT_FALSE;
    }
    pKeyOut = xmldsig->verify(pKey);
    if (pKeyOut == NULL) {
        printf("Invalid signature.
        return VARIANT_FALSE;
    }
    printf("The signature has been verified.
    return VARIANT_TRUE;
}

// Helper function to create dom and dsig objects:
//
VARIANT_BOOL initObjects() {
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.
        return VARIANT_FALSE;
    }
    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)))) {
        printf("Installation of msxml5 is required to run this app.
        return VARIANT_FALSE;
    }
    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    objectsAreInitialized = VARIANT_TRUE;
    return VARIANT_TRUE;
}
// Helper function to release dom and dsig objects:
void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
}

// Main function:
void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    if(VARIANT_TRUE == LoadXML("signature.dsa.xml")) {
        if (VARIANT_TRUE != VerifyXML()) {
            printf("exit with failure.\n");
        }
    }

    cleanObjects();
    CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "KeyFromNodeProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the
New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight KeyFromNodeProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature.dsa.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature resource file, and paste it into the text file you just created.
9. Repeat steps 5-8 for the C++ file above (KeyFromNode.cpp).
10. Build the sample by selecting Build KeyFromNodeProj.exe from the Build menu.
11. Execute the sample application by selecting !Execute KeyFromNodeProj.exe from the Build menu.
12. Verify that your output is the same as that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
When you build and run the **KeyFromNode** code, you should get the following output.

**Input signature element:**

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#HelloWorld" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>BQKNAjTjTw98dp7Uvw4oAmJU8LSU==</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>SxbzV35amsuf4mtKgMHps0Fdm6U98fF6vF81CUpmXVd9ULLXeXHSEw==</ds:SignatureValue>
  <ds:Object Id="HelloWorld" MimeType="text/plain">Hello, World!</ds:Object>
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KeyValue>
      <PKAKeyValue>
        <P>wLHwVoUYz2txHkKnKuQoN/u1fCXqw61ed\n        pC1wpv6E6FBXlO8v8vghHjzfzm2ZLm4Guj8v
        sNXyDTi8zGFXmhqO0O5Me2sUs0gbUzZpeZ7ZN
        0YmaFxWzgeKDg/gb9DrK/USj4BgueB3HAzmF
        QT58u6SM/YBGZKph7JTddYXbodU=
      </P>
      <Q>nk1onvHNR4tMSvPMc0rhOBHgME==</Q>
      <G>QD6OU0Ieg64ntVvb1Rp1w+ehmpPCIjG1Aq+
        w67C3PHqRfor9fBeK6J6Ap111QsBBSau6S00
        LLkHvR2jwsgyAdYMYSuUYoTJZ66pGkMy2U712
        1/oKr1kF8MrEZ3esA2JfvF+h396K2w8/Nv19
        s8xTcdnLePCnAewKhz5nY=
      </G>
      <Y>rv1GX+qY99y0xGrdaJAN1m7yjJ5/7famSv6
        cHVv779WVFBd1FGNVQKrG6sG68XX00G
        4yHLOndiswMYDejB+TuufvKcErgDZERTPy1F
        /xgJ48jhtttrwv8Wy3k279+ufB27NpBC/
        vW2Xf3bh22L7V0KNaOvUTwie2g=
      </Y>
    </KeyValue>
  </KeyInfo>
</ds:Signature>
```
The signature has been verified.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
createSAXProxy Method

[This feature was first implemented for MSXML 5.0.]

Creates and returns a SAX proxy object that can be used by the setReferenceData method to assign a SAX stream as the new data source of the signature object or template.

[JScript]

Script Syntax

```javascript
var SAXProxy = objIXMLDigitalSignature.createSAXProxy();
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set SAXProxy = objIXMLDigitalSignature.createSAXProxy();
```

[C/C++]

C/C++ Syntax Using Smart Pointers

```c
IUnknownPtr SAXProxy = objIXMLDigitalSignature.createSAXProxy();
```

C/C++ Syntax

```c
HRESULT createSAXProxy(IUnknown ** SAXProxy)
```

Parameters

**SAXProxy**

This is a pointer to the resultant SAX proxy object in C/C++ applications not using smart pointers. In Visual Basic, scripting languages, and C/C++ with smart pointers, this is SAX proxy object itself.
Return Values

S_OK
The method succeeded, and the SAX proxy object returned.

E_FAIL
The method failed, and the SAXProxy parameter returns NULL.

Remarks

To assign a SAX stream as the new data source of a signature object, you must first use this method to create a SAX proxy object. Then, connect this object to a SAX reader from which the stream originates.

The SAX proxy object supports the ISAXContentHandler, ISAXLexicalHandler, ISAXErrorHandler, IVBSAXContentHandler, IVBSAXLexicalHandler, IVBSAXErrorHandler interfaces and parses the data stream into XML documents or nodes. It serves as a content handler, error handler, and lexical handler between the SAX reader and the signature object.

The following is a general workflow for signing or verifying XML data from a SAX stream:

1. Create a SAX proxy object using this method.
2. Call the setReferenceData method to set this proxy object as the new data source of a signature object.
3. Create a SAX reader.
4. Set the SAX proxy object as the content handler of the SAX reader,
5. Set the SAX proxy as the lexical handler to process comments, if desired.
6. Set the SAX proxy as the error handler to catch exceptions, if desired.
7. Call the parse method or the parseURL method on the SAX reader to start feeding data into the signature object through the SAX proxy.

Do not reuse the SAX proxy object in a single signing and/or verification. To work with multiple SAX sources, create a proxy object for each source. Any errors, including warnings, will interrupt the SAX stream and prevent the data from being signed or verified.

A SAX proxy object can be used to improve the performance of an application when it is used to sign and/or verify data from a remote location multiple times.
In this case, you can first download the data to a local store, then access the local copy of the data with the help of a SAX proxy object. This saves repeated round trips for data retrieval across the network.

**Example**

This example illustrates how to use the createSAXProxy method and the setReferenceData method to use an XML data file in the local file system as the new data source of a signature template. It uses two resource files: a signature template (signature_template.rsa.xml) and a data file to be signed (test.xml). The signature template contains simple text data ("Hello, World!") embedded as the content of a <ds:object> element. This simple text data is signed as usual. In contrast, test.xml is signed with the help of a SAX proxy object.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- [Resource Files (signature_template.rsa.xml and test.xml)](#)
- [JScript Code (createSAXProxy.js)](#)
- [Visual Basic Code (createSAXProxy.frm)](#)
- [C/C++ Code (createSAXProxy.cpp)](#)
- [Output for the createSAXProxy Example](#)

**Applies To**

[IXMLDigitalSignature](#) | [IXMLDigitalSignatureEx](#)

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[setReferenceData Method](#) | [ISAXXMLReader Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Resource: signature_template.rsa.xml and test.xml

This example uses two resource files: a signature template and an XML file. The XML file will be read by a SAX reader and fed as a SAX stream into the signature object, via a SAX proxy object.

Signature_template.rsa.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig"/>
    <ds:Reference URI="#objData">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="objData">Hello, world!</ds:Object>
</ds:Signature>
```

test.xml

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<root peanut="bar.com">
  <PARTS>
    <PART MAKER="bar.com">
      <PARTNO>12345</PARTNO>
      <DESCRIPTION>VIP - Very Important Part</DESCRIPTION>
    </PART>
    <PART>
      <PARTNO>5678</PARTNO>
      <DESCRIPTION>LIP - Less Important Part</DESCRIPTION>
    </PART>
  </PARTS>
</root>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig, saxProxy, saxReader, fileSys;
var infile, dataID, dataSrc;

NOKEYINFO = 0;
KEYVALUE = 1;
CERTIFICATES = 2;
PURGE = 4;

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
PROV_RSA_FULL = 1;
//// Change this key container name to your own if necessary.
RSA_KEY = "MyRSAFullKeys";
csp = PROV_RSA_FULL;
key = RSA_KEY;
infile = "signature_template.rsa.xml";
dataID = "#objData";

if (InitXML())
{
  if (LoadXML(infile)) {
    /// Sign the data embedded in the signature template.
    dataSrc = null;
    alert("Signing data referenced in signature...\n");
    SignXML(dataID, dataSrc);

    /// Sign the XML data of test.xml with the help of
    /// a SAX proxy object.
    dataSrc = fileSys.GetAbsolutePathName("test.xml");
    alert("\nSigning "+dataSrc+" fed through SAX Proxy...\n");
    LoadXML(infile);
    SignXML(dataID, dataSrc);
  }
}

/////////// Helper functions: ///////////
function alert(str) { WScript.echo(str); }

/////////// Set signature for signing. /////////
function InitXML()
{
  try {
    xmldoc = new ActiveXObject("Msxml2.DOMDOcument.5.0");
    xmldsig= new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
    fileSys = new ActiveXObject("Scripting.FileSystemObject");
try {
    alert("Installation of mxsm15 is required to run this app.
    return false;
} catch (e) {
    alert("Installation of mxsm15 is required to run this app.
    return false;
}

xmlDoc.async = false;
xmlDoc.preserveWhiteSpace = true;
xmlDoc.validateOnParse = false;
xmlDoc.resolveExternals = false;

return true;
}

function LoadXML(file)
{
    if (xmlDoc == null) {
        alert("must instantiate xml dom\n");
        return false;
    }

    if (!xmlDoc.load(file)) {
        alert("Can't load "+ file + "\n");
        return false;
    }

    xmlDoc.setProperty("SelectionNamespaces", DSignNS);
    xmlDsigs.signature = xmlDoc.selectSingleNode("./ds:Signature");

    return true;
}

function SignXML(dataID, srcUrl)
{
    var saxReader = null;

    if (!xmlDsigs.signature)
    {
        alert("Invalid signature template\n");
        return false;
    }

    if (!dataID || dataID=="") {
        alert("Invalid reference ID\n");
        return false;
    }

    // For a non-empty srcUrl, create a SAXReader object and
    // connect it to a SAXProxy object as its contentHandler,
    // so that the data fed through a SAX stream from srcUrl
    // is used in signing.
if (srcUrl && srcUrl != "") {
    try {
        saxReader = new ActiveXObject("Msxml2.SAXXMLReader.5.0");
    } catch (e) {
        alert("Invalid SAXReader: " + e.description + "\n");
        return false;
    }
    saxProxy = xmldsig.createSAXProxy();
    if (saxProxy == null) {
        alert("Failed to create SAX proxy.\n");
        return false;
    }
    try {
        saxReader.contentHandler = saxProxy;
        xmldsig.setReferenceData(dataID, saxProxy);
        saxReader.parseURL(srcUrl);
    } catch (e) {
        alert("Using SAX Proxy: " + e.description + "\n");
    }
}

var oKey = xmldsig.createKeyFromCSP(csp, "", key, 0);
if (!oKey) {
    alert("Invalid key.\n");
    return false;
}

///// Sign the data and purge key info from the resultant signature file, because the key info will be the same for both data sets and omitting them makes the output shorter and more clear.
var oSignedKey = xmldsig.sign(oKey, PURGE);
if (!oSignedKey) {
    alert("sign failed.\n");
    return false;
}

alert("Signing was successful.\n" + "Resultant signature:\n" + xmldoc.xml + "\n");
return true;

Try It!
1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.

2. Copy the XML signature file from Resource Files (signature_template.rsa.xml and test.xml), and paste it into a text file. Save the file as signature_template.rsa.xml.

3. Copy the XML data file from Resource Files, and paste it into a text file. Save the file as test.xml, in the same directory where you saved signature_template.rsa.xml.

4. Copy the JScript listing above, and paste it into a text file. Save the file as createSAXProxy.js, in the same directory where you saved signature_template.rsa.xml and test.xml.

5. From a command prompt, navigate to this directory, then type "cscript createSAXProxy.js".

   **Note** Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe or cscript.exe), if it is not already installed.

6. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim saxReader As New SAXXMLReader50
Dim saxProxy, outfile, provType, keyContainer

Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"
Const PROV_RSA_FULL = 1
' Change this key container name to your own if necessary.
Const RSA_KEY = "MyRSAFullKeys"
Const INFILE = "signature_template.rsa.xml"
Const dataID = "#objData"

Private Function LoadXML(ByVal file As String)
    ' Read the input XML file.
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.resolveExternals = False
    If xmldoc.Load(Path) = False Then
        WriteLine "Can't load " + Path
        WriteLine "Reason: " + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    ' Set the signature property on xmldsig.
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    Dim sig
    Set sig = xmldoc.selectSingleNode("//ds:Signature")
    If sig Is Nothing Then
        WriteLine "failed select <Signature>"
        LoadXML = False
        Exit Function
    End If
    Set xmldsig.signature = sig

    LoadXML = True
End Function
Private Function SignXML(ByVal dataID As String, ByVal srcUrl As String)
    If xmldsig.signature Is Nothing Then
        WriteLine "Invalid signature template."
        SignXML = False
        Exit Function
    End If

    If keyContainer = "" Then
        WriteLine "Invalid key container."
        SignXML = False
        Exit Function
    End If

    ' Get the Key from the default csp provider.
    Set oKey = xmldsig.createKeyFromCSP(provType, ",", keyContainer,
    If oKey Is Nothing Then
        WriteLine "Invalid key"
        SignXML = False
        Exit Function
    End If

    If srcUrl <> "" Then
        ' Reassign the data source.
        Set saxProxy = xmldsig.createSAXProxy
        If saxProxy Is Nothing Then
            SignXML = False
            Exit Function
        End If
        Set saxReader.contentHandler = saxProxy
        xmldsig.setReferenceData dataID, saxProxy
        saxReader.parseURL srcUrl
    End If

    CONTINUE:
    Set oSignedKey = xmldsig.Sign(oKey, PURGE)
    If oSignedKey Is Nothing Then
        WriteLine "sign failed."
        SignXML = False
        Exit Function
    End If

    WriteLine "Signing was succesful."
    WriteLine "Resultant signature: " + vbNewLine
    WriteLine xmlldoc.xml

    SignXML = True

    End Function

Private Sub Form_Load()
'Resize the text box control to the size of the form
Text1.Top = 100
Text1.Left = 100
Text1.Width = Form1.Width - 350
Text1.Height = Form1.Height - 750
Dim newSrc As String
provType = PROV_RSA_FULL
keyContainer = RSA_KEY
writeClear
refID = "#objData"

If LoadXML(INFILE) = True Then
    WriteLine "Signing data referenced in the signature..."
    SignXML refID, ""
    newSrc = App.Path + "\test.xml"
    WriteLine vbNewLine
    WriteLine "Signing " + newSrc + " fed through SAX Proxy..."
    LoadXML INFILE
    SignXML refID, newSrc
End If

End Sub

Private Sub Form_Resize()
    'Resize the text box control to the size of the form
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

' Helper function. Make sure that a TextBox control named Text1
' is present in the project.
Private Function WriteLine(ByVal str As String)
    Text1.Text = Text1.Text + str + vbNewLine
End Function

' Helper function:
Private Function writeClear()
    Text1.Text = ""
End Function

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature file resource file, and paste it into a text file. Save the file as signature_template.rsa.xml.
3. Copy the XML data file resource file, and paste it into a text file. Save the file as test.xml, in the same directory where you saved signature_template.rsa.xml.

4. Create a Standard EXE project in Visual Basic. Save the empty project as createSAXProxy.vbp to the same directory where you saved signature_template.rsa.xml and test.xml. Name the form file createSAXProxy.frm.

5. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v5.0.

6. Copy the Visual Basic code listing above, and paste it into the code editor to replace whatever content is already there.

7. Place a TextBox control onto the form. It should be named as Text1 (the default name). Click it to select it in the Properties window, and then modify the following properties for Text1: set the ScrollBars property to "2 – Vertical" and the MultiLine Property to True.

8. Execute the code by selecting Start from the Run menu.

9. Verify that your output is similar to that listed in the Output topic. You can resize the form as needed to more easily see the output.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: createSAXProxy.cpp

#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"
// Change this key container name to your own if necessary.
#define RSA_KEY "MyRSAFullKeys"
#define INFILE "signature_template.rsa.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmlsig = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);
    // Set the signature property to a <ds:Signature> DOM node.
    xmlsig->signature = xmldoc->selectSingleNode(".//ds:Signature");

    if (xmlsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL SignXML(unsigned short * refID, unsigned short* newSrc)
{
    HRESULT hr;
    ISAXXMLReaderPtr saxReader=NULL;
    if (xmlsig->signature == NULL) {

printf("Invalid signature template\n");
return VARIANT_FALSE;
}

if (refID==NULL || refID == L""") {
printf("Invalid reference ID\n");
return VARIANT_FALSE;
}

if (newSrc!=NULL && newSrc != L""") {
hr = saxReader.CreateInstance(__uuidof(SAXXMLReader50));
if (FAILED(hr)) {
    printf("Invalid SAXReader object\n");
    return VARIANT_FALSE;
}
IUnknownPtr saxProxy = xmldsig->createSAXProxy();
if (saxProxy == NULL) {
    printf("Failed to create SAX proxy\n");
    return VARIANT_FALSE;
}
ISAXContentHandlerPtr ch = NULL;
hr = saxProxy.QueryInterface(__uuidof(ISAXContentHandler),(ISAXContentHandler**)&ch);
if (FAILED(hr)) {
    printf("Can't QI ISAXContentHandler\n");
    return VARIANT_FALSE;
}
hr = saxReader->putContentHandler(ch);
if (FAILED(hr)) {
    printf("Can't set saxProxy\n");
    return VARIANT_FALSE;
}
hr = xmldsig->setReferenceData(refID, saxProxy);
if (FAILED(hr)) {
    printf("Can't set reference data\n");
    return VARIANT_FALSE;
}
hr = saxReader->parseURL(newSrc);
if (FAILED(hr)) {
    printf("Failed to parseURL\n");
    return VARIANT_FALSE;
}
} // At this point, saxReader and ch will be out of scope.

IXMLDSigKeyPtr pKey = xmldsig->createKeyFromCSP(
    PROV_RSA_FULL, "", RSA_KEY, 0);
if (pKey==NULL) {
printf("Invalid key\n");
return VARIANT_FALSE;
}

IXMLDSigKeyPtr pKeyOut = xmldsig->sign(pKey, PURGE);
if (NULL == pKeyOut) {
    printf("sign failed.\n");
    return VARIANT_FALSE;
}

printf("Signing was successful.\n");
printf("Resultant signature:\n\n");
printf((LPCSTR)xmldoc->xml);
printf("\n");
return VARIANT_TRUE;
}

VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)) )
        printf("Installation of msxml5 is required to run this app.\n"
        return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)) )) {
        printf("Installation of msxml5 is required to run this app.\n"
        return VARIANT_FALSE;
    }
    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
}

void main()
{
    if ( CoInitialize(NULL) == E_FAIL) {
        printf("Installation of msxml5 is required to run this app.\n"
        return VARIANT_FALSE;
    }

    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)) )
        printf("Installation of msxml5 is required to run this app.\n"
        return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)) )) {
        printf("Installation of msxml5 is required to run this app.\n"
        return VARIANT_FALSE;
    }
    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}
printf("can't initialize COM Lib\n");
exit(-1);
}

if (!initObjects()) {
    cleanObjects();
    exit(-1);
}

if (VARIANT_TRUE == LoadXML(INFILE)) {
    printf("Signing data referenced in signature...\n");
    SignXML(L"#objData", L"");
}

if (VARIANT_TRUE == LoadXML(INFILE)) {
    printf("Signing test.xml fed through SAX proxy...\n");
    SignXML(L"#objData", L"test.xml");
}

cleanObjects();
CoUninitialize();

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "createSAXProxyProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight createSAXProxyProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_template.rsa.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature template resource file, and paste it into the text file you just created.
9. Repeat steps 5-8 for the XML data file, test.xml.
Note You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

10. Repeat steps 5-8 for the C++ file above (createSAXProxy.cpp).
11. From the Project menu, click Settings..., then click the Link tab.
12. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
13. Build the sample by selecting Build createSAXProxyProj.exe from the Build menu.
14. Execute the sample application by selecting !Execute createSAXProxyProj.exe from the Build menu.
15. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

When you build and run the `createSAXProxy` code, you should get output similar to the following:

Signing data referenced in signature...

Signing was successful.
Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#objData">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>K6t8fcEaWzaF7ybZnuoS+7E204U="</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    BOuhBNVcwOda4gDJDFMfJx6is1MRLmbMGpWYRMMybu1AG0cBSgmlaUA==
  </ds:SignatureValue>
  <ds:Object Id="objData">Hello, world!</ds:Object>
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#"/>
</ds:Signature>
```

Signing C:\XML\SDK\msxml5\examples\dsig\createSAXProxy\setReferenceData from a SAX Proxy...

Signing was successful.
Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  </ds:SignedInfo>
</ds:Signature>
```
<ds:Reference URI="#objData">
  <ds:DigestMethod
    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <ds:DigestValue>i885dg/WbBNacx4kXpnUk7abIiQ=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
  l8xG/E22pN2skpI3hp3HWsZNa0IiKL0GzCNKcXF9RqE0mAV7diGTXg==
</ds:SignatureValue>
<ds:Object Id="objData">Hello, world!</ds:Object>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
</KeyInfo>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
getCSPHandle Method

[This feature was first implemented for MSXML 5.0.]

Retrieves the handle to the cryptographic service provider associated with this key object. The resultant CSP handle can be used in the appropriate CryptoAPI functions to perform other Cryptographic operations.

[C/C++]

C/C++ Syntax

HRESULT getCSPHandle (ULONG_Ptr *hCSProv);

C/C++ Syntax Using Smart Pointers

ULONG_Ptr hCSProv = objIXMLDSigKeyEx.getCSPHandle();

Parameters

hCSProv
Handle to the cryptographic service provider associated with this key object.

Return Values

S_OK
The handle was retrieved successfully.
E_FAIL
Failed to retrieve the handle and the resultant hCSProv parameter is NULL.

Remarks

When the key object is created using the IXMLDigitalSignature::createKeyFromCSP method, you can call this method
to get the handle to the underlying cryptographic service provider. You can examine and use other cryptographic information associated with this key by calling the appropriate CryptoAPI functions.

For keys created by `IXMDDigitalSignature::createKeyFromHMACSecret` or `IXMDDigitalSignatureEx::createKeyFromHMACSecretBinary`, no CSP handles will be created and the resultant `hCSProv` parameter will be NULL.

**Applies To**

`IXMDDigKeyEx`

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [️] in the upper-left corner of the page.

**See Also**

`createKeyFromCSP Method`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
getStoreHandle Method

[This feature was first implemented for MSXML 5.0.]

Retrieves a handle to the certificate store from which the certificate can be retrieved and validated. The method is applicable to C/C++ application only.

C/C++ Syntax

HRESULT getStoreHandle (void** hCertStore);

Parameters

hCertStore
Handle to a certificate store created using CryptoAPI functions, such as CertOpenSystemStore or CertOpenStore.

Return Values

S_OK
Certificate store handle was obtained without error.
E_FAIL
Failed to obtain the certificate store handle. The hCertStore parameter returns NULL.

Remarks

When you are done using the certificate store handle, call CertCloseStore to free the store object.

Applies To

IXMLDigitalSignatureEx

Versioning
MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.

See Also

IXM LDSignatureEx Interface | sign Method | verify Method
getVerifyingCertificate Method

[This feature was first implemented for MSXML 5.0.]

Retrieves the certificate associated with this key used to verify a signature.

[JScript]

Script Syntax

```javascript
var oCertificate = IXMLDSigKeyEx.getVerifyingCertificate();
```

[Visual Basic]

Visual Basic Syntax

```vbnet
Set oCertificate = IXMLDSigKeyEx.getVerifyingCertificate
```

[C/C++]

C/C++ Syntax

```c
HRESULT getVerifyingCertificate (IUnknown** oCertificate);
```

C/C++ Syntax Using Smart Pointers

```c
IUnknownPtr oCertificate = IXMLDSigKeyEx.getVerifyingCertificate();
```

Parameters

`oCertificate`

In C/C++ applications not using smart pointer classes, this is a reference to the certificate associated with the key. In all other cases, this is the certificate of the key.

Return Values
S_OK
   The method call was successful.
E_FAIL
   The method call failed.

Remarks

You must call this method after a signature is verified.

If the signature contains a certificate, this method returns the certificate of the
key. If the signature contains no certificate, the resultant oCertificate
parameter is NULL.

Example

The following example demonstrates how to validate the certificate used in the
signature verification. A signed signature document containing a certificate is
first verified before the certificate is validated by examining the root certificate.
In any production code, a more sophisticated validation procedure should be
followed.

The signed signature document is used as the input for the code examples in
C/C++, JScript, and Visual Basic. The output is the same in each language.

- Resource Files
- JScript Code (getVerifyingCert.js)
- Visual Basic Code (getVerifyingCert.frm)
- C/C++ Code (getVerifyingCert.cpp)
- Output for the getVerifyingCertificate Method Example

Applies To

IXMLDigitalSignature, IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later
To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2000/09/xmldsig#http-signature"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#Object">
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>IFILa99v5A6V1v37H+OiqQ0bmgYRaprUzlV3dPbx88YzDdi9ooyMcsGvpBshBbHQomso+/VcTXUA==</ds:SignatureValue>
    <ds:Object Id="obj" MimeTypetext/plain">Hello, World!</ds:Object>
    <ds:KeyInfo>
        <ds:RSAKeyValue>
            <Modulus>972fKQQrs46EC07zhbsIw362fa4qkq2HRcmLaz/E4qYUDfycwofHO0R4/p/z7C+PSzuuOxeTKL02NdBzfww==</Modulus>
            <Exponent>AQAB</Exponent>
        </ds:RSAKeyValue>
    </ds:KeyInfo>
    <ds:X509Data>
        <ds:X509Certificate>`MIIEjCCBK2gAwIBAgIKJMLTcgABAAACADAJBgcqhkj00AQDmüGMSwIyJKoZIhvCmFhRpmZvQHNLJ3V3wLs8mLnvBtELMAKAGA1EBhMCVVMxMkJAMBgNVBAgTBVN0YXRIcwYDVQQHEwRDaXR5MRywFAYDVQQKEwITWV1cmUgTXk5W5IsMRUwEwYDVQQLEwEZXhBZw5OiFxQfJABqgNBAMTDVNI3YVZU15WG1sQ0EwHhcNMD1wMzE4MjEZNTiwNCwNMDMwMjE0NTIwWjCBEfMB0GBqgGSiBi3DfEJYQbWVABx1jb21wYX55LmNvbTELMAKGA1UEVVMxJAMdBgNVBAgTBVN0XQR1MQ0wCwYDVQQHEwRDaXR5MRIwEAYDVQQKEw1teWNVbXhEDA0BqgNVBAgTB215Z3VdXAXcAzJABgNgNBAMTDAmjLmwwDQyJKoZIhvNAQEBqADgSwAwAMGdnyaEK70BhAju8x27CMNo2uKpEmxOJX12/s/X0KmAFA38gsKHxteEpPqf8+wvj0s72qMXyviNnjXq38CAwEAAoCAAVlwggLtMA4GA1udDwEB/wQAwIGWDATBgNVHSUDE5kTBgEFBQCDAzAdBqGNHQ4EFgUHuQFwfwP0HH10pWgDawmpaw8wgdQGA1UiDwSBzDCBUCw08rgfoE/tz+9LFxNhnusGChgZ6kgZswgZxIzAhBdkgkhikg9w0BCQEwFD1uZm9Adxj1bx14/DbwuyY29tMQswCQYDVQQGEwJEUxEOAwGAIUECBMFU3RhdGU6LgNVBAcTdHkxJAUbgNVBAoTDVlN3VyZSBNeSBYbWxwFTATBgNVBAsTEDRcGFyD1bnQgWDEwA1UEAxMN2uVfjXltx1YbwxQDYIQfY767Mch3rJKiPpwQpsDCBqSYDVR0fB1GMP1IgmTqBMhkpodHRw0i8va3VydGRpbmctMDEuMcVkbW9uZC5jb3JwLm1pY3Jvc3M9mdC5jb20v</ds:X509Certificate>
    </ds:X509Data>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig;

NOKEYINFO = 0;
KEYVALUE = 1;
CERTIFICATES = 2;
PURGE = 4;

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#';";
outfile = "signature_signed.rsa.cert.xml";

if (InitXML())
{
  wki = CERTIFICATES|KEYVALUE|PURGE;
  if (LoadXML(outfile)) {
    alert("Verifying " + outfile+"...\n");
    VerifyXML(wki);
  }
}

///////// Helper functions: ///////////
function alert(str) { WScript.echo(str); }

///////// Set signature for signing. //////////
function InitXML()
{
  try {
    xmldoc = new ActiveXObject("Msxml2.DOMDocumen", ...]
    xmldsig = new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
  }
  catch(e) {
    alert("Installation of mxsml5 is required to run this app.\n");
    return false;
  }

  xmldoc.async = false;
  xmldoc.preserveWhiteSpace = true;
  xmldoc.validateOnParse = false;
  xmldoc.resolveExternals = false;

  return true;
}

function LoadXML(file) {
if (xmlDoc == null) {
    alert("must instantiate xml dom\n");
    return false;
}

if (!xmlDoc.load(file)) {
    alert("Can't load "+ file + "\n");
    return false;
}
xmlDoc.setProperty("SelectionNamespaces", DSIGNS);
xmlDsig.signature = xmlDoc.selectSingleNode("./ds:Signature");
return true;
}

function VerifyXML(fwWriteKeyInfo)
{
    if (!xmlDsig.signature) {
        alert("Invalid signature.\n");
        return false;
    }

    var xpath = "";
    switch ( fwWriteKeyInfo&CERTIFICATES ) {
    case CERTIFICATES:
        xpath = ".//ds:KeyInfo/ds:X509Data";
        break;
    case KEYVALUE:
        xpath = ".//ds:KeyInfo/ds:KeyValue";
        break;
    }

    var oKeyInfo = xmlDoc.selectSingleNode(xpath);
    if (!oKeyInfo) {
        alert("Invalid <KeyInfo> element.\n");
        return false;
    }

    var oKey = xmlDsig.createKeyFromNode(oKeyInfo);
    if (!oKey) {
        alert("Failed to create key from <KeyInfo>\n");
        return false;
    }

    var oVerifiedKey = xmlDsig.verify(oKey);
    if (oVerifiedKey == null) {
        alert("Signature not verified.\n");
    }
return false;
}

alert("\nSignature verified on the data.\n");

if ((fwWriteKeyInfo & CERTIFICATES) == CERTIFICATES)
{
    if (IsCertificateValid(oVerifiedKey))
        alert("Certificate used is valid.\n");
}

return true;

function IsCertificateValid(oKey)
{
    if (!oKey)
    {
        alert("invalid key object.\n");
        return false;
    }

    // Retrieve the certificate from the key that
    // has been used to verify a signature.
    var oCert = oKey.getVerifyingCertificate();
    if (!oCert)
    {
        alert("invalid verifying certificate\n");
        return false;
    }

    // Need to walk up through the certificate's trust chain.
    var oChain = new ActiveXObject("CAPICOM.Chain.2");
    if (!oChain)
    {
        alert("invalid chain object.\n");
        return false;
    }

    // Build a trust chain starting from oCert
    status = oChain.build(oCert);
    if (!status)
    {
        alert("broken trust chain. error="+status+"\n");
        return false;
    }

    // Walk through the trust chain
alert("Examining certificate chain:
");
for (i=1; i<=oChain.Certificates.count; i++)
{
    alert(" Certificate No. "+i+":
");
    oCert = oChain.Certificates.item(i);
    alert(" subject: "+oCert.SubjectName);
    alert(" issuer: "+oCert.IssuerName);
    alert("\n");
}
// Examine the root certificate in the chain
oCert = oChain.Certificates.item(oChain.Certificates.count);
alert("Display the Root Certificate:
");
alert(" subject: " + oCert.SubjectName);
alert(" issuer: " + oCert.IssuerName);
alert("\n");

return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the signature_signed.rsa.cert.xml and paste it into a text file. Save the file as signature_signed.rsa.cert.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as getVerifyingCert.js, in the same directory where you saved signature_signed.rsa.cert.xml.
4. From a command prompt, navigate to this directory, then type "cscript getVerifyingCert.js".

   Note   Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run wscript.exe or cscript.exe), if it is not already installed.

5. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source:
getVerifyingCert.frm

' Be sure to select "CAPICOM v2.0 Type Library"
' and "Microsoft XML v5.0" options from the
' "Project->References..." menu item in
' Visual Studio.
'
Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim dataObj As IXMLDOMNode
Dim infile, provType, keyContainer

Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"

Private Function WriteLine(ByVal str As String)
    Text1.Text = Text1.Text + str + vbNewLine
End Function
Private Function writeClear()
    Text1.Text = ""
End Function

Private Function LoadXML(ByVal file As String)
    ' Read input xml file and display the content in the Text1.
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.resolveExternals = False
    If xmldoc.Load(Path) = False Then
        WriteLine "Can't load " + Path
        WriteLine "Reason: " + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    Set xmldsig.signature = xmldoc.selectSingleNode(".//ds:Signature"
    LoadXML = True
End Function

Private Sub Form_Load()
'Resize the text box to the size of the form
Text1.Top = 100
Text1.Left = 100
Text1.Width = Form1.Width - 350
Text1.Height = Form1.Height - 750
infile = "signature_signed.rsa.cert.xml"
writeClear
wki = CERTIFICATES

If LoadXML(infile) = True Then
    WriteLine "Verifying " & infile & "..."
    VerifyXML wki
End If

End Sub

Private Sub Form_Resize()
'Resize the text box to the size of the form
Text1.Width = Form1.Width - 350
Text1.Height = Form1.Height - 750
End Sub

Private Sub VerifyXML(fWriteKeyInfo)
    Dim xpath As String
    Dim oKeyNode As IXMLDOMNode
    Dim oKey As IXMLDSigKey
    Dim oKeyOut As IXMLDSigKey

    If xmldsig.signature Is Nothing Then
        WriteLine "Invalid signature "
        Exit Sub
    End If
    xpath = ""
    If fWriteKeyInfo = CERTIFICATES Then
        xpath = ".//ds:KeyInfo/ds:X509Data"
    Else
        If fWriteKeyInfo = KEYVALUE Then
            xpath = ".//ds:KeyInfo/ds:KeyValue"
        End If
    End If
    Set oKeyNode = xmldoc.selectSingleNode(xpath)
    If oKeyNode Is Nothing Then
        WriteLine "Invalid key from signature doc."
        Exit Sub
    End If
    Set oKey = xmldsig.createKeyFromNode(oKeyNode)
If oKey Is Nothing Then
    WriteLine "Failed to create key from node."
    Exit Sub
End If

Set oKeyOut = xmldsig.verify(oKey)
If oKeyOut Is Nothing Then
    WriteLine "Signature not verified."
    Exit Sub
End If

WriteLine vbCrLf
WriteLine "Signature verified on the data"
WriteLine vbCrLf

If fWriteKeyInfo = CERTIFICATES Then
    If IsCertificateValid(oKeyOut) = True Then
        WriteLine "Certificate used is valid."
    End If
End If
End Sub

Function IsCertificateValid(ByVal oKey As IXMLDSigKey) As Boolean
    Dim oCert As ICertificate
    Dim oChain As New Chain
    Dim status As Boolean

    If oKey Is Nothing Then
        WriteLine "Invalid key object."
        IsCertificateValid = False
        Return
    End If

    ' Retrieve the certificate from the key that has been used to verify a signature.
    Set oCert = oKey.getVerifyingCertificate
    If oCert Is Nothing Then
        WriteLine "Invalid verifying certificate"
        IsCertificateValid = False
        Return
    End If

    ' Build a trust chain starting from oCert.
    status = oChain.Build(oCert)
    If status = False Then
        WriteLine "Broken trust chain. Error = " & status
        IsCertificateValid = False
        Exit Function
    End If
' Walk through the trust chain.
WriteLine "Examining certificate chain:
" i = 1
For Each oCert In oChain.CERTIFICATES
   WriteLine " Certificate No. " & i & ":";
   WriteLine " subjecyt: " & oCert.SubjectName
   WriteLine " issuer: " & oCert.IssuerName
   WriteLine vbNewLine
   i = i + 1
Next

' Examine the root certificate in the chain.
Set oCert = oChain.CERTIFICATES.Item(oChain.CERTIFICATES.Count)
WriteLine "Display the Root Certificate:"
WriteLine " subject: " & oCert.SubjectName
WriteLine " issuer: " & oCert.IssuerName
WriteLine vbNewLine

IsCertificateValid = True
End Function

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template from Resource Files and paste it into a text file. Save the file as signature_signed.rsa.cert.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as getVerifyingCert.vbp to the same directory where you saved signature_signed.rsa.cert.xml. Name the form file getVerifyingCert.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the boxes for Microsoft XML, v5.0 and CAPICOM v2.0 Type Library.
5. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named "Text1". Select the Text1 control and and in the Properties window, modify the following settings: select "2 - Vertical" as the value for the MultiLine property and set the value of the ScrollBars property to True.
6. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor to replace whatever code is already there.
7. Execute the code by selecting Start from the Run menu.
8. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: getVerifyingCert.cpp

#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

#include <capicom.dll>
using namespace CAPICOM;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define INFILE "signature_signed.rsa.cert.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    _bstr_t ns = DSIGNS;
    xmldoc->setProperty("SelectionNamespaces", ns);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode(".//ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL IsCertificateValid(IXMLDSigKeyPtr pKey)
{
if (pKey == NULL) {
    printf("invalid key object.\n");
    return VARIANT_FALSE;
}

// Retrieve the certificate from the verifying key.
ICertificatePtr pCert=NULL;
pCert = pKey->getVerifyingCertificate();
if (pCert == NULL) {
    printf("Can't get verifying certificate\n");
    return VARIANT_FALSE;
}

// Build and walk through the certificate's trust chain.
IChain2Ptr pChain = NULL;
HRESULT hr = pChain.CreateInstance(__uuidof(Chain));
if (FAILED(hr)) {
    printf("Can't instantiate a trust chain.\n");
    return VARIANT_FALSE;
}
if (VARIANT_FALSE == pChain->Build(pCert)) {
    printf("broken trust chain.\n");
    return VARIANT_FALSE;
}

printf("Examining certificate chain:\n");
long count = pChain->Certificates->Count;
for (long i=1; i<=count; i++) {
    printf(" Certificate No. %d:\n", i);
    if (pCert) {
        pCert.Release();
pCert=NULL;
    }
    pCert = pChain->Certificates->Item[i];
    if (pCert == NULL) {
        printf("Can't get the %d-th certificate.\n", i);
        return VARIANT_FALSE;
    }
    printf(" subject: %s\n",(LPSTR)pCert->SubjectName);
    printf(" issuer: %s\n",(LPSTR)pCert->IssuerName);
}

// Examine the root certificate in the chain.
if (pCert) {
    pCert.Release();
pCert=NULL;
}

pCert = pChain->Certificates->Item[count];
if (pCert == NULL) {
    printf("Can't get root certificate.\n");
    return VARIANT_FALSE;
}
printf("Display the Root certificate:\n");
printf(" subject: %s\n", (LPSTR)pCert->SubjectName);
printf(" issuer: %s\n\n", (LPSTR)pCert->IssuerName);

if (pCert) pCert.Release();
if (pChain) pChain.Release();

return VARIANT_TRUE;
}

VARIANT_BOOL VerifyXML(XMLSIG_WRITEKEYINFO fWriteKeyInfo)
{
    IXMLDOMNodePtr pKeyInfo, pNode;
    IXMLDSigKeyPtr pKey, pKeyOut;

    if (xmldsig->signature == NULL) {
        printf("Invalid signature.\n");
        return VARIANT_FALSE;
    }

    switch (fWriteKeyInfo & CERTIFICATES) {
    case CERTIFICATES:
        pKeyInfo = xmldoc->selectSingleNode(".//ds:KeyInfo/ds:X509Data");
        break;
    case KEYVALUE:
        pKeyInfo = xmldoc->selectSingleNode(".//ds:KeyInfo/ds:KeyValue");
        break;
    }

    if (pKeyInfo == NULL) {
        printf("Invalid <ds:KeyInfo>\n");
        return VARIANT_FALSE;
    }

    pKey = xmldsig->createKeyFromNode(pKeyInfo);
    if (pKey == NULL) {
        printf("Invalid key from <ds:KeyInfo>\n");
        return VARIANT_FALSE;
    }

    pKeyOut = xmldsig->verify(pKey);
    if (pKeyOut == NULL) {

\printf("Invalid signature.\n")
return VARIANT_FALSE;
}

\printf("Signature verified on the data.\n\n")
if ( (fwriteKeyInfo & CERTIFICATES) == CERTIFICATES) {
    if (IsCertificateValid(pKeyOut)) {
        \printf("Certificate used is valid.\n")
    }
}

return VARIANT_TRUE;
}

VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        \printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50))) )
        \printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    xmldoc->async = VARIANT_FALSE;
xmldoc->validateOnParse = VARIANT_FALSE;
xmldoc->preserveWhiteSpace = VARIANT_TRUE;
xmldoc->resolveExternals = VARIANT_FALSE;
objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xml_doc) xmldoc.Release();
    if (xml_dsig) xml_dsig.Release();
}

void main()
{

    if (CoInitialize(NULL) == E_FAIL) {
        \printf("can't initialize COM Lib\n")
        exit(-1);
    }

    if (!initObjects()) {

cleanObjects();
exit(-1);
}

printf("Verifying %s\n\n", INFILE);
if (VARIANT_TRUE == LoadXML(INFILE)) {
    VerifyXML(CERTIFICATES);
}
cleanObjects();
CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "getVerifyingCertProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight getVerifyingCertProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_signed.rsa.cert.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature template file from Resource Files (signature_signed.rsa.cert.xml), and paste it into the text file you just created.

Note You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 5-8 for the C++ file above (getVerifyingCert.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting **Build getVerifyingCertProj.exe** from the **Build** menu.
13. Execute the sample application by selecting **!Execute getVerifyingCertProj.exe** from the **Build** menu.
14. Verify that your output is similar to that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

The `getVerifyingCert` example should produce output similar to the following.

Verifying signature_signed.rsa.cert.xml...

Signature verified on the data.

Examining certificate chain:

  Certificate No. 1:

    subject: CN=me, OU=mygroup, O=mycompany, L=City, S=State, C=US,
    issuer:   CN=SecureMyXmlCA, OU=Department X, O=Secure My Xml, L=City,

Display the Root Certificate:

    subject: CN=me, OU=mygroup, O=mycompany, L=City, S=State, C=US,
    issuer:   CN=SecureMyXmlCA, OU=Department X, O=Secure My Xml, L=City,

Certificate used is valid.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
getVerifyingCertificateContext Method

[This feature was first implemented for MSXML 5.0.]

Retrieves the certificate context associated with this key used to verify a signature.

[C/C++]

C/C++ Syntax

HRESULT getVerifyingCertificateContext (void** pCertContext);

C/C++ Syntax Using Smart Pointers

void *pCertContext = oIXMLDSigEx.getVerifyingCertificateContext();

Parameters

pCertContext

In C/C++ application without using smart pointers, this is the reference to the context of the certificate of this key that has been used to verify a signature. In the other case, this is the context of the certificate of this key that has been used to verify a signature.

Return Values

S_OK
    Operation was successful.

E_FAIL
    Operation failed.

Remarks
You can call this method after a signature is verified.

To use the resultant certificate context, type cast it to the PCCERT_CONTEXT type. After using the certificate context, it is your responsibility to free the context by calling CertFreeCertificateContext function of CryptoAPI.

**Example**

The following example demonstrates how to validate the certificate of the key used in the signature verification. A signed signature document containing a certificate is first verified before the certificate is validated by building its chain. If the trust chain is built without errors, the certificate is deemed valid. In any production code, more sophisticated validation criteria might be necessary.

- The resource file is a signed signature document as listed in Resource Files *(signature_signed.rsa.cert.xml).*
- **C/C++ Code (getVerifyingCert.cpp)**
- **Output for the getVerifyingCertificate Method Example**

**Applies To**

[IXMLDigitalSignatureEx](#)

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**

**getVerifyingCertificate Method**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: getVerifyingCertContext.cpp

#import <msxml5.dll>
using namespace MSXML2;

#include <stdio.h>
#include <wincrypt.h>

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define INFILE "signature_signed.rsa.cert.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    // Load signature document from file:
    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

// Test if the certificate of a key (pKey) is valid.
In this simple example, the certificate is verified if a trust chain can be built up without errors. In a production application, more elaborate verification criteria may be necessary. For more information, see the CryptoAPI documentation.

```c++
VARIANT_BOOL IsCertificateValid(IXMLDSigKeyExPtr pKey)
{
    if (pKey == NULL) {
        printf("invalid key object.\n");
        return VARIANT_FALSE;
    }

    // Retrieve the certificate from the verifying key.
    PCCERT_CONTEXT pCert=NULL;
    pCert = (PCCERT_CONTEXT)pKey->getVerifyingCertificateContext();
    if (pCert == NULL) {
        printf("Can't get verifying certificate context\n");
        return VARIANT_FALSE;
    }

    // Use CryptoAPI to verify the certificate.
    CERT_CHAIN_ENGINE_CONFIG chainConfig;
    PCCERT_CHAIN_CONTEXT pChainContext;
    CERT_CHAIN_PARA chainPara;

    HCERTCHAINENGINE hChainEngine=NULL; // Use the default

    // Initialize chainPara:
    chainPara.cbSize = sizeof(CERT_CHAIN_PARA);
    chainPara-RequestedUsage.dwType = USAGE_MATCH_TYPE_AND;
    chainPara-RequestedUsage.Usage.cUsageIdentifier = 0;
    chainPara-RequestedUsage.Usage.rgpszUsageIdentifier = NULL;

    // Initialize chainConfig:
    chainConfig.cbSize = sizeof(CERT_CHAIN_ENGINE_CONFIG);
    chainConfig.hRestrictedRoot = NULL;
    chainConfig.hRestrictedTrust = NULL;
    chainConfig.hRestrictedOther = NULL;
    chainConfig.cAdditionalStore = 0;
    chainConfig.rghAdditionalStore = NULL;
    chainConfig.dwFlags = CERT_CHAIN_CACHE_END_CER
    chainConfig.dwUrlRetrievalTimeout = 0;
    chainConfig.MaximumCachedCertificates = 0;
    chainConfig.CycleDetectionModulus = 0;

    // Creates a certificate chain engine to build a certificate trust
    if( !CertCreateCertificateChainEngine(&chainConfig, &hChainEngine)
    {
        printf("Can't create certificate chain engine. Error Code= %d\n");
    }
```
return VARIANT_FALSE;
}

// Build a certificate chain from the chain engine.
if( !CertGetCertificateChain(hChainEngine, // chain eng
pCert, // Pointer t
NULL, // Use the d
NULL, // Search no
&chainPara, // Use AND l
CERT_CHAIN_REVOCATION_CHECK_END_CER
NULL, // Currently
&pChainContext)) // Return a
{
    printf("Failed to complete the trust chain of the certificate.
    printf("Error code = %d\n", GetLastError());
CertFreeCertificateChain(pChainContext);
CertFreeCertificateChainEngine(hChainEngine);
CertFreeCertificateContext(pCert);
return VARIANT_FALSE;
}

// Verification successful.
CertFreeCertificateChain(pChainContext);
CertFreeCertificateChainEngine(hChainEngine);
CertFreeCertificateContext(pCert);

return VARIANT_TRUE;
}
VARIANT_BOOL VerifyXML(XMLDSIG_WRITEKEYINFO fWriteKeyInfo)
{
    IXMLDOMNodePtr pKeyInfo, pNode;
    IXMLDSigKeyPtr pKey, pKeyOut;

    if (xmldsig->signature == NULL) {
        printf("Invalid signature.\n");
        return VARIANT_FALSE;
    }

    switch (fWriteKeyInfo & CERTIFICATES)
    {
    case CERTIFICATES:
        pKeyInfo = xmldoc->selectSingleNode(".//ds:KeyInfo/ds:X509Data");
        break;
    case KEYVALUE:
        pKeyInfo = xmldoc->selectSingleNode(".//ds:KeyInfo/ds:KeyValue");
        break;
    }
if (pKeyInfo == NULL) {
    printf("Invalid <ds:KeyInfo>\n");
    return VARIANT_FALSE;
}

pKey = xmldsig->createKeyFromNode(pKeyInfo);
if (pKey == NULL) {
    printf("Invalid key from <ds:KeyInfo>\n");
    return VARIANT_FALSE;
}

pKeyOut = xmldsig->verify(pKey);
if (pKeyOut == NULL) {
    printf("Invalid signature.\n");
    return VARIANT_FALSE;
}

printf("Signature verified on the data.\n\n");

if ((fwriteKeyInfo & CERTIFICATES) == CERTIFICATES) {
    if (IsCertificateValid(pKeyOut)) {
        printf("certificate used is valid.\n");
    }
}

return VARIANT_TRUE;

VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50))))
        printf("Installation of msxml5 is required to run this app.\n");
        return VARIANT_FALSE;
}

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)))) {
        printf("Installation of msxml5 is required to run this app.\n");
        return VARIANT_FALSE;
    }
    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}
Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "getVerifyingCertContextProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight getVerifyingCertContextProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_signed.rsa.cert.xml" in the File name text box.
7. Click **OK**.
8. Copy the XML signature template file from **Resource:** `signature_signed.rsa.cert.xml`, and paste it into the text file you just created.

   **Note** You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 5-8 for the C++ file above (`getVerifyingCertContext.cpp`).
10. From the **Project** menu, click **Settings...**, then click the **Link** tab.
11. In **Object/library** modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click **OK**.
12. Build the sample by selecting **Build getVerifyingCertContextProj.exe** from the **Build** menu.
13. Execute the sample application by selecting **!Execute getVerifyingCertContextProj.exe** from the **Build** menu.
14. Verify that your output is the same as that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

The output for the `getVerifyingCertContext` example is as follows:

Verifying signature.

Signature verified on the data.

certificate used is valid.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
setDebugOutput Method

[This feature was first implemented for MSXML 5.0.]

Sets an IStream object on the IXMLDigitalSignature object to receive debug information generated by the IXMLDigitalSignature object.

C/C++ Syntax

HRESULT setDebugOutput (IStream* debugOutput);

Parameters

debugOutput
   An IStream object to receive debugging information.

Return Values

S_OK
   The debug output stream object was set without error.
E_FAIL
   Failed to set the debug output.

Remarks

The IXMLDigitalSignature object generates debug information. This information consists of digest values and the signature value as the input to calculations. Such information can be helpful for debugging an application when the input data is invalid or improper. For example, if data referenced in the <ds:Reference> element must undergo an XLST transformation before it is digested, an improper style sheet or template will output an incorrect digest value. This scenario might also apply in other cases involving canonicalization and other transformations. The debug output could help you determine the source of such problems.
To catch the debug output, you need to create an IStream class that implements at least the Commit and Write methods. Other methods on the IStream interface can simply return E_NOTIMPL.

The IXMLDigitalSignature object invokes IStream::Commit with the grfCommitFlags parameter set to zero (0) before it begins to calculate a digest value. The object calls this method with the grfCommitFlags parameter set to one (1) after it finishes calculating the digest value. If you decide to direct the debug output to a file for each digest calculation, you can implement the IStream::Commit method in such a way that a new file is created if grfCommitFlags is 0 and the file is closed when grfCommitFlags is 1. If you want all the output to go to the standard output (stdout), you can use, for example, printf() to mark off the beginning of a digest calculation when grfCommitFlags is 0 and to mark off the end of the digest calculation when grfCommitFlags is 1. This is illustrated in the example below.

The IXMLDigitalSignature object invokes IStream::Write when it has the debug information ready. You can direct the output to a file if a file is opened, or you can print the debug output to stdout as shown in the example below. For each digest, the IXMLDigitalSignature object writes out the debug information in chunks. The maximum size of a chunk is 1024 bytes.

The IXMLDigitalSignature object also outputs debug information when calculating a signature value. In signature signing, the object writes out the <ds:SignedInfo> element to the debug output target after digesting all the data referred to in the <ds:Reference> elements. In signature verification, it writes out the <ds:SignedInfo> element to the debug output target before calculating all the data digests. For an illustration, see the example below.

**Example**

The example below demonstrates how to catch the debug output from the IXMLDigitalSignature object and direct it to the standard output (stdout).

A DBStream class implementing the IStream interface is created (DBStream.cpp and DBStream.h). An instance of this class is passed to the IXMLDigitalSignature::setDebugOutput method call. All the debug data are displayed using printf(). You could also implement this class to save the output to one or more files, or to store the debug information in another XML
DOM object. The same DBStream object is made available for both sign and verify in the main program.

The input signature template (signature.template.xml) contains an embedded reference (<ds:Reference URI="#obj" ...>) and an external one (<ds:Reference URI="http://www.example.com">). This means that two digests are encountered for each sign or verify, and the DBStream object records debug data for four digest values.

The example outputs a signed document persisted to signature.document.xml, as well as output displayed in the application's console.

- Resource File (signature.template.xml)
- C/C++ Code (setDebugOutput.cpp)
- Output for the getVerifyingCertificate Method Example

Applies To

IXMLDigitalSignature

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☰ in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#">
      <ds:DIGESTValue></ds:DIGESTValue>
    </ds:Reference>
    <ds:Reference URI="http://www.example.com">
      <ds:DIGESTValue></ds:DIGESTValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="obj" Mime-Type="text/plain">Hello, World!</ds:Object>
  <ds:KeyInfo/>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: setDebugOutput.cpp, DBStream.h, and DBStream.cpp

The source code consists of the following three files.

<table>
<thead>
<tr>
<th>Source file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setDebugOutput.cpp</td>
<td>Main program that instantiates a DBStream object, sets it on the IXMLDigitalSignature object to receive debug output, and calls sign/verify on IXMLDigitalSignature.</td>
</tr>
<tr>
<td>DBStream.h</td>
<td>Header file for the DBStream class.</td>
</tr>
<tr>
<td>DBStream.cpp</td>
<td>Class implementation of DBStream.</td>
</tr>
</tbody>
</table>

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "setDebugOutputProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight setDebugOutputProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature.template.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature template file from Resource: signature.template.xml, and paste it into the text file you just created.

Note You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).
9. Repeat steps 5-8 for the `setDebugOutput.cpp` C++ file.
10. Repeat steps 5-8 for the `DBStream.h` header file.
11. Repeat steps 5-8 for the `setDebugOutput.cpp` C++ file.
12. From the **Project** menu, click **Settings...**, then click the **Link** tab.
13. In **Object/library** modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click **OK**.
14. Build the sample by selecting **Build setDebugOutputProj.exe** from the **Build** menu.
15. Execute the sample application by selecting **!Execute setDebugOutputProj.exe** from the **Build** menu.
16. Verify that you get output similar to that listed in the **Output** topic at the end of this example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
This is the main program for the setDebugOutput Method example. This code performs the following steps:

1. Instantiates DBStream.
2. Calls setDebugOutput.
3. Calls on an XML DOM object that has loaded the signature.template.xml file.

```c++
#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;
#include "DBStream.h"

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'"
#define SIG_TEMP "signature.template.xml"
#define SIG_DOC "signature.document.xml"
// Change this key container name to your own if necessary.
#define RSA_KEY "myRSAFullKeys"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDDigitalSignatureExPtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;
DBStream * dbStream = NULL;

// Load a signature document into dom and assign it to the xmldsig object.
VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);
}
```
// Set the signature property to a <ds:Signature> DOM node.
xmlsig->signature = xmldoc->selectSingleNode("./ds:Signature");

if (xmlsig->signature == NULL) {
    printf("Failed to set the signature property.\n");
    return VARIANT_FALSE;
}

printf("\n%s has been loaded into an XML DOM\n", (LPCSTR)sigFile);
return VARIANT_TRUE;

/////////////////////////////////////////////////////////////////////

// Verify signature with a key embedded in the signature document.

VARIANT_BOOL VerifyXML()
{
    IXMLDSigKeyPtr  pKeyOut;

    pKeyOut = xmlsig->verify(NULL);
    if (pKeyOut == NULL) {
        printf("Invalid signature.\n");
        return VARIANT_FALSE;
    }

    printf("\nSignature verified.\n");
    return VARIANT_TRUE;
}

VARIANT_BOOL SignXML(XMLDSIG_WRITEKEYINFO fwWriteKeyInfo)
{
    if (xmlsig->signature == NULL) {
        printf("Invalid signature template\n");
        return VARIANT_FALSE;
    }

    IXMLDSigKeyPtr pKey = xmlsig->createKeyFromCSP(
                      PROV_RSA_FULL, ",", RSA_KEY, 0);
    if (pKey == NULL) {
        printf("Invalid key\n");
        return VARIANT_FALSE;
    }

    IXMLDSigKeyPtr pKeyOut = xmlsig->sign(pKey, fwWriteKeyInfo);
    if (NULL == pKeyOut) {
        printf("sign failed.\n");
        return VARIANT_FALSE;
    }
printf("\nData signed.\n\n");

return VARIANT_TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Helper function to create dom and dsig objects:
//
VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.\n"
            return VARIANT_FALSE;
    }

    if (FAILED(xmlDom.CreateInstance(__uuidof(DOMDocument50)) ))
        
            return VARIANT_FALSE;
    }

    xmlDom->async = VARIANT_FALSE;
    xmlDom->validateOnParse = VARIANT_FALSE;
    xmlDom->preserveWhiteSpace = VARIANT_TRUE;

    dbStream = new DBStream();
    if (dbStream==NULL)
        printf("Can't instantiate DBStream\n");
        return VARIANT_FALSE;
    }

    HRESULT hr = xmldsig->setDebugOutput(dbStream);
    if (FAILED(hr))
        
            return VARIANT_FALSE;
    }

    objectsAreInitialized = VARIANT_TRUE;
    return VARIANT_TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Helper function to release dom and dsig objects:
//
void cleanObjects()
{
    if (xmlDom) xmlDom.Release();
    if (xmldsig) xmldsig.Release();
    if (dbStream) dbStream->Release();
}
// Main function:
//
void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    if(VARIANT_TRUE == LoadXML(SIG_TEMP)) {
        if (VARIANT_TRUE != SignXML(KEYVALUE)) {
            printf("exit with failure.\n");
        }
        xmldoc->save(SIG_DOC);
    }

    if(VARIANT_TRUE == LoadXML(SIG_DOC)) {
        if (VARIANT_TRUE != VerifyXML()) {
            printf("exit with failure.\n");
        }
    }

    cleanObjects();
    CoUninitialize();
}
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: DBStream.h

The following is the header file for the DBStream class used in the setDebugOutput project.

```cpp
#include <windows.h>
#import <msxml5.dll>
using namespace MSXML2;

paralleled
// An IStream object to receive debug info:
class DBStream : public IStream
{
protected:
    int _nCount;
    DWORD _nState;

private:
    long _refcount;
    IXMLDOMDocument3Ptr _xmldbg;
    BSTR *bstrBuf;

public:

paralleled
// Constructor and destructor:
//
DBStream(void);
~DBStream();

paralleled
// IUnknown interface:
//
HRESULT STDMETHODCALLTYPE QueryInterface(REFIID riid,
    void ** ppvObject);
ULONG STDMETHODCALLTYPE AddRef( void);
ULONG STDMETHODCALLTYPE DBStream::Release( void);

paralleled
// IStream Interface:
//
HRESULT STDMETHODCALLTYPE Commit(DWORD grfCommitFlags);
HRESULT STDMETHODCALLTYPE Write(void const* pv,
```
ULONG cb,
ULONG * pcbWritten);

HRESULT STDMETHODCALLTYPE Read(void * pv,
ULONG cb,
ULONG * pcbRead);

HRESULT STDMETHODCALLTYPE Seek(LARGE_INTEGER dlibMove,
DWORD dwOrigin,
ULARGE_INTEGER * plibNewPosition);

HRESULT STDMETHODCALLTYPE SetSize(ULARGE_INTEGER libNewSize);

HRESULT STDMETHODCALLTYPE CopyTo(IStream * pstm,
ULARGE_INTEGER cb,
ULARGE_INTEGER * pcbRead,
ULARGE_INTEGER * pcbWritten);

HRESULT STDMETHODCALLTYPE Revert(void);

HRESULT STDMETHODCALLTYPE LockRegion( ULARGE_INTEGER libOffset,
ULARGE_INTEGER cb,
DWORD dwLockType);

HRESULT STDMETHODCALLTYPE UnlockRegion(ULARGE_INTEGER libOffset,
ULARGE_INTEGER cb,
DWORD dwLockType);

HRESULT STDMETHODCALLTYPE Stat(STATSTG * pstatstg,
DWORD grfStatFlag);

HRESULT STDMETHODCALLTYPE Clone(IStream ** ppstm);

private:
    void STDMETHODCALLTYPE dprintf( char * format, ...);
};
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
The following is the class implementation of DBStream used in the setDebugOutput project.

```c
#include <stdio.h>
#include "DBStream.h"

/********************************************
// Constructor and destructor:
//
DBStream::DBStream(void)
{
   _nCount = 1;
   _refcount = 1;
}

DBStream::~DBStream()
{
}

/********************************************
// IUnknown interface:
//
HRESULT STDMETHODCALLTYPE DBStream::QueryInterface(
   REFIID riid,
   void ** ppvObject)
{
   if (riid == IID_IUnknown)
   {
      *ppvObject = static_cast<IUnknown*>(this);
   }
   else if (riid == IID_IStream)
   {
      *ppvObject = static_cast<IStream*>(this);
   }
   else
   {
      *ppvObject = NULL;
      return E_NOINTERFACE;
   }
   AddRef();
   return S_OK;
```
ULONG STDMETHODCALLTYPE DBStream::AddRef( void)
{
    return InterlockedIncrement(&_refcount);
}

ULONG STDMETHODCALLTYPE DBStream::Release( void)
{
    if (InterlockedDecrement(&_refcount) == 0)
    {
        delete this;
        return 0;
    }
    return _refcount;
}

///////////////////////////////////////////////////////////
// IStream Interface:
///////////////////////////////////////////////////////////
HRESULT STDMETHODCALLTYPE DBStream::Commit(DWORD grfCommitFlags)
{
    _nState = grfCommitFlags;
    switch(_nState)
    {
    case 0:// Start digest computation
        dprintf("\n== Catching digest \%d. ===\n",_nCount);
        break;
    case 1:// End digest computation
        dprintf("=== Digest \%d caught.===\n\n",_nCount);
        _nCount++;
        break;
    }
    return S_OK;
}

HRESULT STDMETHODCALLTYPE DBStream::Write(
    void const* pv,
    ULONG cb,
    ULONG * pcbWritten)
{
    dprintf(" ===Begin DBStream::Write (cb=%d)\n", cb);
    dprintf("%s\n", pv);
    dprintf(" ===End DBStream::Write (cb=%d)\n", cb);
    return S_OK;
}

HRESULT STDMETHODCALLTYPE DBStream::Read(
void * pv,
ULONG cb,
ULONG * pcbRead)
{
  return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::Seek(
  LARGE_INTEGER dlibMove,
  DWORD dwOrigin,
  ULARGE_INTEGER * plibNewPosition)
{
  return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::SetSize(
  ULARGE_INTEGER libNewSize)
{
  return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::CopyTo(
  IStream * pstm,
  ULARGE_INTEGER cb,
  ULARGE_INTEGER * pcbRead,
  ULARGE_INTEGER * pcbWritten)
{
  return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::Revert(void)
{
  return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::LockRegion(
  ULARGE_INTEGER libOffset,
  ULARGE_INTEGER cb,
  DWORD dwLockType)
{
  return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::UnlockRegion(
  ULARGE_INTEGER libOffset,
  ULARGE_INTEGER cb,
  DWORD dwLockType)
{
  return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE DBStream::Stat(
    STATSTG * pstatstg,
    DWORD grfStatFlag)
{
    return E_NOTIMPL;
}

HRESULT STDMETHODCALLTYPE DBStream::Clone(
    IStream ** ppstm)
{
    return E_NOTIMPL;
}

// This helper method outputs a message to both stdout and
// debug-output in Visual Studio.
void STDMETHODCALLTYPE DBStream::dprintf( char * format, ...)
{
    static char buf[1024];
    va_list args;
    va_start( args, format );
    vsprintf( buf, format, args );
    va_end( args);
    OutputDebugStringA( buf);
    printf("%s", buf);
}
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

The output for the **setDebugOutput** example consists of debug output and non-debug output. The latter is shown in bold letters. The debug output from each **write** is displayed between the following lines:

```
===Begin DBStream::Write (cb=***)
===End DBStream::Write (cb=***)
```

Notice that `<ds:SignedInfo>` is thrown out in a different order from the debug output in `sign` and `verify`.

The following is the application output.

*signature.template.xml has been loaded into an XML DOM*

```
=== Catching digest 1. ===
===Begin DBStream::Write (cb=113)
===End DBStream::Write (cb=113)
=== Digest 1 caught.===

=== Catching digest 2. ===
===Begin DBStream::Write (cb=339)
<!-- HTML -->
</HEAD>
<TITLE>Example Web Page</TITLE>
</HEAD>
<body>
<p>You have reached this web page by typing "example.com", "example.net", or "example.org" into your web browser.</p>
<p>These domain names are reserved for use in documentation and are available for registration.</p>
</BODY>
</HTML>

===End DBStream::Write (cb=339)
=== Digest 2 caught.===
```
Data signed.

signature.document.xml has been loaded into an XML DOM
You have reached this web page by typing &quot;example.com&quot;, &quot;example.net&quot;, or &quot;example.org&quot; into your web browser.

These domain names are reserved for use in documentation and are available for registration.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
setReferenceData Method

[This feature was first implemented for MSXML 5.0.]

Assigns a new data source to a specified <ds:Reference> element within a signature object. You can use this method to sign or verify different data from the default data source.

[JScript]

Script Syntax

objXMLDigitalSignature.setReferenceData(uri, pSRC);

[Visual Basic]

Visual Basic Syntax

objXMLDigitalSignature.setReferenceData(uri, pSRC)

[C/C++]

C/C++ Syntax Using Smart Pointers

objXMLDigitalSignature->setReferenceData(uri, pSRC);

C/C++ Syntax

HRESULT setReferenceData(
    BSTR uri,
    IUnknown* pSRC);

Parameters

uri [in]

Identifies the <ds:Reference> element. The string value of this parameter is matched against the URI attribute value of a <ds:Reference> element
within an XML digital signature object. To locate a <ds:Reference> element without the URI attribute, set this parameter to "##null".

**pSrc** [in]
The new data source for use with the specified <ds:Reference> element for future verification or signing. The new data can be represented as an IXMLDOMNode, an IStream object, or a SAX proxy object created by the createSAXProxy Method method. For a SAX proxy, the caller must start the SAX reader by calling its parse or parseURL methods or performing other equivalent operations before sign or verify can proceed.

**Return Values**

**S_OK**
The value returned if successful.

**E_FAILURE**
The value returned if the method call failed.

**Remarks**

This method does not resolve or check the content of the uri parameter. It uses uri only to locate appropriate <ds:Reference> elements in the signature or template. To specify the <ds:Reference> element with NULL URI attribute value (that is, the <ds:Reference> element without the URI attribute), use "##null" in the uri parameter. This method does not modify the <ds:Signature> XML fragment.

You must set the signature property before calling this method.

The new data source (pSrc) object can be any of the following types:

- An IXMLDOMNode object, which must be either a document or an element node, is treated as an XML document.
- An IStream object is treated as a binary stream containing either XML or non-XML data, unless the stream undergoes a transformation that outputs XML. In the latter case, the IStream object is treated as an XML document.
- A SAX proxy object, which must be connected to a SAX Reader that accepts a SAX stream, is treated as an XML document.

Although <ds:Signature> can have multiple <ds:Reference> elements with the
same URI attribute value, you cannot override their data source using the setReferenceData method. Calling setReferenceData on such a signature object to override the data source will result in an error.

The setReferenceData method initiates the computation of the digest value either immediately or some time later. Therefore, once set, the new data source should not be modified or destroyed before you call sign or verify. Otherwise, the result is undefined.

**Example**

This example uses a enveloping signature template that contains two <ds:Object> elements. Each contains a simple text string. One object (obj2) is referenced in the <ds:Reference> element and the other is not. By default, signing is performed on the referenced data only. However, you can use the setReferenceData method to specify different data to be signed and/or verified.

The example uses an enveloping signature template, and outputs a signed signature file.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource File (signature_template.setRef.rsa.xml)
- JScript Code (setRef.js)
- Visual Basic Code (setRef.frm)
- C/C++ Code (setRef.cpp)
- Output for the setRef Example

**Applies To**

IXMLDigitalSignature | IXMLDigitalSignatureEx

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

createSAXProxy Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Resource:
signature_template.setRef.rsa.xml

<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj2" Type="http://www.w3.org/2000/09/xmldsig#">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue />
  <ds:Object Id="obj1" MimeType="text/plain">Hello, World!</ds:Object>
  <ds:Object Id="obj2">Hi, everyone!</ds:Object>
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
var xmldoc, xmldsig, infile, outfile;
var szResult = "";

NOKEYINFO = 0;
KEYVALUE = 1;
CERTIFICATES = 2;
PURGE = 4;

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#';";
PROV_RSA_FULL = 1;
//Change this key container name to your own if necessary.
RSA_KEY = "MyRSAFullKeys";
csp = PROV_RSA_FULL;
key = RSA_KEY;
infile = "signature_template.setRef.rsa.xml";
outfile1 = "signature.setRef.rsa.obj1.xml";
outfile2 = "signature.setRef.rsa.obj2.xml";

if (InitXML())
{
    alert("Attempting to sign the object of 'obj1'.\n");
    outfile = outfile1;
    if (LoadXML(infile)) {
        SignXML("obj1");
    }

    if (LoadXML(outfile)) {
        VerifyXML("obj1");
    }

    alert("\n");
    alert("Attempting to sign the object of 'obj2'.\n");
    outfile = outfile2;
    if (LoadXML(infile)) {
        SignXML("obj2");
    }

    if (LoadXML(outfile)) {
        VerifyXML("obj2");
    }
}

///////// Helper functions: ///////////
function alert(str) { WScript.echo(str); }

///////// Set signature for signing. /////////
function InitXML()
{
    try {
        xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
        xmldsig= new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
    }
    catch (e) {
        alert("Installation of mxsml5 is required to run this app.\n");
        return false;
    }

    xmldoc.async = false;
    xmldoc.preserveWhiteSpace = true;
    xmldoc.validateOnParse = false;
    xmldoc.resolveExternals = false;
    return true;
}

function LoadXML(file)
{
    if (xmldoc == null) {
        alert("must instantiate xml dom\n");
        return false;
    }

    if (!xmldoc.load(file)) {
        alert("Can't load " + file + "\n");
        return false;
    }

    xmldoc.setProperty("SelectionNamespaces", DSIGNS);
    xmldsig.signature = xmldoc.selectSingleNode(".//ds:Signature");
    return true;
}

function SignXML(objID)
{
    if (!xmldsig.signature)
    {
        alert("Invalid signature template\n");
        return false;
    }

    var oKey = xmldsig.createKeyFromCSP(csp, ",", key, 0);
    if (!oKey)
    {
        alert("Invalid key.\n");
        return false;
    }
function VerifyXML(objID)
{
    if (!xmldsig.signature) {
        alert("Invalid signature.\n");
        return false;
    }

    var oKeyInfo = xmldoc.selectSingleNode("./ds:KeyInfo/ds:KeyValue");
    if (!oKeyInfo) {
        alert("Invalid <KeyInfo> element.\n");
        return false;
    }

    var oKey = xmldsig.createKeyFromNode(oKeyInfo);
    if (!oKey) {
        alert("Failed to create key from <KeyInfo>\n");
        return false;
    }

    xpath = ".//ds:Object[@Id='"+objID+"']";
    dataObj = xmldoc.selectSingleNode(xpath);
    xmldsig.setReferenceData("#obj2", dataObj);

    var oVerifiedKey = xmldsig.verify(oKey);
    if (oVerifiedKey == null) {
        alert("Signature not verified.\n");
    }
}
alert("Signature verified on the data.\n");
return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template from Resource: (signature_template.setRef.rsa.xml, and paste it into a text editor. Save the file as signature_template.setRef.rsa.xml.
3. Copy the JScript listing above, and paste it into a text editor. Save it into a file as setRef.js, in the same directory where you saved signature_template.setRef.rsa.xml.
4. From a command prompt, navigate to this directory, then type "cscript setRef.js".

Note Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run cscript.exe), if it is not already installed.

5. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source: setRef.frm

Dim xmldoc As New DOMDocument50
Dim xmlsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim dataObj As IXMLDOMNode
Dim outfile
Dim provType, keyContainer

Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
Const PROV_RSA_FULL = 1
' Change this key container name to your own if necessary.
Const RSA_KEY = "MyRSAFullKeys"
Const infile = "signature_template.setRef.rsa.xml"
Const outfile2 = "signature.setRef.rsa.obj2.xml"
Const outfile1 = "signature.setRef.rsa.obj1.xml"

Private Function WriteLine(ByVal str As String)
    Text1.Text = Text1.Text + str + vbNewLine
End Function
Private Function writeClear()
    Text1.Text = ""
End Function

Private Function LoadXML(ByVal file As String)
    ' Read the input XML file and display the content in the text3.
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.resolveExternals = False
    If xmldoc.Load(Path) = False Then
        WriteLine "Can't load " + Path
        WriteLine "Reason: " + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    LoadXML = True
End Function
Private Function SignXML(objID As String)
    If xmldoc.xml = "" Then
        WriteLine "Invalid signature template."
        SignXML = False
        Exit Function
    End If

End If

If keyContainer = "" Then
    WriteLine "Invalid key container."
    SignXML = False
    Exit Function
End If

Set xmlDsSignature = xmlDoc.SelectSingleNode("://ds:Signature")
Set oKey = xmlDsSignature.createKeyFromCSP(provType, "", keyContainer,"
    If oKey Is Nothing Then
        WriteLine "Invalid key"
        SignXML = False
        Exit Function
End If

xpath = "./ds:Object[@Id='" + objID + "]"
Set dataObj = xmlDoc.SelectSingleNode(xpath)
xmlDsSignature.setReferenceData "#obj2", dataObj

Set oSignedKey = xmlDsSignature.Sign(oKey, KEYVALUE)
    If oSignedKey Is Nothing Then
        WriteLine "sign failed."
        SignXML = False
        Exit Function
End If

WriteLine "The specified data was signed successfully."
WriteLine "Resultant signature: " + vbNewLine
WriteLine xmlDoc.xml

output = App.Path + "/" + outfile
xmlDoc.Save (output)
SignXML = True
End Function

Private Function VerifyXML(ByVal objID As String)
    If xmlDoc.xml = "" Then
        WriteLine "Invalid XML signature file."
        VerifyXML = False
        Exit Function
End If

Set signature = xmlDoc.SelectSingleNode("://ds:Signature")
    If signature Is Nothing Then
        WriteLine "Invalid <Signature> element"
        VerifyXML = False
        Exit Function
End If
Set xmlDsSignature = signature
Set oKeyInfo = xmlDoc.selectSingleNode("./ds:KeyInfo/ds:KeyValue")
If oKeyInfo Is Nothing Then
    WriteLine "Invalid <KeyInfo> element."
    VerifyXML = False
    Exit Function
End If
Set oPubKey = xmlsig.createKeyFromNode(oKeyInfo)
If oPubKey Is Nothing Then
    WriteLine "Can't generate public key for verification."
    VerifyXML = False
    Exit Function
End If

xpath = ".//ds:Object[@Id='" + objID + "]"
Set dataObj = xmlDoc.selectSingleNode(xpath)
xmldsig.setReferenceData "#obj2", dataObj

Set oVerifiedKey = xmlsig.Verify(oPubKey)
If oVerifiedKey Is Nothing Then
    WriteLine "Signature not verified."
End If

WriteLine "Signature verified on the data."
VerifyXML = True
End Function

Private Sub Form_Load()
' Set text box to use form to determine its width
' and height when form is loaded
Text1.Left = 100
Text1.Top = 100
Text1.Width = Form1.Width - 350
Text1.Height = Form1.Height - 750

provType = PROV_RSA_FULL
keyContainer = RSA_KEY
writeClear

WriteLine "Attempting to sign the object of 'obj1'." + vbNewLine
outfile = outfile1
If LoadXML(infile) = True Then
    SignXML "obj1"
End If

If LoadXML(outfile) = True Then
    VerifyXML "obj1"
End If
WriteLine vbCrLf
WriteLine "Attempting to sign the object of 'obj2'." + vbCrLf
outfile = outfile2
If LoadXML(infile) = True Then
    SignXML "obj2"
End If

If LoadXML(outfile) = True Then
    VerifyXML "obj2"
End If

End Sub

Private Sub Form_Resize()
    'Set text box to use form in determining its width and height
    'when form is resized
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Try It!

1. Ensure that you have completed all the procedures in *Getting Started with XML Digital Signatures*.

2. Copy the XML signature template from Resource: *signature_template.setRef.rsa.xml*, and paste it into a text editor. Save it to a file named *signature_template.setRef.rsa.xml*.

3. Create a Standard EXE project in Visual Basic. Save the empty project as *setRef.vbp* to the same directory where you saved *signature_template.setRef.rsa.xml*. Name the form file *setRef.frm*.

4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for *Microsoft XML, v5.0*.

5. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named "Text1". Select the Text1 control and and in the Properties window, modify the following settings: select "2 - Vertical" as the value for the ScrollBars property and set the value of the Multiline property to True.

6. Copy the Visual Basic code listing above, and paste it into the code editor to replace whatever code is already there.

7. Execute the code by selecting Start from the Run menu.

8. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: setRef.cpp

#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define RSA_KEY "MyRSAFullKeys"
#define INFILE "signature_template.setRef.rsa.xml"
#define OUTFILE1 "signature.setRef.rsa.obj1.xml"
#define OUTFILE2 "signature.setRef.rsa.obj2.xml"

_bstr_t infile, outfile;
IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode("//ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL SignXML(_bstr_t objID)
{

if (xmldsig->signature == NULL) {
    printf("Invalid signature template\n");
    return false;
}

IXMLDSigKeyPtr pKey = xmldsig->createKeyFromCSP(
    PROV_RSA_FULL, "", RSA_KEY, 0);
if (pKey==NULL) {
    printf("Invalid key\n");
    return false;
}

_bstr_t xpath = ".//ds:Object[@Id='"+ objID +"']";
IXMLDOMNodePtr pNode = xmldoc->selectSingleNode(xpath);
if (pNode == NULL) {
    printf("Can't select node\n");
    return false;
}

HRESULT hr = xmldsig->setReferenceData("#obj2", pNode);
if (FAILED(hr)) {
    printf("Can't set reference data\n");
    return false;
}

IXMLDSigKeyPtr pKeyOut = xmldsig->sign(pKey, KEYVALUE);
if (NULL == pKeyOut) {
    printf("sign failed.\n");
    return false;
}

printf("The specified data was signed successfully.\n");
printf("Resultant signature:\n\n");
printf((LPCSTR)xmldoc->xml);
printf("\n");

hr = xmldoc->save(outfile);
if (FAILED(hr)) {
    printf("can't save the signed signature file.\n");
    return false;
}

return true;

VARIANT_BOOL VerifyXML(_bstr_t objID)
{
    IXMLDOMNodePtr pKeyInfo, pNode;
    IXMLDSigKeyPtr pKey, pKeyOut;
pKeyInfo = xmldoc->selectSingleNode("./ds:KeyInfo/ds:KeyValue");
if (pKeyInfo == NULL) {
    printf("Invalid <ds:KeyInfo>
    return VARIANT_FALSE;
}

pKey = xmlsig->createKeyFromNode(pKeyInfo);
if (pKey== NULL) {
    printf("Invalid key from <ds:KeyInfo>
    return VARIANT_FALSE;
}

_bstr_t xpath = ".//ds:Object[@Id='" + objID + "]";
pNode = xmldoc->selectSingleNode(xpath);
HRESULT hr = xmlsig->setReferenceData("#obj2", pNode);
if (FAILED(hr)) {
    printf("Can't set Reference data source\n")
    return VARIANT_FALSE;
}

pKeyOut = xmlsig->verify(pKey);
if (pKeyOut== NULL) {
    printf("Invalid signature.\n")
    return VARIANT_FALSE;
}

printf("Signature verified on the data.\n");
return VARIANT_TRUE;

VARIANT_BOOL initObjects() {

    if (FAILED(xmlsig.CreateInstance((__uuidof(MXDigitalSignature50))))) {
        printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance((__uuidof(DOMDocument50))))) {
        printf("Installation of msxml5 is required to run this app.\n")
        return VARIANT_FALSE;
    }

    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
}

void main()
{
    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    infile = INFILE;

    printf("Attempting to sign the object of 'obj1'\n");
    outfile = OUTFILE1;
    if (VARIANT_TRUE == LoadXML(infile)) {
        SignXML("obj1");
    }

    if(VARIANT_TRUE == LoadXML(outfile)) {
        VerifyXML("obj1");
    }

    printf("Attempting to sign the object of 'obj2'\n");
    outfile = OUTFILE2;
    if (VARIANT_TRUE == LoadXML(infile)) {
        SignXML("obj2");
    }

    if(VARIANT_TRUE == LoadXML(outfile)) {
        VerifyXML("obj2");
    }

    cleanObjects();
    CoUninitialize();
}

Try It!
1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "setRefProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight setRefProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_template.setRef.rsa.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature template file from Resource Files (signature_template.setRef.rsa.xml), and paste it into the text file you just created.

   Note  You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 5-8 for the C++ file above (setRef.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting Build setRefProj.exe from the Build menu.
13. Execute the sample application by selecting !Execute setRefProj.exe from the Build menu.
14. Verify that your output is similar to that listed in the Output topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
When you build and run the `setRef` code, you should get an output similar to the following. There will be differences in the key information, and therefore in the final signature values and digest values.

Notice that in the output the signature values and digest values are different for "obj1" and "obj2".

Attempting to sign the object of 'obj1'

The specified data was signed successfully.
Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj2" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>TjR4fN1q3zgKhIrr/o7iY/la22A=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>lcP+UrGnWPb9b61Ny4zJmeQnzkkbOt+g+UsEXLBnbaFD38PTOKQ5YFJ8udOBtbR
  PbSSVizk9Hjv0udabuMKs15EJLtM/c1rvF008ZxcN77c+j/eH0GnLAqSimCVN3G
  k39o6JUG50R/KAoJ4B5NhCj1lGUk=
  </ds:SignatureValue>
  <ds:Object Id="obj1" MimeType="text/plain">Hello, World!</ds:Object>
  <ds:Object Id="obj2">Hi, everyone!</ds:Object>
</ds:Signature>
</ds:Signature>
</ds:Signature>
</ds:Signature>
```

```xml
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <RSAKeyValue>
    <Modulus>yQAHzR7z1edINO08A88UbmcS3cUnIi5TbqSakH0jL8xghT45FNMrMmR5YVdfSNKqn8e
  HYE0xjpw07Ai+07Z08K3UWcYSUmSyTPF1cVdfj6VkiN0AU3BVEA3GKBnccTd2wjRb/p
  puhNlmjwSZKL7sWifHqGZr0qar0=</Modulus>
  <Exponent>AQB</Exponent>
</RSAKeyValue>
</KeyInfo>
```
Signature verified on the data.
Attempting to sign the object of 'obj2'

The specified data was signed successfully.
Resultant signature:

```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj2" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>UDmNPz5eONRovrmzaXoZ2vm5YJs=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    FcFn47QoRE0wz3t34S3qkhGYbp7cIMBagwp8+VurTK+ImUX9NjVRKiA02ps+2FRrdZuX1cvep9x91njy0ssWN/ikmqKSEo08w15X73hrWA5a6vedS7+sisrkvnW+vY4BeztcH6ETMnbxwRafqKvwVt0+09Y=
  </ds:SignatureValue>
  <ds:Object Id="obj1" MimeType="text/plain">Hello, World!</ds:Object>
  <ds:Object Id="obj2">Hi, everyone!</ds:Object>
  <ds:Signature>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <KeyValue>
        <RSAKeyValue>
          <Modulus>
            yQAHaJRsZledIN08A88U6cE38cUNi5TboSYkHoJ8xghT45FNMM+MmR5YVdHYEQxjpw007Ai+o7Z08K3UWCYSUmSytPF1cYdfj6VkiN0AU3BVEA3GKBncCTpuhNlmJwSZKL7swifHqGZ0qarQ=
          </Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </ds:Signature>
</ds:Signature>
```

Signature verified on the data.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
**setStoreHandle Method**

[This feature was first implemented for MSXML 5.0.]

Sets a certificate store handle on the signature object, so that certificate information can be inserted into the `<ds:KeyInfo>` element.

This method is callable in C/C++ applications only.

**C/C++ Syntax**

```c
HRESULT setStoreHandle (void* hCertStore);
```

**Parameters**

*hCertStore*

Handle to a certificate store created using CryptoAPI functions, such as `CertOpenSystemStore` or `CertOpenStore`.

**Return Values**

*S_OK*

Certificate store handle was set without error.

*E_FAIL*

Failed to set the certificate store handle.

**Remarks**

Calling this method releases all the certificates previously assigned to this object.

**Example**

The example below is functionally similar to the [store property example](#). It shows the effect of setting the certificate store on an `IXMLDigitalSignatureEx` object by calling the `setStoreHandle` method. When a signature is signed
without calling this method and the CERTIFICATES flag is set in the call to the
sign method, the resultant signature document contains the certificate of the key
used in sign. However, when the signature is signed after this method is called
to set the certificate store, the resultant signature document contains all the
certificates in the store, in addition to the certificate of the key used in sign.

In this example, the certificate store has three certificates. So the signed
signature document contains four <ds:Certificate> entries, two of which are
identical.

- Resource File (signature_template.setStoreHandle.rsa.xml)
- C/C++ Code (setStoreHandle.cpp)
- Output for the setStoreHandle Method Example

Applies To

IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

IXMLDigitalSignatureEx Interface | sign Method | verify Method | store
Property
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Resource: signature_template.setStoreHandle.rsa.xml

The following resource is used as input for the `setStoreHandle` example in C/C++.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig
      <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
          <ds:DigestValue/>
        </ds:Reference>
      </ds:SignedInfo>
    <ds:SignatureValue />
    <ds:Object Id="obj" MimeType="text/plain">Hello, World!</ds:Object>
    <ds:KeyInfo></ds:KeyInfo>
  </ds:Signature>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: setStoreHandle.cpp

#include <msxml5.dll>
using namespace MSXML2;

#include <stdio.h>
#include <wincrypt.h>

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'

#define RSA_KEY "myRSAFullKeys"
#define INFILE "signature_template.setStoreHandle.rsa.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignatureExPtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;
HCERTSTORE hCertStore = NULL;

// Method that loads a source xml file:
VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }
    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }
    xmldoc->setProperty("SelectionNamespaces", DSIGNS);
    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");
    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }
    return VARIANT_TRUE;
}

// Method that signs a signature document:
VARIANT_BOOL SignXML(XMLDSIG_WRITEKEYINFO fwWriteKeyInfo)
{
    if (xmlsig->signature == NULL) {
        printf("Invalid signature template\n");
        return false;
    }

    IXMLDSigKeyPtr pKey = xmlsig->createKeyFromCSP(
        PROV_RSA_FULL, ",", RSA_KEY, 0);
    if (pKey == NULL) {
        printf("Invalid key\n");
        return false;
    }

    IXMLDSigKeyPtr pKeyOut = xmlsig->sign(pKey, fwWriteKeyInfo);
    if (NULL == pKeyOut) {
        printf("sign failed.\n");
        return false;
    }

    printf("The specified data was signed successfully.\n");
    printf("Resultant signature:\n");
    printf((LPCSTR)xmldoc->xml);
    printf("\n");
    return true;
}

// Initialize the project: creating DOM and opening the "MY" certificate store.
VARIANT_BOOL initObjects()
{
    if (FAILED(xmlsig.CreateInstance(__uuidof(MXDigitalSignature50)))) {
        printf("Installation of msxml5 is required to run this app.\n");
        return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)))) {
        printf("Installation of msxml5 is required to run this app.\n");
        return VARIANT_FALSE;
    }

    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;

    if (hCertStore = CertOpenSystemStore(0, "MY"))
    {
        //
{  
    printf("The MY system store is open. Continue.\n");
}

else
{
    printf("The MY system setStoreHandle did not open.\n");
    return VARIANT_FALSE;
}

objectsAreInitialized = VARIANT_TRUE;
return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmlsig) xmlsig.Release();
    if (hCertStore) CertCloseStore(hCertStore, 0);
}

void main()
{
    HRESULT hr;

    if (CoInitialize(NULL) == E_FAIL) {
        printf("can't initialize COM Lib\n");
        exit(-1);
    }

    if (!initObjects()) {
        cleanObjects();
        exit(-1);
    }

    if (VARIANT_TRUE == LoadXML(INFILE)) {
        // First sign without setting a store handle.
        printf("Sign with fwWriteKeyInfo = CERTIFICATES:\n");
        SignXML(CERTIFICATES);

        if (hCertStore) {
            hr = xmlsig->setStoreHandle((void*)hCertStore);
            printf("the store is set on xmlsig.\n");
        } else {
            printf("invalid certstore\n");
        }

        // Then sign with setting a store handle.
printf("Sign with fwWriteKeyInfo = CERTIFICATES:\n");
SignXML(XMLDSIG_WRITEKEYINFO(CERTIFICATES|PURGE));
}
cleanObjects();
CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "setStoreHandleProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight setStoreHandleProj files. From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature_template.setStoreHandle.rsa.xml" in the File name text box.
7. Click OK.
8. Copy the XML signature template file from Resource Files, and paste it into the text file you just created.
9. Repeat steps 5-8 for the C++ file above (setStoreHandle.cpp).
10. From the Project menu, click Settings..., then click the Link tab.
11. In Object/library modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click OK.
12. Build the sample by selecting Build setStoreHandleProj.exe from the Build menu.
13. Execute the sample application by selecting !Execute setStoreHandleProj.exe from the Build menu.
14. Verify that you output is similar to that listed in the Output topic.
Output

The output for the `setStoreHandle` example corresponds to the results from two calls to `sign`. The first call is made without setting the certificate store on the XML Digital Signature object (`hCertStore`). The second call is made after setting the certificate store. The result from the first call contains only one `<x509Data>` element. This is because when CERTIFICATES flag is passed in, `sign` tries to get the copy of the certificate of the key used to sign and, when the certificate is found as the key is created using `createKeyFromCSP`, it is inserted in the signed signature document.

The result from the second call contains four `<x509Data>` elements. This is because the `sign` method first tries to get the certificate of the key used and inserts any found certificate into the resultant signature document, as shown above. Because the certificate store is set, the `sign` method then gets all the certificates from that store and inserts them into the signed signature. In this case, there are three certificates in the store.

Output from first call

```
The MY system store is open. Continue.
Sign with fwWriteKeyInfo = CERTIFICATES:
The specified data was signed successfully.
Resultant signature:
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>8UIaiFvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>IFILa99v5A6V1v37H+OiqO0bmGyRaprUzlV3dPbx88YzDdIu9GooyMcsGvpBshBbHQoMso+/VctXUA==</ds:SignatureValue>
  <ds:Object Id="obj" MimeType="text/plain">Hello, World!</ds:Object>
```
the store is set on xmlsig.
Sign with fwWriteKeyInfo = CERTIFICATES:
The specified data was signed successfully.
Resultant signature:
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#;Object">
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
  </ds:Reference>
</ds:Signature>

Output from second call

the store is set on xmlsig.
Sign with fwWriteKeyInfo = CERTIFICATES:
The specified data was signed successfully.
Resultant signature:
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#;Object">
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
  </ds:Reference>
</ds:Signature>

Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#DigestValue"
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
sign Method

[This feature was first implemented for MSXML 5.0.]

Signs the document referenced in the <ds:Signature> element that has been assigned to the signature property of this IXMDigitalSignature object. The key object supplied must contain a private key and can be generated from one of the following methods:

- createKeyFromCSP
- createKeyFromCertContext
- createKeyFromHMACSecret
- createKeyFromHMACSecretBinary
- createKeyFromNode

[JS/Script]

Script Syntax

```javascript
var oSignedKey = objXMLDigitalSignature.sign(oKey, fwWriteKeyInfo);
```

[Visual Basic]

Visual Basic Syntax

```vbscript
Set oSignedKey = objXMLDigitalSignature.sign(oKey, fwWriteKeyInfo)
```

[C/C++]

C/C++ Syntax Using Smart Pointers

```c++
IXMLDSigKeyPtr oSignedKey = 
    objXMLDigitalSignature->sign(oKey, fwWriteKeyInfo);
```

C/C++ Syntax

```c++
HRESULT sign(
```
IXMLDSigKey* oKey,
XMLDSIG_WRITEKEYINFO fwWriteKeyInfo,
IXMLDSigKey** oSignedKey);

Parameters

oKey
A key object implementing the IXMLDSigKey interface. This object cannot be NULL.

fwWriteKeyInfo
A flag to specify how the key information should be handled in the resultant signature object. Possible values are discussed in XMLDSIG_WRITEKEYINFO enum.

oSignedKey [out, retval]
The IXMLDSigKey object used in signing. The return parameter is NULL if the method failed. Otherwise it is the same object passed in the oKey parameter.

Return Values

S_OK
The value returned if the method call was successful.

E_FAIL
The value returned if the implementation failed to get the data for CryptoAPI.

E_ACCESSDENIED
The value returned if the method is called in a non-trusted context or the caller lacks a sufficient permission to access the supplied key.

In addition, the method passes the failure codes from CryptoAPI that are not covered by S_FALSE or NULL.

Remarks

Signing using this method amounts to performing the following tasks:

1. Calculate digest values for the document referenced in every <ds:Reference> element under <ds:SignedInfo>:
   1. Resolve the URI attribute of the <ds:Reference> element. If the URI
attribute is absent, data set by IXMDigitalSignature::setReferenceData is used.

2. Perform the transformations specified in the optional <ds:transforms> child element.

3. Calculate the digest of the referenced data using the algorithm named in the <ds:DigestMethod> child element in the signature template and output the hash value as the text value of the <ds:DigestValue> child element.

2. Fill in the <ds:SignatureMethod> element with the name of the algorithm determined by the signing key. For HMAC, <ds:HMACOutputLength> is created, if absent, and its content is set according to the value of the length parameter of the createKeyFromHMACSecret or createKeyFromHMACSecretBinary method.


4. Calculate the digest of the canonicalized <ds:SignedInfo> to produce the signature hash.

5. Sign the signature hash using the private key provided and saves this signed signature as the text value of the <ds:SignatureValue> element.

6. Save the information of the key and certificates to the <ds:KeyInfo> element according the value of fWriteKeyInfo parameter passed to the sign method.

7. Return the key in the oSignedKey parameter if signing succeeded. Otherwise return NULL.

From this we see how a signature template might be set up. For more information on how to set up the signature template, see the discussions given in the signature Property topic.

**Note** The sign method is the only method that can change the <ds:Signature> element specified by the signature property.

**Note** You cannot sign any data in a non-trusted context, such as a script embedded in an HTML page. Doing so will result in an error.

**Example**

The following example illustrates how the sign method is used to sign data with XML digital signature. It uses a simple signature template.
(signature_template.sign.rsa.xml) as the input. The template has three empty sub elements: <ds:DigestValue>, <ds:SignatureValue>, and <ds:KeyInfo>. The first two will be filled in after sign returns. The last one will be filled in when sign is called with fwWriteKeyInfo=KEYVALUE. It is left intact if fwWriteKeyInfo=NOKEYINFO and all the existing content is cleared if fwWriteKeyInfo=PURGE. When fwWriteKeyInfo=CERTIFICATES, a <X509Data> element is inserted when the certificate of the key used is available.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- **Resource File (signature_template.sign.rsa.xml)**
- **JScript Code (sign.js)**
- **Visual Basic Code (sign.frm)**
- **C/C++ Code (sign.cpp)**
- **Output for the sign Example**

**Applies To**

IXMLDigitalSignature | IXMLDigitalSignatureEx

**Versioning**

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the Language Filter button ☐ in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Resource:
signature_template.sign.rsa.xml

Notice that the `<ds:DigestValue>`, `<ds:SignatureValue>`, and `<ds:KeyInfo>` elements are empty in the signature template.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj"
      Type="http://www.w3.org/2000/09/xmldsig#;Object">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:Object Id="obj" MimeType="text/plain">Hello, World!</ds:Object>
  <ds:KeyInfo></ds:KeyInfo>
</ds:Signature>
```
var xmldoc, xmldsig, infile;
var szResult = "";

NOKEYINFO = 0;
KEYVALUE = 1;
CERTIFICATES = 2;
PURGE = 4;

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
PROV_RSA_FULL = 1;
// Change this key container name to your own if necessary.
RSA_KEY = "MyRSAFullKeys";

csp = PROV_RSA_FULL;
key = RSA_KEY;
inFile = "signature_template.sign.rsa.xml";

if (InitXML())
{
    if (LoadXML(infile)) {
        alert("Sign with fwWriteKeyInfo = NOKEYINFO:");
        SignXML(NOKEYINFO);

        alert("Sign with fwWriteKeyInfo = KEYVALUE:");
        SignXML(KEYVALUE|PURGE);

        alert("Sign with fwWriteKeyInfo = CERTIFICATES:");
        SignXML(CERTIFICATES);

        alert("Sign with fwWriteKeyInfo = CERTIFICATES|PURGE:");
        SignXML(CERTIFICATES|PURGE);

        alert("Sign with fwWriteKeyInfo = PURGE:");
        SignXML(PURGE);
    }
}

///////// Helper functions. ///////////
function alert(str) { WScript.echo(str); }
function InitXML()
{
    try {
        xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
        xmldsig = new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
    }
    catch (e) {
        alert("Installation of mxsml5 is required to run this app.
    return false;
    }
    xmldoc.async = false;
    xmldoc.preserveWhiteSpace = true;
    xmldoc.validateOnParse = false;
    xmldoc.resolveExternals = false;
    return true;
}
function LoadXML(file)
{
    if (xmldoc == null) {
        alert("must instantiate xml dom\n");
        return false;
    }
    if (!xmldoc.load(file)) {
        alert("Can't load "+ file + "\n");
        return false;
    }
    xmldoc.setProperty("SelectionNamespaces", DSIGNS);
    xmldsig.signature = xmldoc.selectSingleNode("./ds:Signature");
    return true;
}
function SignXML(fwWriteKeyInfo)
{
    if (!xmldsig.signature)
    {
        alert("Invalid signature template\n");
        return false;
    }
    var oKey = xmldsig.createKeyFromCSP(csp, ",", key, 0);
    if (!oKey)
    { 
        alert("Invalid key.\n");
        return false;
    }
```javascript
var oSignedKey = xmldsig.sign(oKey, fwWriteKeyInfo);
if (!oSignedKey)
{
    alert("sign failed.\n");
    return false;
}

alert("The specified data was signed successfully.\n"+
   "Resultant signature:\n"+
   xmlDoc.xml + "\n");
return true;
```

**Try It!**

1. Ensure that you have completed all the procedures in [Getting Started with XML Digital Signatures](#).
2. Copy the XML signature template from [Resource Files](#), and paste it into a text file. Save the file as `signature_template.sign.rsa.xml`.
3. Copy the JScript listing above, and paste it into a text file. Save the file as `sign.js`, in the same directory where you saved `signature_template.sign.rsa.xml`.
4. From a command prompt, navigate to this directory, then type "cscript sign.js".

   **Note**  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run cscript.exe or wscript.exe), if it is not already installed.

5. Verify that your output is the similar to that listed in the [Output](#) topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Visual Basic Source: sign.frm

Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim dataObj As IXMLDOMNode
Dim infile, provType, keyContainer

Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
Const PROV_RSA_FULL = 1
' Change this key container name to your own if necessary.
Const RSA_KEY = "MyRSAFullKeys"

Private Function WriteLine(ByVal str As String)
    Text1.Text = Text1.Text + str + vbNewLine
End Function
Private Function writeClear()
    Text1.Text = ""
End Function

Private Function LoadXML(ByVal file As String)
    ' Read input XML file and display the content in the text3.
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.resolveExternals = False
    If xmldoc.Load(Path) = False Then
        WriteLine "Can't load " + Path
        WriteLine "Reason: " + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    Set xmldsig.signature = xmldoc.selectSingleNode("//ds:Signature
    LoadXML = True
End Function
Private Function SignXML(fwWriteKeyInfo As Long)
    If xmldsig.signature Is Nothing Then
        WriteLine "Invalid signature template."
        SignXML = False
        Exit Function
    End If

    If keyContainer = "" Then
        WriteLine "Invalid key container."
SignXML = False
Exit Function
End If

Set oKey = xmldsig.createKeyFromCSP(provType, "", keyContainer, 0)
If oKey Is Nothing Then
  WriteLine "Invalid key"
  SignXML = False
  Exit Function
End If

Set oSignedKey = xmldsig.sign(oKey, fwWriteKeyInfo)
If oSignedKey Is Nothing Then
  WriteLine "sign failed."
  SignXML = False
  Exit Function
End If

WriteLine "The specified data was signed successfully."
WriteLine "Resultant signature: "
WriteLine xmlldoc.xml

SignXML = True
End Function

Private Sub Form_Load()
  'Set text box to use form to determine its width
  'and height when form is loaded
  Text1.Left = 100
  Text1.Top = 100
  Text1.Width = Form1.Width - 350
  Text1.Height = Form1.Height - 750

  provType = PROV_RSA_FULL
  keyContainer = RSA_KEY
  infile = "signature_template.sign.rsa.xml"

  writeClear
  If LoadXML(infile) = True Then
    WriteLine "Sign with fwWriteKeyInfo = NOKEYINFO:"
    SignXML NOKEYINFO

    WriteLine "Sign with fwWriteKeyInfo = KEYVALUE:"
    SignXML KEYVALUE

    WriteLine "Sign with fwWriteKeyInfo = CERTIFICATES:"
    SignXML CERTIFICATES

    WriteLine "Sign with fwWriteKeyInfo = CERTIFICATES|PURGE:"
    SignXML CERTIFICATES + PURGE
  End If
End Sub
WriteLine "Sign with fwWriteKeyInfo = PURGE:
    SignXML PURGE
End If

End Sub

Private Sub Form_Resize()
    'Set text box to use form in determining its width and height
    'when form is resized
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature template resource file, and paste it into a text file. Save the file as signature_template.sign.rsa.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as sign.vbp to the same directory where you saved signature_template.sign.rsa.xml. Save the form file as sign.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, and then check the box for Microsoft XML, v5.0.
5. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named "Text1". In the Properties window, set the value for the ScrollBar property to "2 – Vertical" and the value of the MultiLine property to True.
6. Copy the Visual Basic code listing above, and paste it into the code editor to replace whatever is already there.
7. Execute the code by selecting Start from the Run menu.
8. Verify that the output is similar to that listed in the Output topic. You can resize the form as needed to view the output text more easily.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
This C++ code example makes use of smart pointer class wrappers around the interface.

```c++
#include <stdio.h>
#include <msxml5.dll>
using namespace MSXML2;

#define DSIGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
// Change this key container name to your own if necessary.
#define RSA_KEY "MyRSAFullKeys"
#define INFILE "signature_template.sign.rsa.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;
VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized)
    {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE)
    {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSIGNS);

    // set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode(".//ds:Signature");

    if (xmldsig->signature == NULL)
    {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL SignXML(XMLDSIG_WRITEKEYINFO fwWriteKeyInfo)
{

if (xmlsig->signature == NULL) {
    printf("Invalid signature template\n");
    return false;
}

IXMLDSigKeyPtr pKey = xmlsig->createKeyFromCSP(
    PROV_RSA_FULL, "", RSA_KEY, 0);
if (pKey==NULL) {
    printf("Invalid key\n");
    return false;
}

IXMLDSigKeyPtr pKeyOut = xmlsig->sign(pKey, fwWriteKeyInfo);
if (NULL == pKeyOut) {
    printf("sign failed.\n");
    return false;
}

printf("The specified data was signed successfully.\n");
printf("Resultant signature:\n");
printf((LPCSTR)xmldoc->xml);
printf("\n");
return true;
}

VARIANT_BOOL initObjects()
{
    if (FAILED(xmlsig.CreateInstance(__uuidof(MXDigitalSignature50)) ) ) {
        printf("Installation of msxml5 is required to run this app.\n"
    return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)) ) ) { 
    printf("Installation of msxml5 is required to run this app.\n"
    return VARIANT_FALSE;
    }

    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;

    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}

void cleanObjects()
{
if (xmldoc) xmldoc.Release();
if (xmlsig) xmlsig.Release();
}

void main()
{
if ( CoInitialize(NULL) == E_FAIL) {
    printf("can't initialize COM Lib\n");
    exit(-1);
}
if (!initObjects()) {
    cleanObjects();
    exit(-1);
}
if (VARIANT_TRUE == LoadXML(INFILE)) {
    printf("Sign with fwWriteKeyInfo = NOKEYINFO:\n");
    SignXML(NOKEYINFO);
    printf("Sign with fwWriteKeyInfo = KEYVALUE:\n");
    SignXML(KEYVALUE);
    printf("Sign with fwWriteKeyInfo = CERTIFICATES:\n");
    SignXML(CERTIFICATES);
    printf("Sign with fwWriteKeyInfo = CERTIFICATES|PURGE:\n");
    SignXML(XMLDSIG_WRITEKEYINFO(CERTIFICATES|PURGE));
    printf("Sign with fwWriteKeyInfo = PURGE:\n");
    SignXML(PURGE);
}
    cleanObjects();
    CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "signProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The **Win32 Console Application** property page will appear. For the type of Console Application, select **An empty project** and click **Finish**. When the **New Project Information** box displays, click **OK**.

5. Select **FileView** on the project browser, and highlight **signProj files**. From the **File** menu, select **New**.

6. On the **Files** tab of the **New** dialog box, highlight **C++ Source File**. Then type "signature_template.sign.rsa.xml" in the **File name** text box.

7. Click **OK**.

8. Copy the XML signature template **resource file**, and paste it into the text file you just created.

   **Note**  You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 5-8 for the **C++ file** above (**sign.cpp**).

10. From the **Project** menu, click **Settings...**, then click the **Link** tab.

11. In **Object/library** modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click **OK**.

12. Build the sample by selecting **Build signProj.exe** from the **Build** menu.

13. Execute the sample application by selecting **!Execute signProj.exe** from the **Build** menu.

14. Verify that your output is the similar to that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

When you build and run the `sign` code, you should get output similar to the following. The differences are due to the cryptographic key used. Compare the output from the two cases where the `fwWriteKeyInfo` parameter is set to `CERTIFICATES` versus set to `CERTIFICATES|PURGE`.

Sign with `fwWriteKeyInfo` = NOKEYINFO:
The specified data was signed successfully.
Resultant signature:
```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    dvMOHXb+YMNVixBloLAHXm4QoXN5ykEufYRV9ZXySf9bhdDBrq9JtxOGpm0SzjXW5TgnVvv6L3nA==
  </ds:SignatureValue>
  <ds:Object Id="obj" Mime-type="text/plain">Hello, World!</ds:Object>
  <ds:KeyInfo></ds:KeyInfo>
</ds:Signature>
```

Sign with `fwWriteKeyInfo` = KEYVALUE:
The specified data was signed successfully.
Resultant signature:
```xml
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
```

Sign with fwWriteKeyInfo = CERTIFICATES:
The specified data was signed successfully.
Resultant signature:
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nh4="</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    dvMOHxB+YMNV1xBoLoLAHJXYm4QoXN5ykEUfYRV9ZXYsF9bhdDBrq9Jtx0Gpm0SzjXW5TgnVvv6L3nA==
  </ds:SignatureValue>
  <ds:Object Id="obj" MimeType="text/plain">Hello, World!</ds:Object>
  <ds:KeyInfo>
    <KeyValue xmlns="http://www.w3.org/2000/09/xmldsig#">
      <RSAKeyValue>
        <Modulus>
          4yn1pJSp1PC8i3961dqcHm8OGyRxMgGREwq5WyMEvnnQAinRoN6LFk0jKrBgKVks4kfSwJDaw==
        </Modulus>
        <Exponent>AQAB</Exponent>
      </RSAKeyValue>
    </KeyValue>
  </ds:KeyInfo>
</ds:Signature>
Sign with fwWriteKeyInfo = CERTIFICATES|PURGE:
The specified data was signed successfully.
Resultant signature:

<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:KeyInfo>
    <ds:RecipientKeyInfo/>
  </ds:KeyInfo>
  <ds:SignatureValue/>
</ds:Signature>
dvMOHXb+YMNV1xBloLAHJXYm4QoXN5ykEUfYRV9ZXySf9bhdDBrq9Jtx0Gpm0SzjXW5TgnVv6L3nA==

<ds:SignatureValue>
<ds:Object	Id="obj"	MimeType="text/plain">
Hello,
World!</ds:Object>
</ds:Signature>

Sign with fwWriteKeyInfo = PURGE:
The specified data was signed successfully.
Resultant signature:
<?xml version="1.0"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#obj" Type="http://www.w3.org/2000/09/xmldsig#;Object">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>8UIaiFvvufk+ybGUHcMk5+2Nmh4=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
</ds:Signature>
dvM0HXb+YMNV1xBloLAHJXYm4QoXN5ykEUfYRV9ZXySf9bhdDBrq9Jtx0Gpm0SzjXW5TgnVvv6L3nA==
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
verify Method

[This feature was first implemented for MSXML 5.0.]

Verifies the data referenced in an already signed XML digital signature document, using the embedded or a supplied public key.

[JS/Script]

**JS/Script Syntax**

```javascript
var verifiedKey = objXMLDigitalSignature.verify(key);
```

[Visual Basic]

**Visual Basic Syntax**

```vbnet
Set verifiedKey = objXMLDigitalSignature.verify(key)
```

[C/C++]

**C/C++ Syntax Using Smart Pointers**

```c
IXMLDSigKeyPtr verifiedKey = objXMLDigitalSignature->verify(key);
```

**C/C++ Syntax**

```c
HRESULT verify(
    IXMLDSigKey* key,
    IXMLDSigKey** verifiedKey);
```

**Parameters**

- `key [in]`

  An `IXMLDSigKey` object obtained from one of the following methods: `createKeyFromNode`, `createKeyFromHMACSecret`, or `createKeyFromHACSecretBinary`. If NULL, this method obtains the
required key information from the Signature/KeyInfo/(KeyValue|X509Data) elements.

`verifiedKey [out, retval]`

The IXmLDSigKey object actually used to verify. NULL if the signature and digest values are invalid and the method fails.

**Return Values**

- **S_OK**
  Signature verification succeeded.
- **S_FALSE**
  Signature or digest are invalid.
- **E_FAIL**
  Signature verification failed.

In addition, the method passes the failure codes from CryptoAPI that are not covered by S_FALSE or NULL.

**Remarks**

A valid signature with legitimate digest and signature values and appropriate key information, as opposed to a signature template, must be set to the signature property before this method is called. For more information see the signature property.

To verify signatures, your code must perform the following tasks:

2. Calculate the digest of canonicalized `<ds:SignedInfo>` using the SHA1 algorithm.
3. Verify the signature value (<`ds:SignatureValue`>) using the provided or embedded key and the digest obtained in the previous step.
4. If the signature values do not match, the method returns and the verification is deemed failed and the output `verifiedKey` parameter is set to NULL. Otherwise it continues.
5. Walk through the `<ds:Reference>` elements, perform all specified transforms, compute the digest, and compare it with the `<ds:DigestValue>`
content of each entry.

6. If all computed digest and signature values match their declared counterparts, the signature is deemed valid and the key used is returned as the output parameter to the caller. Otherwise, the signature is invalid and the output key object is set to NULL.

After the verify method returns successfully, you can call getVerifyingCertificate or getVerifyingCertificateContext to retrieve existing certificates and verify their validity using either CryptoAPI or CAPICOM.

If an embedded key contains a <ds:RetrievalMethod> element, an additional query of the key will be made using the method declared in this element. A valid <ds:RetrieveMethod> element must contain a valid URI attribute and may or may not contain <ds:transforms>.

Example

This example illustrates how to use the verify method to verify a signature.

We've provided source files for the sample in three languages: JScript, Visual Basic, and C++. The output is the same in each language.

- Resource File (signature.verify.dsa.xml)
- JScript Code (verify.js)
- Visual Basic Code (verify.frm)
- C/C++ Code (verify.cpp)
- Output for the verify Example

Applies To

IXMLDigitalSignature | IXMLDigitalSignatureEx

Versioning

MSXML 5.0 and later

To view reference information for Visual Basic, C/C++, or Script only, click the
Language Filter button in the upper-left corner of the page.

See Also

sign Method | signature Property | createKeyFromNode Method | createKeyFromHMACSecret Property
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <ds:Reference URI="#obj2" Type="http://www.w3.org/2000/09/xmldsig#Object">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>UDmNPz5eONROvrmzaVoZ2v5YJs==</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>VN9YkEwx1vp5L/0K2Vky838duwNc+DdudpA7Z0xMcr5mgw/yL0Kg==</ds:SignatureValue>
  <ds:Object Id="obj1" MimeType="text/plain">Hello, World!</ds:Object>
  <ds:Object Id="obj2">Hi, everyone!</ds:Object>
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KeyValue>
      <DSAKeyValue>
        <P>
          wLHwVoUYz2txHknKUQoN/u1ftCXqw61ed8/
pC1WPv6EV6FBx10E8vGHJfzm3ZLm4GU/
j8VsNXyDti8zGFxmhqO05Me2sUp0gZhZpe7Zn0YMaFXWgeKDg/
          gb9DrK/USj4BGueB3HAzmFQT8Qu6SM/YBGKph7JTddYXbodU=</P>
        </P>
        <Q>nkI1onvHNR4tMsvPMc0rHOBhGE=</Q>
        <G>
          QD60UIeg64ntvVbp1Rbw+wempPCIjg1Aq+
w67C3PHqRfor9fBeKf6J6Ap11lQsBBSau6SOOL
          LkHvR2jwsygAdYMYSuUYoTJZ66pGkMy2U7121/
oKr1kF8MrEzh3sesA2JfV+h396K2wa/
          Nv19s8xTcDhxLepcnAewKhz5nY=
        </G>
        <Y>
          rv1GX+/qY99y0xGrdaJAN1m7YjJ5/
          j7famSv6cHVeUwY79wXFBD1FGNv0K
          GiyGsRG6BXO0G4yHL0ndIsywSMYD
          ejb+TuUfvkGcEPGDZertPy1F/xgJ4
          8jhPttwrv8Wy3k279+ufB27NpBC/v
          W2Rxf3bh22L7Vo0KNaWuTwie2g=
        </Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
</ds:Signature>
<PgenCounter>AAAAZQ==</PgenCounter>  
<Seed>soBYpn66Yy/B4Or2y1zqB2mZr//U==</Seed>  
</DSAKeyValue>  
</KeyValue>  
</KeyInfo>  
</ds:Signature>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
JScript Source: verify.js

var xmldoc, xmldsig, infile;

DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'";
infile = "signature.verify.dsa.xml";

if (InitXML())
{
  alert("Verifying signature.\n");
  if (LoadXML(infile)) {
    VerifyXML();
  }
}

///////// Helper functions: //////////
function alert(str) { WScript.echo(str); }

///////// Set signature for signing. //////////
function InitXML()
{
  try {
    xmldoc = new ActiveXObject("Msxml2.DOMDOcument.5.0");
    xmldsig = new ActiveXObject("Msxml2.MXDigitalSignature.5.0");
  }
  catch (e) {
    alert("Installation of mxsml5 is required to run this app.\n");
    return false;
  }

  xmldoc.async = false;
  xmldoc.preserveWhiteSpace = true;
  xmldoc.validateOnParse = false;
  xmldoc.resolveExternals = false;
  return true;
}

function LoadXML(file)
{
  if (xmldoc == null) {
    alert("must instantiate xml dom\n");
    return false;
  }

  if (!xmldoc.load(file)) {
    alert("Can't load " + file + "\n");
  }
return false;
}
xmlDoc.setProperty("SelectionNamespaces", DSIGNS);
xmldsig.signature = xmlDoc.selectSingleNode(".//ds:Signature");
return true;
}

function VerifyXML()
{
    if (!xmldsig.signature) {
        alert("Invalid signature.\n");
        return false;
    }

    var oKeyInfo = xmlDoc.selectSingleNode(".//ds:KeyInfo/ds:KeyValue");
    if (!oKeyInfo) {
        alert("Invalid <KeyInfo> element.\n");
        return false;
    }

    var oKey = xmldsig.createKeyFromNode(oKeyInfo);
    if (!oKey)
    {
        alert("Failed to create key from <KeyInfo>\n");
        return false;
    }

    var oVerifiedKey = xmldsig.verify(oKey);
    if (oVerifiedKey == null) {
        alert("Signature not verified.\n");
    }

    alert("Signature verified.\n");
    return true;
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature resource file, and paste it into a text file. Save the file as signature.verify.dsa.xml.
3. Copy the JScript listing above, and paste it into a text file. Save the file as verify.js, in the same directory where you saved signature.verify.dsa.xml.
4. From a command prompt, navigate to this directory, then type "cscript verify.js".
**Note**  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (to run cscript.exe or wscript.exe), if it is not already installed.

5. Verify that your output is the same as that listed in the [Output](#) topic.
Dim xmldoc As New DOMDocument50
Dim xmldsig As New MXDigitalSignature50
Dim dsigKey As IXMLDSigKey
Dim dataObj As IXMLDOMNode

Const DSIGNS = "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
Const INFILE = "signature.verify.dsa.xml"

Private Function WriteLine(ByVal str As String)
    Text1.Text = Text1.Text + str + vbNewLine
End Function

Private Function writeClear()
    Text1.Text = ""
End Function

Private Function LoadXML(ByVal file As String)
    ' Read input xml file and display the content in the text1.
    Path = App.Path + "\" + file
    xmldoc.async = False
    xmldoc.preserveWhiteSpace = True
    xmldoc.validateOnParse = False
    xmldoc.resolveExternals = False
    If xmldoc.Load(Path) = False Then
        WriteLine "Can't load " + Path
        WriteLine "Reason: " + xmldoc.parseError.reason
        LoadXML = False
        Exit Function
    End If
    xmldoc.setProperty "SelectionNamespaces", DSIGNS
    Set xmldsig.signature = xmldoc.selectSingleNode(".//ds:Signature")
    LoadXML = True
End Function

Private Function VerifyXML()
    If xmldsig.signature Is Nothing Then
        WriteLine "Invalid signature."
        VerifyXML = False
        Exit Function
    End If

    Set oKeyInfo = xmldoc.selectSingleNode(".//ds:KeyInfo/ds:KeyValue")
    If oKeyInfo Is Nothing Then
        WriteLine "Invalid <KeyInfo> element."
    End If
End Private
VerifyXML = False
Exit Function
End If

Set oPubKey = xmldsig.createKeyFromNode(oKeyInfo)
If oPubKey Is Nothing Then
    WriteLine "Can't generate public key for verification."
    VerifyXML = False
    Exit Function
End If

Set oVerifiedKey = xmldsig.verify(oPubKey)
If oVerifiedKey Is Nothing Then
    WriteLine "Signature not verified."
End If

WriteLine "Signature verified."
VerifyXML = True
End Function

Private Sub Form_Load()
    writeClear
    WriteLine "Verify in signature."
    If LoadXML(INFILE) = True Then
        VerifyXML = True
    End If
End Sub

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
2. Copy the XML signature resource file, and paste it into a text file. Save the file as signature.verify.dsa.xml.
3. Create a Standard EXE project in Visual Basic. Save the empty project as verify.vbp to the same directory where you saved signature.verify.dsa.xml. Name the form file verify.frm.
4. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, then check the box for Microsoft XML, v5.0.
5. Copy the Visual Basic code listing above, and paste it into the code editor to replace whatever is already there.
6. Place a TextBox control onto the form. It should be named as Text1 (the default name). Click it to select it in the Properties window, and then modify the following properties for Text1: set the ScrollBars property to "2
– Vertical" and the **MultiLine** Property to True.
7. Execute the code by selecting **Start** from the **Run** menu.
8. Verify that the output is the same as that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
C/C++ Source: verify.cpp

#include <stdio.h>
#import <msxml5.dll>
using namespace MSXML2;

#define DSGNS "xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
#define INFILE "signature.verify.dsa.xml"

IXMLDOMDocument3Ptr xmldoc = NULL;
IXMLDigitalSignaturePtr xmldsig = NULL;

VARIANT_BOOL objectsAreInitialized = VARIANT_FALSE;

VARIANT_BOOL LoadXML(_bstr_t sigFile)
{
    if (!objectsAreInitialized) {
        printf("Must initialize objects before loading signature.\n");
        return VARIANT_FALSE;
    }

    if (xmldoc->load(sigFile) == VARIANT_FALSE) {
        printf("Can't load %s\n", (LPCSTR)sigFile);
        return VARIANT_FALSE;
    }

    xmldoc->setProperty("SelectionNamespaces", DSGNS);

    // Set the signature property to a <ds:Signature> DOM node.
    xmldsig->signature = xmldoc->selectSingleNode("./ds:Signature");

    if (xmldsig->signature == NULL) {
        printf("Failed to set the signature property.\n");
        return VARIANT_FALSE;
    }

    return VARIANT_TRUE;
}

VARIANT_BOOL VerifyXML() {
    IXMLDOMNodePtr pKeyInfo, pNode;
    IXMLDSigKeyPtr pKey, pKeyOut;

    pKeyInfo = xmldoc->selectSingleNode("./ds:KeyInfo/ds:KeyValue");

    return VARIANT_TRUE;
}
if (pKeyInfo == NULL) {
    printf("Invalid <ds:KeyInfo>\n");
    return VARIANT_FALSE;
}

pKey = xmldsig->createKeyFromNode(pKeyInfo);
if (pKey == NULL) {
    printf("Invalid key from <ds:KeyInfo>\n");
    return VARIANT_FALSE;
}

pKeyOut = xmldsig->verify(pKey);
if (pKeyOut == NULL) {
    printf("Invalid signature.\n");
    return VARIANT_FALSE;
}

printf("Signature verified.\n");
return VARIANT_TRUE;
}

VARIANT_BOOL initObjects()
{
    if (FAILED(xmldsig.CreateInstance(__uuidof(MXDigitalSignature50)))
        printf("Installation of msxml5 is required to run this app.\n"
                return VARIANT_FALSE;
    }

    if (FAILED(xmldoc.CreateInstance(__uuidof(DOMDocument50)))
        printf("Installation of msxml5 is required to run this app.\n"
                return VARIANT_FALSE;
    }

    xmldoc->async = VARIANT_FALSE;
    xmldoc->validateOnParse = VARIANT_FALSE;
    xmldoc->preserveWhiteSpace = VARIANT_TRUE;
    xmldoc->resolveExternals = VARIANT_FALSE;
    objectsAreInitialized = VARIANT_TRUE;

    return VARIANT_TRUE;
}

void cleanObjects()
{
    if (xmldoc) xmldoc.Release();
    if (xmldsig) xmldsig.Release();
}

void main()
{
if ( CoInitialize(NULL) == E_FAIL) {
    printf("can't initialize COM Lib\n");
    exit(-1);
}

if (!initObjects()) {
    cleanObjects();
    exit(-1);
}

printf("Verifying signature.\n\n");
if (VARIANT_TRUE == LoadXML(INFILE)) {
    VerifyXML();
}

    cleanObjects();
    CoUninitialize();
}

Try It!

1. Ensure that you have completed all the procedures in Getting Started with XML Digital Signatures.
3. From the File menu, select New. On the Projects tab of the New dialog box that appears, select Win32 Console Application in the left pane. Then type "verifyProj" in the Project name field. For the project Location field, either accept the default setting or choose another location. Click OK.
4. The Win32 Console Application property page will appear. For the type of Console Application, select An empty project and click Finish. When the New Project Information box displays, click OK.
5. Select FileView on the project browser, and highlight verifyProj files.
From the File menu, select New.
6. On the Files tab of the New dialog box, highlight C++ Source File. Then type "signature.verify.dsa.xml" in the File name text box.
7. Copy the XML signature resource file, and paste it into the file you've just created.
8. Click OK.

Note You can also copy the file into the project's main directory using Windows Explorer (or a command prompt).

9. Repeat steps 5-8 for the C++ file above (verify.cpp).
10. From the **Project** menu, click **Settings...**, then click the **Link** tab.
11. In **Object/library** modules, type "crypt32.lib". Insert this text either before or after the existing string that lists all objects and modules for the current project. Then click **OK**.
12. Build the sample by selecting **Build verifyProj.exe** from the **Build** menu.
13. Execute the sample application by selecting **!Execute verifyProj.exe** from the **Build** menu.
14. Verify that the result is the same as that listed in the **Output** topic.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
Output

When you build and run the `verify` code, you should get the following output.

Verifying signature.

Signature verified.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Enumerated Types

The XML Digital Signature implementation uses the following enumerated types:

<table>
<thead>
<tr>
<th>Enumerated Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLDSIG_WRITEKEYINFO</td>
<td>Defines flags used to specify the modes in which the sign method inserts the key information in a signed signature document.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XMLDSIG_WRITEKEYINFO Enum

The XMLDSIG_WRITEKEYINFO enumeration defines flags used to specify how the <ds:KeyInfo> element of a signed signature document will be filled in after the XML data is signed successfully. Each of the element is passed in as an input parameter in the sign Method.

Syntax

```cpp
enum _XMLDSIG_WRITEKEYINFO
{
    NOKEYINFO = 0,
    KEYVALUE = 1,
    CERTIFICATES = 2,
    PURGE = 4
};
```

Values

NOKEYINFO
No information will be inserted into the <ds:KeyInfo> element. Any existing content of the <ds:KeyInfo> element remains intact.

KEYVALUE
Add an appropriate <ds:KeyValue> element and its relevant child elements to the <ds:KeyInfo> element, leaving other elements intact.

CERTIFICATES
Add available, certificates to <ds:KeyInfo>, leaving other elements intact. In this case, the <ds:KeyInfo> element will contain the <ds:X509Data> element and its relevant child elements. See Remarks below for more information.

PURGE
The <ds:KeyInfo> element will be empty. Any existing content is purged.

Remarks

The fWriteKeyInfo parameter value passed in to the sign method can take an individual elemental value listed above or a combination of them. The combined
values have the following meaning:

<table>
<thead>
<tr>
<th>fWriteKeyInfo</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYVALUE</td>
<td>Add key value and certificates to <code>&lt;ds:KeyInfo&gt;</code>, leaving anything else intact.</td>
</tr>
<tr>
<td>CERTIFICATES</td>
<td>Remove everything from <code>&lt;ds:KeyInfo&gt;</code> first and then add key value and certificates.</td>
</tr>
<tr>
<td>PURGE</td>
<td>Remove everything from <code>&lt;ds:KeyInfo&gt;</code> first and then add key value.</td>
</tr>
<tr>
<td>KEYVALUE</td>
<td>Remove everything from <code>&lt;ds:KeyInfo&gt;</code> first and then add certificates.</td>
</tr>
</tbody>
</table>

When you sign data with the CERTIFICATES flag and the key has a matching certificate in the "MY" certificate store, the certificate will be inserted into the resultant signature document. A key has a matching certificate if any of the following conditions are true:

- It is created using the `createKeyFromCertContext` method.
- It is created using the `createKeyFromCSP` method and there is a corresponding certificate in the "MY" certificate store.
- A certificate store has been set using the `putref_store` property method or the `setStoreHandle` method.

**Example**

See the example given for the `sign Method`. 
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
# Supported Transforms and Their Namespace URIs

MSXML supports the following transforms in processing an XML digital signature:

<table>
<thead>
<tr>
<th>Transform</th>
<th>Element in Signature Template</th>
<th>Namespace URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1</td>
<td><code>&lt;SignatureDigest&gt;</code></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#sha1">http://www.w3.org/2000/09/xmldsig#sha1</a></td>
</tr>
<tr>
<td>DSA with SHA1 (DSS)</td>
<td><code>&lt;SignatureMethod&gt;</code></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#ds">http://www.w3.org/2000/09/xmldsig#ds</a></td>
</tr>
<tr>
<td>RSA with SHA1</td>
<td><code>&lt;SignatureMethod&gt;</code></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#rs">http://www.w3.org/2000/09/xmldsig#rs</a></td>
</tr>
<tr>
<td>HMAC with SHA1</td>
<td><code>&lt;signatureMethod&gt;</code></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#hr">http://www.w3.org/2000/09/xmldsig#hr</a></td>
</tr>
<tr>
<td>Canonical XML without comments</td>
<td><code>&lt;CanonicalizationMethod&gt;</code></td>
<td><a href="http://www.w3.org/TR/2001/REC-xml-20010315">http://www.w3.org/TR/2001/REC-xml-20010315</a></td>
</tr>
<tr>
<td>Canonical XML with comments</td>
<td><code>&lt;CanonicalizationMethod&gt;</code></td>
<td><a href="http://www.w3.org/TR/2001/REC-xml-20010315#WithComments">http://www.w3.org/TR/2001/REC-xml-20010315#WithComments</a></td>
</tr>
<tr>
<td>XSLT</td>
<td><code>&lt;Transformation&gt;</code></td>
<td><a href="http://www.w3.org/TR/1999/REC-xslt-19991116">http://www.w3.org/TR/1999/REC-xslt-19991116</a></td>
</tr>
<tr>
<td>XPath</td>
<td><code>&lt;Transformation&gt;</code></td>
<td><a href="http://www.w3.org/TR/1999/REC-xpathl19991116">http://www.w3.org/TR/1999/REC-xpathl19991116</a></td>
</tr>
<tr>
<td>Enveloped Signature</td>
<td><code>&lt;Transformation&gt;</code></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#er">http://www.w3.org/2000/09/xmldsig#er</a> signature</td>
</tr>
<tr>
<td>Base64 encoding</td>
<td><code>&lt;Transformation&gt;</code></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#b64">http://www.w3.org/2000/09/xmldsig#b64</a></td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
XML Digital Signature Elements

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

Digital Signature Elements

<ds:DSAKeyValue>
<ds:CanonicalizationMethod>
<ds:DigestMethod>
<ds:DigestValue>
<ds:Exponent>
<ds:HMACOutputLength>
<ds:KeyInfo>
<ds:KeyName>
<ds:KeyValue>
<ds:Manifest>
<ds:MgmtData>
<ds:Modulus>
<ds:Object>
<ds:PgenCounter>
<ds:PGPData>
<ds:PGPKeyID>
<ds:PGPKeyPacket>
<ds:Reference>
<ds:RetrievalMethod>

No information available for this release.
No information available for this release.
Other Resources
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:CanonicalizationMethod>

Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:CanonicalizationMethod
   Algorithm=anyURI>
</ds:CanonicalizationMethod>

Attributes

Algorithm
   Required.

Element Information

Number of occurrences Unlimited
Parent elements <ds:SignedInfo>
Child elements
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:DigestMethod> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:DigestMethod
   Algorithm=anyURI>
</ds:DigestMethod>

Attributes

Algorithm
   Required.

Element Information

Number of occurrences Unlimited
Parent elements <ds:Reference>
Child elements None.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:DSAKeyValue> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:DSAKeyValue/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyValue>
Child elements <ds:PgenCounter>, <ds:Seed>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:DigestValue> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:DigestValue/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:reference>
Child elements None.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:KeyInfo> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:KeyInfo
  Id=ID>
</ds:KeyInfo>

Attributes

Id
  Optional.

Element Information

Number of occurrences  Unlimited
Parent elements     <ds:Signature>
                      <ds:KeyName>, <ds:KeyValue>, <ds:MgmtData>,
Child elements      <ds:PGPData>, <ds:RetrievalMethod>,
                      <ds:SPKIData>, <ds:X509Data>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:KeyName> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:KeyName/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyInfo>
Child elements
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:KeyValue> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:KeyValue/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyInfo>
Child elements <ds:DSAKeyValue>, <ds:RSA:KeyValue>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:Manifest> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:Manifest
   Id=ID>
</ds:Manifest>

Attributes

Id
   Optional.

Element Information

Number of occurrences Unlimited
Parent elements
Child elements <ds:Reference>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:MgmtData> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:MgmtData/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyInfo>
Child elements

Remarks

Example
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:Object> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:Object
   Id=ID
   MimeType=string
   Encoding=anyURI>
</ds:Object>

Attributes

Id
   Optional.
MimeType
   Optional.
Encoding
   Optional.

Element Information

Number of occurrences Unlimited

Parent elements
Child elements
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:PGPData> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:PGPData/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyInfo>
Child elements <ds:PGPKeyID>, <ds:PGPKeyPacket>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:Reference> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:Reference
   Id=ID
   URI=anyURI
   Type=anyURI>
</ds:Reference>

Attributes

Id
   Optional.
URI
   Optional.
Type
   Optional.

Element Information

Number of occurrences        Unlimited
Parent elements              <ds:SignedInfo>
Child elements               <ds:Transforms>, <ds:DigestMethod>, <ds:DigestValue>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:RetrievalMethod> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:RetrievalMethod
   URI=anyURI
   Type=anyURI>
</ds:RetrievalMethod>

Attributes

URI
   Required.

Type
   Optional.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyInfo>
Child elements <ds:Transforms>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:RSAKeyValue> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:RSAKeyValue/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyValue>
Child elements <ds:Exponent>, <ds:Modulus>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:Signature> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:Signature Id=ID>

Attributes

Id
Optional.

Element Information

Number of occurrences  Unlimited
Parent elements
Child elements  <ds:SignedInfo>, <ds:SignatureValue>, <ds:KeyInfo>, <ds:Object>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:SignatureMethod> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:SignatureMethod Algorithm=anyURI>

Attributes

Algorithm
   Required.

Element Information

Number of occurrences Unlimited
Parent elements   <ds:SignedInfo>
Child elements    <ds:HMACOutputLength>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:SignatureProperties> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:SignatureProperties Id=ID>
</ds:SignatureProperties>

Attributes

Id
Optiona.

Element Information

Number of occurrences Unlimited
Parent elements
Child elements <ds:SignatureProperty>
<ds:SignatureProperty> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:SignatureProperty
  Target=anyURI
  Id=ID>
</ds:SignatureProperty>

Attributes

Target
  Required.

Id
  Optional.

Element Information

Number of occurrences Unlimited
Parent elements <ds:SignatureProperties>
Child elements None.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:SignatureValue> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:SignatureValue Id=ID>

Attributes

Id
Optional.

Element Information

**Number of occurrences**  Unlimited
Parent elements  <ds:Signature>
Child elements  <ds:SignedInfo>, <ds:SignatureValue>, <ds:KeyInfo>, <ds:Object>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:SignedInfo> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:SignedInfo Id=ID>

Attributes

Id
Optional.

Element Information

Number of occurrences Unlimited
Parent elements <ds:Signature>
Child elements <ds:CanonicalizationMethod>, <ds:SignatureMethod>, <ds:Reference>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:SPKIData> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:SPKIData/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:KeyInfo>
Child elements <ds:SPKISexp>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:Transform> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:Transform
   Algorithm= anyURI>

Attributes

Algorithm
   Required.

Element Information

Number of occurrences Unlimited
Parent elements <ds:Transforms>
Child elements <ds:XPath>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - Digital Sign
<ds:Transforms> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:Transforms/>

Attributes

None.

Element Information

Number of occurrences Unlimited
Parent elements <ds:Reference>
Child elements <ds:Transform>
<ds:X509Data> Element

[This feature was first implemented for MSXML 5.0. This is preliminary documentation and subject to change.]

<ds:X509Data/>

Attributes

None.

Element Information

**Number of occurrences**  Unlimited

Parent elements  
<ds:KeyInfo>
<ds:X509IssuerSerial>, <ds:X509SKI>,
<ds:X509SubjectName>, <ds:X509Certificate>,
<ds:X509CRL>, <ds:X509IssuerName>,
<ds:X509SerialNumber>

Child elements
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XML Schemas

Similar to data modeling in a relational database, a schema provides a way to define the structure of XML documents. A schema specifies the elements that can be used in XML documents along with the structure that these elements must follow in order to be valid for that specific schema.

In other words, a schema describes the vocabulary (rules or grammar) that complying XML documents must follow in order to be considered schema-valid with that particular schema. Validation of XML documents ensures that the external data conforms to the rules (grammar) required by the schema.

The following sections provide information about two types of schemas: XML Schema definition language (XSD) schemas, and XML-Data Reduced (XDR) schemas.

- [Getting Started with XML Schemas](#)
- [XSD Schema Developer's Guide](#)
- [XSD Schema Reference](#)
- [XDR Schema Developer's Guide](#)
- [XDR Schema Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
XML Schemas Overview

XML schemas are powerful and complex tools for creating structure and validating that structure in compliant XML documents. Schemas are XML files that typically have an .xsd file name extension. Schemas describe the contents of XML documents using valid XML; elements and attributes are declared using element and attribute elements.

An organization and its partners can benefit by using schemas to exchange data (through schema-compliant XML documents). Schemas are also useful for keeping XML documents consistent throughout an organization.

This section provides information to help you get started with XML schemas.

- Frequently Asked Questions about Schemas
- Comparing Schema Languages
- Version and Conformance
Frequently Asked Questions about Schemas

The following list includes frequently asked questions about schemas.

**What is the difference between a declaration and a definition?**

**What is the difference between XML Schema (XSD) and XDR schema?**

**What is the difference between a declaration and a definition?**

Declarations describe the content models (structure) of elements and attributes inside XML document instances. Definitions create new data types (simple and complex types).

**What is the difference between XML Schema (XSD) and XDR schema?**

The XML Schema definition language (XSD) is the current World Wide Web Consortium (W3C) specification for XML schemas. XML-Data Reduced (XDR) schema is an interim schema language offered to developers while the W3C worked on a draft implementation. MSXML versions 4.0 and later support both XML Schema (XSD) and XDR schema.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Comparing Schema Languages

This section compares schema languages to help you make decisions about implementing schemas into your XML applications.
Document Type Definitions (DTDs)

- DTDs specify content models for a set of XML documents.
- DTDs keep XML documents consistent.
- DTDs can be shared.
- DTDs provide a formal and complete definition of an XML vocabulary.
- DTDs are restricted to one DTD for each XML document.
Schemas

Like a DTD, a schema specifies the structure or model of a particular set of documents. Following are the benefits of using a schema language to describe XML document structure:

- Schemas use XML syntax.
- Schemas can be parsed by an XML parser.
- Schemas provide support for data types; the type of data contained in an element can be specified.
- Schemas allow global elements (elements that must be used the same way throughout the XML document) and local elements (elements that have a specific meaning in context).
- Schemas offer data type support (integer, string, Boolean), and the ability to create other data types.
Support for XDR schemas

The following products support XML-Data Reduced (XDR) schemas:

- Microsoft® BizTalk™ Server
- Microsoft SQL Server™ 2000
- Microsoft Office 2000
- Microsoft Internet Explorer version 5.0 and later

XDR schemas also have tools support in the following product:

- Extensibility's XML Authority
XSD Schemas

XML Schema definition language (XSD) schemas provide support for data types and for defining structures. This support allows you to create robust XML Schemas and validate XML instance documents against these schemas.

See Also

XDR Schema Developer's Guide
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
## Version and Conformance

MSXML versions 4.0 and later comply with the World Wide Web Consortium (W3C) 2001 XML Schema Recommendation.

<table>
<thead>
<tr>
<th>Version</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSXML</td>
<td>No support for schemas.</td>
</tr>
<tr>
<td>MSXML 2.0</td>
<td>Support for XML-Data Reduced (XDR) schemas.</td>
</tr>
<tr>
<td>MSXML 2.6</td>
<td>Support for XDR schemas.</td>
</tr>
<tr>
<td>MSXML 3.0</td>
<td>Support for XDR schemas.</td>
</tr>
<tr>
<td>MSXML 4.0</td>
<td>Support for XML Schema definition language (XSD) and XDR schemas.</td>
</tr>
<tr>
<td>MSXML 5.0</td>
<td>Support for XML Schema definition language (XSD) and XDR schemas.</td>
</tr>
</tbody>
</table>
# XSD Schema Developer's Guide

The following sections provide information about XML Schema definition language (XSD) schemas.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concepts</td>
<td>Provides conceptual information about XML Schemas.</td>
</tr>
<tr>
<td>How Do I?</td>
<td>Provides validation samples in Visual Basic, script, and C++.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Concepts

The following sections provide conceptual information about XML Schema definition language (XSD) schemas.

- Schema Structures
- Data Types in Schemas
- Using Namespaces in Schemas
- Validation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Schema Structures

Schema structures define and describe a class of XML documents. Schemas may also provide for the specification of additional document information, such as normalization and defaulting of attribute and element values.

Any application that consumes well-formed XML can use the World Wide Web Consortium (W3C) XML Schema: Structures specification to create and describe structures and constraints for XML documents.

Because schemas have the ability for self-documentation, they can be used to define, describe, and categorize XML vocabularies for classes of XML documents.

Structures allow you to describe content models for schemas.

This section covers the following topics.

- Definitions and Declarations
- Data Type Definitions
- Declaring Attributes
- Declaring Elements
- Defining Complex Types
- Content Models
- Notation Declarations
- Using Content from Other Schemas
- Documenting Schemas
- Using Other Namespaces
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Definitions and Declarations

There are two different types of markup in XML schemas that correspond to two parts of the specification: definitions and declarations.

Definitions create new types (simple and complex types).

Declarations describe the content models of elements and attributes. Declarations are an association between a name and the set of constraints for the specific declaration. The declarations are not types. There is a separate definition that specifies the nature of that type within the declaration.

An element whose content is a simple type can be defined in one element (such as Invoice) while elements with complex types must first be declared with two attributes, name and type. The name attribute is the name of the element; the type attribute is used to associate name with its type definition. It is the type definition that defines the content model for that type which any other element can also use. The complex type is defined as an element called complexType. It is also possible to nest the complexType element in the element declaration if it only applies to that one element, in which case the type attribute is not needed.

There are two kinds of type declarations that are used to create structures: simple declarations and complex declarations.
Simple Declarations

Simple declarations create simple types. You can create derived data types from base types. These derived types are created from types that are built into the XML Schema specification or from types that are based from another simple type.
Complex Declarations

Complex declarations create complex types. Complex types are used to describe the content model.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Data Type Definitions

Data type definitions are created from another base type using either extension or restriction.

Simple type definitions describe the content.

Complex type definitions describe the structure or the content model of the XML document.

The following can be complex data types:

- Elements that contain other elements.
- Elements that contain both elements and text.
- Elements that contain only text.
- Elements that are empty.

This section includes the following topics.

- Ur-Type Definitions
- Simple Type Definitions
- Complex Type Definitions
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Ur-Type Definitions

The set of type definitions for a schema form a hierarchical tree with a single root known as the Type Definition Hierarchy. The root of this tree is called a ur-type definition. The ur-type definition is unique because it can be used as a complex type or a simple type, depending on the context of the definition.

Ur-types are unconstrained. They are the base type for all simple and complex types.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Simple Type Definitions

A simple type definition is a set of constraints on the value space and lexical space of a data type. Constraints are in the form of a restriction on the base type or the specification of a list type that is constrained by another simple type definition.

Simple type definitions are for all the built-in primitive and derived data types that are present in schemas.

A simple type definition is a set of constraints on strings and information about their values. This applies to values of attribute elements or element elements that have no child elements. It applies to attribute values and text-only content of elements.

A simple type definition uses the `simpleType` element, attributes, and valid constraining facets.

Each simple type definition (built-in or derived) is a restriction of some particular simple base type definition. For built-in simple types, this is the simple version ur-type definition, `anySimpleType`, which is understood to be a restriction of the ur-type definition. Simple types may also be defined whose members are lists of items themselves constrained by some other simple type definition.
Deriving Simple Types Using Restriction

New simple types can be defined using restriction.

Example

The following example shows a simple type definition that is created using a restriction base. The restriction element constrains the values of the simple type definition to values that are positive integers, as defined by the World Wide Web Consortium (W3C) Datatypes specification. These values must also be under 100, as indicated by the maxExclusive element.

```xml
<xs:simpleType>
  <xs:restriction base="xs:positiveInteger">
    <xs:maxExclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```
Deriving Simple Types Using List

New simple types can be defined using list. Data types derived by list contain a white space-delimited list of values that conform to the base type. A list data type must be derived from an atomic data type.

Example

The following example shows a simple unbounded list of floating-point numbers.

```xml
<element name="decimalsList">
  <simpleType>
    <list itemType="decimal"/>
  </simpleType>
</element>
```

Example

The following example shows an acceptable use of the decimalsList data type in complying XML documents.

```xml
<decimalsList>-2.0 -3.0 8.6</decimalsList>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Complex Type Definitions

New complex types are defined using the complexType element. These definitions usually contain a set of element declarations, element references, and attribute declarations.

A complex type definition is a set of attribute declarations and content type that apply to the attributes and children of an element information item. Each complex type definition is either a restriction of a complex base type definition or an extension of a simple or complex base type definition or a restriction of the ur-type definition.

A complex type that extends another does so by having additional content model particles at the end of the other type definition's content model, or by having additional attribute declarations, or both.

Example

The following example shows a complex type definition for USAddress.

```xml
<xs:complexType name="USAddress">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="state" type="xs:string"/>
    <xs:element name="zip" type="xs:decimal"/>
  </xs:sequence>
  <xs:attribute name="country" type="xs:NMTOKEN" fixed="US"/>
</xs:complexType>
```

See Also

Declaring Attributes | Declaring Elements
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Declaring Attributes

Attributes are declared using the `attribute` element.

Example

The following example shows an attribute declaration for `blue`.

<xs:attribute name="blue" />
Referencing a Global Attribute Declaration

You can define attributes globally and use the `ref` attribute to reference global attributes.

**Example**

The following example shows a reference to the global attribute, blue.

```xml
<xs:element name="eyeColor" type="eyeColorType"/>
<xs:complexType name="eyeColorType">
  <xs:attribute ref="blue"/>
  <xs:attribute name="light"/>
</xs:complexType>
```

**Example**

The following example shows another reference to the global attribute, blue.

```xml
<xs:element name="contactLense" type="contactLenseType"/>
<xs:complexType name="ContactLenseType"/>
  <xs:attribute ref="blue"/>
</xs:complexType>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Declaring Elements

Elements are declared using the **element** element.

**Example**

The following example shows an element declaration for *Software*.

```xml
<xs:element name="Software" />
```
Referencing a Global Element Declaration

If you declare the complex type for this element at the top of the document (global declaration), you can reference this element in the remainder of the schema using the ref attribute.

Example

The following example shows an element declaration for Software element.

```xml
<xs:element name="Software" />
  <xs:complexType>
    <xs:element name="title" />
    <xs:element name="company" />
  </xs:complexType>
</xs:element>
```

The following example shows a reference to the global element declaration for the Software element.

```xml
<xs:element name="Catalog">
  <xs:complexType>
    <xs:element ref="Software"/>
  </xs:complexType>
</xs:element>
```

Elements without a type defined will take a ur-type definition.

See Also

Ur-Type Definitions
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Defining Complex Types

Definitions of complex types are created using the `complexType` element, its attributes, and any valid facets. Usually, complex types will contain a set of element declarations, attribute declarations, and element references.

Example

The following example creates a complex type definition for the `Client` element to describe the content model.

```xml
<xs:element name="Client">
  <xs:complexType>
    <xs:attribute name="CompanyName" type="xs:string"/>
    <xs:attribute name="City" type="xs:string"/>
    <xs:attribute name="State" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

See Also

[Content Models](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Content Models

A content model is the formal description of the structure and allowable content of an element that is used to validate an XML document. Schema content models offer more control on element occurrences than DTD content models. In addition, schema content models allow the validation of mixed content.

A content model can restrict a document to a certain set of element types and attributes, describe and constrain the relationships between these different components, and uniquely identify specific elements. Sharing a content model allows businesses to exchange structured information.

This section includes the following topics.

- Groups
- Named Model Groups
- Attribute Groups
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Groups

There are three group elements: choice, sequence, and all.
group element

The group element groups a set of element declarations so that they can be incorporated into complex type definitions. The group element is limited to the top level of the content model.
choice element

The **choice** element allows one and only one of the elements contained in the group to be present within the containing element.
sequence element

The sequence element requires the element in the group to appear in the specified sequence within the containing element.
**all element**

The *all* element allows the elements in the group to appear (or not appear) in any order in the containing element.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Named Model Groups

The **group** element is used both for the definition of a group and for any reference to a named group. You can use a model group to define a set of elements that can be repeated through the document. This is useful for building a complex type definition. Named model groups can be further defined by using `sequence`, `choice`, or all child elements.

A named model group consists of element declarations, wildcards, and other model groups. Named model groups must have a **name** attribute and are declared at the top level of the schema.

An unnamed group element is not declared at the top level and must have a `ref` attribute that references an existing named group. Unnamed groups must not have a **name** attribute. A named group can be referenced, multiple times, using an unnamed group in the XML Schema document.

**Example**

The following example creates a named model group, `stationeries`, that allows a limited set of 1-5 books and 1-5 pens to appear in the group.

```xml
<xs:group name="stationeries">
  <xs:choice>
    <xs:element name="books" minOccurs="1" maxOccurs="5"/>
    <xs:element name="pens" minOccurs="1" maxOccurs="5"/>
  </xs:choice>
</xs:group>
```
Referencing a Global Group Declaration

You can use the **ref** attribute to reference a group by name in order to use the group description.

**Example**

The following example references a model group, stationeries, inside of the `choice` element.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:group name="stationeries">
    <xs:sequence>
      <xs:element name="books" minOccurs="1" maxOccurs="5" />
      <xs:element name="pens" minOccurs="1" maxOccurs="5" />
    </xs:sequence>
  </xs:group>

  <xs:element name="Order">
    <xs:complexType>
      <xs:choice>
        <xs:element name="table" minOccurs="1" maxOccurs="5" />
        <xs:group ref="stationeries" minOccurs="1" />
      </xs:choice>
    </xs:complexType>
  </xs:element>

</xs:schema>
```
Attribute Groups

Attribute groups are useful for grouping together attributes.

Example

The following example creates an attribute group, testMethods.

```xml
<xs:attributeGroup name="testMethods" />
  <xs:attribute name="writeVisualBasicCode" use="required" />
  <xs:attribute name="blackbox" use="optional" />
  <xs:attribute name="writeScriptCode" use="required" />
</xs:attributeGroup>
```
Referencing a Global Attribute Group Declaration

If you want to reference the group of attributes for a full name on an element, you must add a reference to it using the `ref` attribute of the `attributeGroup` element.

**Example**

The following example references a global attribute group, `testMethods`.

```xml
<xs:element name="testSuite">
   <xs:complexType>
      <xs:attributeGroup ref="testMethods" />
   </xs:complexType>
</xs:element>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Notation Declarations

Notation declarations associate a name with an identifier for a notation. The notation declarations are not validated. They are referenced when validating strings that are members of NOTATION simple type definitions are called in the XML document.

The `public` attribute of the `notation` element takes a public identifier from the ISO 8879 specification and system attribute takes a URI reference.

Example

The following example associates the jpeg file with a system executable, viewer.exe.

The content of this element would be a binary definition of the jpeg.

```xml
<xs:notation name="jpeg" public="image/jpeg" system="viewer.exe" />
<xs:element name="picture">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
        <xs:attribute name="pictype"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

The following example shows an element, `picture`, in an XML document.

```xml
<picture pictype="jpeg">...</picture>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Using Content from Other Schemas

The **any** element is used to add an element from another namespace to the schema. It allows any element from the specified namespace(s) to appear in the containing schema. The content of the **any** element can appear as a child element of **complexType, group, sequence, all, or choice** elements.

**Example**

The following example allows **catalog** element to contain any well-formed XML data that appears in the **fabrikam** catalog.

```xml
<xs:element name="catalog">
  <xs:complexType>
    <xs:any namespace="http://www.fabrikam.com/catalog"
           minOccurs="0" maxOccurs="unbounded"
           processContents="skip" />
  </xs:complexType>
</xs:element>
```

**See Also**

[Using Namespaces in Schemas](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Documenting Schemas

Because an XML schema is well-formed XML, you can include comments using XML comment syntax.

<!-- This is a comment -->

The XML parser does not pass comments to the application.

XML schema has three elements for schema meta data, for both application use and human readers.

Annotations allow you to make schemas self-documenting. By attaching a style sheet to the schema, you can create a readable version that explains the use of your schema.
Annotations

Annotations can appear anywhere in a schema to explain any element or attribute or type definition.

annotation element

The annotation element is the parent element for appinfo and documentation elements. This element can appear anywhere in the schema and allows you to document or comment on parts of your schema.

appinfo element

The appinfo element provides information for external applications, such as XML parsers. This element must be a child element of the annotation element.

documentation element

The documentation element allows you to place comments for human readers, such as copyright information, instructions about model group descriptions, and so on. This element must be child element of the annotation element.

Example

The following example shows an annotation that describes the purpose of the schema.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>Schema for processing submitted applications</xs:appinfo>
    <xs:documentation>
      Submitted applications for Human Resources department
    </xs:documentation>
  </xs:annotation>
</xs:schema>
```

Example
The following example shows an annotation to explain the purpose and application use of the relocationStatus element.

```xml
<xs:element name="relocationStatus">
  <xs:complexType>
    <xs:annotation>
      <xs:appinfo>
        If relocationStatus is Seattle, no further processing for r
      </xs:appinfo>
      <xs:documentation>
        This element is for determining relocation status to determ
      </xs:documentation>
    </xs:annotation>
  </xs:complexType>
</xs:element>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
Using Other Namespaces

The **include** element allows you to reference and use schema components from external namespaces so that you do not have to redefine elements in every schema that requires the same existing content model or data type. It is possible to create new schemas that borrow or leverage from existing schemas.

**Example**

The following example shows the definitions and declarations from the `humanresources.xsd` schema are available to the containing schema through the use of the **include** element.

```xml
<xs:schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema-datatypes"
    targetNamespace="http://www.example.com">
    <xs:include xmlns:myschema="http://www.myschema.com/humanresources.xsd"/>
</xs:schema>
```

Alternatively, if you want to merge data from documents marked up according to two schemas, you could create a schema that would validate a new XML document (XML document instances). In the instance of a document marked up according to this schema, you only need to declare this schema (not the included schema).
Multiple Schemas

Multiple schemas from the same target namespace can be nested. This is useful if you want to create new XML documents based on data coming from several sources.

Example

The following example shows a schema for cookbooks that includes the content models from the recipebooks schema. The recipebooks schema describes content models for desserts and wines.

```xml
<xs:schema xmlns="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.com/schemas/cookbooks">
  <xs:import schemaLocation="http://www.example.com/schemas/recipebooks.xsd"/>
  <xs:element name="catalog" type="catalogType"/>
  <xs:complexType name="catalogType">
    <xs:sequence>
      <xs:element name="desserts" type="cookbooks:dessertsType" />
      <xs:element name="wines" type="cookbooks:winesType" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

See Also

[Using Namespaces in Schemas](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Data Types in Schemas

Schemas are powerful tools for creating structures because they support a rich set of data types.

The following topics describe the data types in schemas.

- Primitive and Derived Data Types
- Atomic, List, and Union Data Types
- Anonymous and Named Data Types
- Parts of Data Types
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Primitive and Derived Data Types

There are two basic kinds of data types: primitive data types and derived data types.
Primitive Data Types

Primitive data types are those that are not defined in terms of other data types. Because primitive data types are the basis for all other types, they cannot have element content or attributes. However, they can contain values. Primitive data types are built into XML schemas. Examples of primitive data types are string, float, decimal, anyURI, and QName.

For more information, see Primitive XML Data Types.
Derived Data Types

Derived data types are those that are defined in terms of other data types, called base types. Derived types may have attributes, and may have element or mixed content. Instances of derived types can contain any well-formed XML that is valid according to their data type definition. They may be built-in or user-derived. Base types can be primitive types or derived data types. Derived data types are created using extension and restriction facets. They can be built-in or user-derived data types.

New types may be derived from either a primitive type or another derived type. For example, integers are a subset of real numbers. Therefore, the XML schema integer type is derived from the decimal number type, which is its base type. You can also derive an even more restricted type of integer by using the minInclusive and maxInclusive elements. This is an example of a simple type definition.

Example

The following example shows a simpleType element that defines a derived data type, integer, that is restricted to negative values.

```
<simpleType name="negativeInteger">
  <restriction base="xsi:integer">
    <minInclusive value="-100"/>
    <maxInclusive value="-1"/>
  </restriction>
</simpleType>
```

As with the primitive types, the World Wide Web Consortium (W3C) has defined a set of built-in derived data types. These types are part of the W3C XML Schemas specification.

For a list of built-in derived types, see Derived XML Data Types.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Atomic, List, and Union Data Types

Atomic Data Types

Atomic data types have values that cannot be divided or broken down further. Atomic data types can be either primitive or derived.

Numbers and strings are atomic data types because their values cannot be described using smaller parts. XML schema has no concept of a character as a data type and a string is an atomic primitive type.

Example

The following is an example of an atomic data type of string.

<atomString>This cannot be divided further because this is a string.<br/>
</atomString>

Example

The following is an example of an atomic derived type that is a date that can be derived from the string primitive type.

<atomDate>03-18-2001</atomDate>
List Data Types

List data types have values that consist of a finite length sequence of values of an atomic data type. List data types are comprised of sequences of atomic data types. Each part of a list data type carries a meaningful value.

There are three built-in list types: **NMTOKENS**, **IDREFS**, and **ENTITIES**.
Union Data Types

Union data types are those whose value spaces and lexical spaces are the union of value spaces and lexical spaces of two or more other data types. A union data type enables an element or attribute value to be one or more instances of a type drawn from the union or combination of multiple atomic and list types. Union types have a `memberTypes` attribute value that is a list of all the types in the union. Union types can have `pattern` and `enumeration` facets.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Anonymous and Named Data Types

Data types can be either anonymous or named. Named types allow you to use the type in other parts of the schema.
Anonymous Data Types

Anonymous data types are referenced only once and contain few constraints. These data types are identified by the lack of a type identifier in the element. The use of anonymous types saves the overhead of naming and explicitly referencing elements.
Named Types and Anonymous Types

Simple and custom types can be named so that they can be reused in other places in the schema. Anonymous types are used only within the element in which the definition appears. You are not required to name every custom type that you create. You can omit the cross reference between the element and type for types that are used once for an element.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Parts of Data Types

Data types consist of three parts:

- value space
- lexical space
- facets (fundamental or constraining)
Value space

Value space refers to a set of literal values. Derived types inherit their value space from the base type; derived types are restricted by their base types.
Lexical space

Lexical space refers to the representation of literal values. Lexical space is the set of string literals that represent the values of data types. These literals always consist of text characters from any XML-legal subset of Unicode character set.
Facets

Facets are properties of the value space. A facet is one of the defining properties of a data type that distinguishes it from other data types. The World Wide Web Consortium (W3C) defines a facet as a single defining aspect of a value space. Facets provide definition for simple types. A facet is a restriction to a data type.

Facets of a data type serve to distinguish aspects of one data type that differ from other data types. There are two types of facets: fundamental facets and nonfundamental (constraining) facets. Fundamental facets define the data type. Nonfundamental or constraining facets place constraints or restrictions on the data type.

**Fundamental Facets**

There are five fundamental facets.

**Equal**
- Different values can be compared and determined to be equal or not.

**Ordered**
- For some data types, defined relationships exist between values.

**Bounded**
- Order data types may be constrained to a range of values.

**Cardinality**
- The number of values within the value space.

**Numeric**

**Constraining Facets**

Constraining facets limit the values of a derived data type.

There are several constraining facets that can be applied to a derived data type.

**enumeration**

**fractionDigits, totalDigits**

**length, minLength, maxLength**
pattern

minExclusive, maxExclusive, minInclusive, maxInclusive

whiteSpace
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Using Namespaces in Schemas

Schemas are a collection of rules (also referred to as a grammar or vocabulary) that consist of type definitions (simple and complex), as well as element and attribute declarations. Because XML documents can have several vocabularies to describe different elements and attributes, the use of namespaces and prefixes removes ambiguity for element and attribute declarations. Distinguishing between element and attribute names for each namespace is essential when you use schemas from more than one namespace.

A namespace is usually a string used to differentiate between the namespaces, such as "urn:www.microsoft.com", "http://www.microsoft.com", "http://www.w3.org/2001/XMLSchema", and "uuid:1234567890".
XML Schema Preamble

An XML schema consists of a preamble that is followed by declarations.

Example

The following syntax uses the schema element and references three commonly used XML vocabularies using the xmlns attribute.

```xml
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema-datatypes"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="1.0">
</schema>
```

The schema element uses its attributes to identify any external namespaces that are to be used by the schema and all of its child elements.

The first xmlns attribute refers to the standard Worldwide Web Consortium (W3C) XML Schema Recommendation that is used by most XML schemas. The other xmlns attributes include references to the basic XML-Data Reduced (XDR) schema elements such as element, attribute, complexType, group, simpleType, and so forth.
Default namespaces

For an XML Schema definition language (XSD) schema, the schema has a top-level schema element. The schema element definition must include the following namespace.

http://www.w3.org/2001/XMLSchema

It is not necessary to use xs or xsi as the namespace identifiers. You can use any prefix that does not conflict with the W3C namespaces that identify the XML schema specifications, however, xs and xsi are the recommended conventions.

If you identify the W3C namespace as the default namespace, you do not need to include the prefix in every XML schema declaration and can use unqualified names such as <element>. For example, <xs:element> would be unnecessary if the default namespace is declared as http://www.w3.org/2001/XMLSchema.

Example

The following syntax instructs the XML parser to use XML Schema, <xs:schema>, to validate the structure and content of an element.

<x:schema
 <xs:element name="Price" type="msc:mydecimal"/>
</xs:schema>

For XML-Data Reduced (XDR) schemas, the Schema element must include the following namespace. Note that Schema is initial capitalized for XDR schemas.

urn:schemas-microsoft-com:xml-data

To use XDR schema data types, the Schema element must include the following namespaces.

urn:schemas-microsoft-com:xml-data
urn:schemas-microsoft-com:datatypes

Example
The following syntax instructs the XML parser to use XDR schema, \texttt{x-schema}, to validate the structure and content of an element.

\begin{verbatim}
<BOOK xmlns="x-schema:http://www.microsoft.com/BookInfo.xml">
  <TITLE>Creepy Crawlies</TITLE>
  <PRICE currency="US Dollar">22.95</PRICE>
</BOOK>
\end{verbatim}

The \texttt{x-schema} identifier at the start of the namespace URI instructs the XML parser to treat the reference as a valid reference to an XDR schema; it retrieves the schema from the URL and validates against it.

For more information about declaring namespaces, namespace prefixes, and namespace scope, see \textit{Using Namespaces in Documents}.
Specifying the Version

The `version` attribute is informational and represents the version number of the schema. You can use the `version` to ensure that the XML application is using a specific version of the schema to validate XML documents. You can use multiple schema versions: one for development, another for testing, another for production, and others for individual international markets or regional markets.

Example

The following syntax instructs the XML parser to use version "1.0" of the `myschema` namespace to validate the structure and content of an element that uses the `myschema` namespace.

```xml
<xs:schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema-datatypes"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:myschema="http://myschema.com/schema1"
    version="1.0">
</xs:schema>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Validation

Validation is the process of comparing an XML document to a corresponding schema. If an XML document conforms to all of the rules in the schema, the document is considered valid. As an introduction to validation, this section provides the following topics:

- Validation Overview
- Deterministic and Non-Deterministic Schemas
- Using Inline Schemas (XSD)

See Also

How Do I? (XSD) | Validate an XML Document or Fragment Using C/C++ (DOM) | Validate an XML Document or Fragment Using Visual Basic (DOM) | IXMLDOMSchemaCollection/XMLSchemaCache

Other Resources

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Validation Overview

A schema defines the structure of XML documents. It provides details about the content model: which elements it contains and in what order, what its content can be, and which content these attributes can contain.

A schema verifies that the incoming XML documents are in the expected format.

An XML schema can be used to:

- Validate the content of an XML document.
- Determine whether the XML document is a valid instance of the vocabulary (grammar or rules) expressed by the XML schema.
- Describe the vocabulary for use by others creating XML documents.
- Define the elements that can appear within an XML document and the attributes that can be associated with an element.
- Define whether an element is empty or can include text.
- Define a default value for an attribute.
- Define elements that can contain child elements.
- Define the sequence for child elements that appear in an element.
- Define the number of child elements.
Writing validation rules

The following can be used to write validation rules.

- Document Type Definition (DTD)
- XML-Data Reduced (XDR) schema
- XML Schema definition language (XSD) schema

See Also

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Deterministic and Non-Deterministic Schemas

Validation depends on whether the schema is deterministic or non-deterministic.

A deterministic schema is a schema that is not ambiguous. This means that the parser used by the Schema Object Model (SOM) can determine the sequence in which elements should occur for an XML document to be valid.

It is possible for an XML Schema to be ambiguous, or non-deterministic. A schema is non-deterministic if the parser is unable to clearly determine the structure to validate with the schema. When validation is attempted on a non-deterministic schema, the parser generates an error.
Deterministic Schema

The following deterministic XML Schema specifies that a valid document must contain an element named root that has the following content:

- An element, apple, followed by an element, berry
  
or
- An element, coffee, followed by an element, dairy

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="root" type="rootTypes" />
  <xs:complexType name="myKitchen">
    <xs:choice>
      <xs:sequence>
        <xs:element name="apple"/>
        <xs:element name="berry"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="coffee"/>
        <xs:element name="dairy"/>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

Using the preceding schema, the parser is able to follow the schema logic when it processes an instance document. If the element root is followed in the document by the element apple, the parser determines that it has encountered the first part of the sequence of apple and berry. Likewise, if the parser encounters the element coffee after encountering the root element, the parser determines that it has encountered the first part of the sequence of coffee and dairy. Any other ordering of elements in the instance document is invalid according to this schema.
Non-Deterministic Schema

With a non-deterministic schema, the parser cannot determine a sequence for the elements in the document being processed. The following is an example of a non-deterministic schema:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="root" type="rootTypes"/>
  <xs:complexType name="myKitchen">
    <xs:choice>
      <xs:sequence>
        <xs:element name="apple"/>
        <xs:element name="berry"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="apple"/>
        <xs:element name="coffee"/>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

Using the preceding schema, when the parser encounters the element `root` followed by an element `apple`, it is unable to determine, without looking ahead to the next element, whether the `apple` element is the first part of the sequence `apple` and `berry` or the first part of the sequence of `apple` and `coffee`. Because the parser used by the SOM does not perform forward checking, the parser generates the following error message when validation is attempted using a non-deterministic schema:

Content model must be deterministic

See Also

XML Schema Reference (XSD)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
Using Inline Schemas (XSD)

Beginning with Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office, support has been implemented to allow you to use inline XML schemas that conform to the XML Schemas (XSD) specification.

Inline schemas can be useful when it is inconvenient to physically separate the schema and the instance XML document. For example, if you generate an XSD schema from data retrieved from a database, it might be more convenient to put the schema and the data in the same file.

**Note** Support for inline XDR schemas was previously implemented in MSXML 3.0 and 4.0. For more information, see [Using Inline XDR Schemas](#).

In MSXML 5.0 and later, inline XSD schema support is enabled by default whenever you create a new DOM document or SAX stream. This support includes a compiling step for any inline schema contents, prior to subsequent parse/validate operations. This is done to ensure that the inline schema fragment is valid as an XML schema, before it is applied or used to validate the non-schema XML remainder of the DOM document or SAX stream.

The ability to control whether inline schemas are compiled and used to validate their host documents can be set in one of two ways, depending on whether you are working with the DOM or the SAX APIs:

- For DOM programming, you can turn off the second-level `UseInlineSchema` property on the current `DOMDocument` object.
- For SAX programming, use the `putFeature` method to turn off the "use-inline-schema" feature on the `SAXXMLReader` object.

If inline schema processing is turned off, inline schemas are treated no differently from any other XML document fragment, and any special meaning they have for validation is ignored.

**Important** Because inline schema compilation can take place even when validation is turned off for the DOM or SAX instance, some parsing errors
can occur if you compile inline schemas when you do not intend to validate the document. To avoid or resolve such errors, turn off inline schema support using the appropriate DOM or SAX mechanism, as mentioned above.

Inline schemas are referenced and handled in the same way as external schemas that are declared using either the `schemaLocation` attribute or the `noNamespaceSchemaLocation` attribute to specify a URI or path location to the external schema file.

Just like with external schemas, documents and document fragments validated using inline schemas must be identified by their namespace, which should match the `targetNamespace` attribute in the schema. For example, a schema with a target namespace of "http://example.com" is applied only to those fragments that also belong to the "http://example.com" namespace.

If a schema is used without `targetNamespace`, it is applied to the fragments, which belong to the default namespace. By design, inline schemas can never be applied to the root element.

In the following section, we will look at examples of how inline XSD schemas differ from external XSD schemas that reference an external location, such as another file or a schema cache.

**Examples**

This section provides the following examples of using inline schemas.

- [Example 1: Comparing an Inline to an External Schema](#)
- [Example 2: Adding a Namespace to the Schema](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Example 1: Comparing an Inline to an External Schema

To get started using inline schemas, we will look at an abridged version of the sample XML file (books.xml) (use-inline.xml). This file has been modified to include an inline schema, which will be used to validate the contents of any <book> element nodes that appear in the non-schema portion of the XML document. The inline schema is nested within an <xsd:schema> element.

Note By design, inline schemas are never applied to the root element when validating document contents.

File: use-inline.xml

```xml
<?xml version="1.0"?>
<catalog xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- START OF SCHEMA -->
<xsd:schema>
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="genre" type="xsd:string"/>
        <xsd:element name="price" type="xsd:float"/>
        <xsd:element name="publish_date" type="xsd:date"/>
        <xsd:element name="description" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<!-- END OF SCHEMA -->
<book id="bk101">
  <author>Gambardella, Matthew</author>
  <title>XML Developer's Guide</title>
  <genre>Computer</genre>
  <price>44.95</price>
  <publish_date>2000-10-01</publish_date>
  <description>An in-depth look at creating applications with XML.</description>
</book>
```
The inline schema version of the sample document is treated exactly the same as the external schema version of the same file (use-external.xml) provided below. In the external version, the only differences are the following:

- At the `<catalog>` element, the XML schema instancing ("xsi:" ) namespace is declared.
- At the `<book>` element, the xsi:schemaLocation schema instance attribute is used to declare that the contents of the `<book>` element can be validated against a schema contained in an external XSD file (books1.xsd), located in the same directory.

File: use-external.xml

```xml
<?xml version="1.0"?>
<catalog xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <book xsi:schemaLocation="book books1.xsd" id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
</catalog>
```

File: books1.xsd

```xml
<?xml version="1.0"?>
<xsd:schema>
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="genre" type="xsd:string"/>
        <xsd:element name="price" type="xsd:float"/>
        <xsd:element name="publish_date" type="xsd:date"/>
        <xsd:element name="description" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
File: validate.vbs

'Create an XML DOMDocument object.
Dim xmldom, sXmlFile
Set xmldom = CreateObject("MSXML2.DOMDocument.5.0")

'Get the name of the file to validate as user input.
sXmlFile = InputBox("Enter the name of the file to validate:", _
Wscript.ScriptName, "use-inline.xml")

'Load and validate the specified file into the DOM.
xmldom.async = False
xmldom.validateOnParse = True
xmldom.load sXmlFile

'REturn validation results in message to the user.
If xmldom.parseError.errorCode <> 0 Then
    MsgBox xmldom.parseError.errorCode & " " & _
    xmldom.parseError.reason
Else
    Dim node
    xmldom.setProperty "SelectionLanguage", "XPath"
    Set node = xmldom.selectSingleNode("//book")
    MsgBox "Validation successful" & vbCrLf & _
    "======================" & vbCrLf & node.xml
End If

Try It!

1. Copy the inline schema version of the sample XML file above and paste it into a text file. Save the file as use-inline.xml.
2. Copy the external schema version of the sample XML file from above. Paste it into a text file, and save the file as use-external.xml in the same directory where you saved use-inline.xml.
3. Copy the XSD schema for the <book> element from above. Paste it into a text file, and save the file as books1.xsd in the same directory where you saved the file in the previous steps.
4. Copy the VBScript code above. Paste it into a text file, and save the file as validate.vbs. Use the same directory as you did in the previous steps.
5. Double-click validate.vbs.

An input box will appear that allows you to specify the name of the file to validate.
6. Click **OK** to validate the default file (**use-inline.xml**).
7. Repeat the previous step but substitute the version with the external XSD schema (**use-external.xml**) instead.

Both the inline and external versions of the XML sample file should validate successfully and display the same output.

**See Also**

[Using Inline Schemas (XSD)](#) | [Example 2: Adding a Namespace to the Schema](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Example 2: Adding a Namespace to the Schema

In the last example, we allowed the schemas to be applied using the default namespace. This was done in order to keep it simple and to contrast clearly what differentiates using inline schemas from using external schemas.

But what if you need to incorporate a schema that validates to another namespace? In this example, we will look at the changes needed to modify the files you created in Example 1 so that they are namespace aware. For the purposes of this example, we will determine that the <book> element needs to exist in a separate "x" namespace instead.

File: inline-namespaced.xml

The following is an altered version of the use-inline.xml file you created in Example 1. The changes are highlighted in bold.

```xml
<?xml version="1.0"?>
<catalog xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:x="urn:book">
<!-- START OF SCHEMA -->
<xsd:schema targetNamespace="urn:book">
    <xsd:element name="book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="author" type="xsd:string"/>
                <xsd:element name="title" type="xsd:string"/>
                <xsd:element name="genre" type="xsd:string"/>
                <xsd:element name="price" type="xsd:float"/>
                <xsd:element name="publish_date" type="xsd:date"/>
                <xsd:element name="description" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
<!-- END OF SCHEMA -->
<x:book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
```
File: external-namespaced.xml

The following is an altered version of the use-external.xml file you created in Example 1. The changes are highlighted in bold.

```xml
<?xml version="1.0"?>
<catalog xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:x="urn:book">
  <x:book xsi:schemaLocation="book books2.xsd" id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </x:book>
</catalog>
```

File: books2.xsd

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="urn:book">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="genre" type="xsd:string"/>
        <xsd:element name="price" type="xsd:float"/>
        <xsd:element name="publish_date" type="xsd:date"/>
        <xsd:element name="description" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
File: validate2.vbs

The following is an altered version of the validate.vbs file you created in Example 1. The changes are highlighted in bold.

'Create an XML DOMDocument object.
Dim xmldom, sXmlFile
Set xmldom = CreateObject("MSXML2.DOMDocument.5.0")

'Get the name of the file to validate as user input.
sXmlFile = InputBox("Enter the name of the file to validate:",_ 
Wscript.ScriptName, "inline-namespaced.xml")

'Load and validate the specified file into the DOM.
xmldom.async = False
xmldom.validateOnParse = True
xmldom.load sXmlFile

'Return validation results in message to the user.
If xmldom.parseError.errorCode <> 0 Then
   MsgBox xmldom.parseError.errorCode & " " & _
   xmldom.parseError.reason
Else
   Dim node
   xmldom.setProperty "SelectionNamespaces", "xmlns:x='urn:book'"
   xmldom.setProperty "SelectionLanguage", "XPath"
   Set node = xmldom.selectSingleNode("//x:book")
   MsgBox "Validation successful" & vbCrLf & _
      "=====================" & vbCrLf & node.xml
End If

Try It!

1. Copy the namespaced inline version of the sample XML file above and paste it into a text file. Save the file as inline-namespaced.xml.
2. Copy the namespaced external version of the sample XML file from above. Paste it into a text file, and save the file as external-namespaced.xml in the same directory where you saved inline-namespaced.xml.
3. Copy the XSD schema for the <book> element from above. Paste it into a text file, and save the file as books2.xsd in the same directory where you saved the file in the previous steps.
4. Copy the VBScript code above. Paste it into a text file, and save the file as validate2.vbs. Use the same directory as you did in the previous steps.
5. Double-click validate2.vbs.
An input box will appear that allows you to specify the name of the file to validate.

6. Click **OK** to validate the default file (**inline-namespaced.xml**).
7. Repeat the previous step, but substitute the version with the external XSD schema (**external-namespaced.xml**) instead.

Both the inline and external versions of the XML sample file should validate successfully and display the same output.

**See Also**

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schem
## How Do I?

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Description</th>
</tr>
</thead>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Validate an XML Document Against an XML Schema Using Visual Basic

You can use Visual Basic to validate an XML document against an XML Schema definition language (XSD) schema. To validate a document, add the schema to the XMLSchemaCache object, set the schemas property of a DOMDocument object to reference the schema in the XMLSchemaCache object, and then load the XML file you want to validate into the DOMDocument object. Validation is performed when the document is loaded into the DOMDocument object. Validation errors are handled using the parseError property of the DOMDocument object. This topic provides a quick example of how to validate a document against a schema and consists of the following.

books.xsd
The XML Schema that is used to validate the books.xml file.
books.xml
The file that is loaded and validated against the books.xsd schema.
Microsoft Visual Basic validation code
Creates an XMLSchemaCache object, adds the schema to it, and sets the schemas property of the DOMDocument object to reference the schema in the XMLSchemaCache object. In addition, the code loads the document into the DOMDocument object and returns any validation errors that occur.

To get started

1. Open Visual Basic 6.0, and in the New Project dialog box, double-click Standard EXE.
2. On the Project menu, click References.
3. In the Available References list, select Microsoft XML,v5.0, and then click OK.
4. Add a Command button to Form1.
5. Save the project to a folder on your hard drive.
The XML Document (books.xml)

The books.xml file used in this example, shown below, is a modified version of the books.xml file used throughout the MSXML SDK. Notice that "urn:books" is declared as the default namespace for this document.

If you want to run the example in this topic, copy the following code to a text editor, such as Notepad, and then save the file as books.xml.

```xml
<?xml version="1.0"?>
<x:catalog xmlns:x="urn:books">
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
</x:catalog>
```
The XML Schema (books.xsd)

The following document shows the books.xsd schema used in this example.

If you want to run the example in this topic, copy the following code to a text editor, such as Notepad, and then save the file as books.xsd.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" 
targetNamespace="urn:books"
xmlns:b="urn:books">
  <xs:element name="catalog" type="b:CatalogData"/>

  <xs:complexType name="CatalogData">
    <xs:sequence>
      <xs:element name="book" type="b:bookdata" minOccurs="0" 
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="bookdata">
    <xs:sequence>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="genre" type="xs:string"/>
      <xs:element name="price" type="xs:float"/>
      <xs:element name="publish_date" type="xs:date"/>
      <xs:element name="description" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```
Visual Basic Validation Code

The following code snippet shows how you can perform validation on the books.xml document. This code creates an `XMLSchemaCache50` object, adds the namespace URI declaration ("urn:books") and the schema (books.xsd) to the object, and then references the schema using the `schemas` property of the `DOMDocument` object. Validation is performed when the books.xml file is loaded into the `DOMDocument` object. Validation errors are returned using the `parseError` property of the `DOMDocument` object.

To run the example

1. Copy the code shown below to the Command1_Click procedure.

   ```vba
   Private Sub Command1_Click()
   'Create a schema cache and add books.xsd to it.
   Dim xmlschema As MSXML2.XMLSchemaCache50
   Set xmlschema = New MSXML2.XMLSchemaCache50
   xmlschema.Add "urn:books", App.Path & "\books.xsd"

   'Create an XML DOMDocument object.
   Dim xmldom As MSXML2.DOMDocument50
   Set xmldom = New MSXML2.DOMDocument50

   'Assign the schema cache to the DOM document.'schemas collection.
   Set xmldom.schemas = xmlschema

   'Load books.xml as the DOM document.
   xmldom.async = False
   xmldom.Load App.Path & "\books.xml"

   'Return validation results in message to the user.
   If xmldom.parseError.errorCode <> 0 Then
     MsgBox xmldom.parseError.errorCode & " " & _
     xmldom.parseError.reason
   Else
     MsgBox "No Error"
   End If

   End Sub
   ```

2. On the Visual Basic toolbar, click **Start**, and then click the **Command1**
button on form1.

When you run the example, it returns a "No Errors" message box.
Testing the Validation Code

To generate an error message for the validation code, change the books.xml document so the title element appears directly after the catalog element, as shown in bold in the following example. This makes the document invalid according to the books.xsd schema. Save the books.xml file and then run the Visual Basic code. The code returns the following error message:

"1072898028 Element content is invalid according to DTD/Schema. Expecting: book."

<?xml version="1.0"?>
<x:catalog xmlns:x="urn:books">
  <title>2000-10-01</title>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
</x:catalog>

See Also

XML Schema Examples | XML Schema Element Map | XML Schema Elements | XML Data Types Reference | Primitive XML Data Types | Derived XML Data Types | Data Type Facets | IXMLDOMSchemaCollection/XMLSchemaCache

Other Resources

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
Validate an XML Document Against an XML Schema Using Script

To validate an XML document file to an XML Schema definition language (XSD) schema file from within Windows script you use the schemaLocation attribute. The example below demonstrates this validation technique using the following three files:

- **doc.xml**: The XML file to be validated.
- **doc.xsd**: The XML Schema file against which doc.xml will be validated. This file is referenced in doc.xml with the schemaLocation attribute.
- **validate.js**: A Windows script file used to perform the validation and return the results as an alert message.

**To validate to XML Schema using Windows script**

1. Open Notepad
2. Create the three files: **doc.xml**, **doc.xsd**, and **validate.js**.

   To create the files, copy each of the following samples and paste them to the Notepad window. Save them all in the same folder, using the names provided.

   **doc.xml**

   ```xml
   <?xml version="1.0"?>
   <x:doc xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xmlns:x='http://xsdtesting'
       xsi:schemaLocation='http://xsdtesting doc.xsd'>
   <x:notDeclared/>
   </x:doc>
   ```

   **doc.xsd**

   ```xml
   <?xml version="1.0"?>
   <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'
       targetNamespace = "http://xsdtesting"
       xmlns:x = "http://xsdtesting"
       elementFormDefault = "qualified">
   ```
validate.js

var x = new ActiveXObject("MSXML2.DOMDocument.5.0");
x.async = false;
x.validateOnParse = true;
x.resolveExternals = true;
x.load("doc.xml");
if (x.parseError.errorCode !== 0)
{
    WScript.Echo("errorReason=" + x.parseError.reason);
}
else
    WScript.Echo("===NO PARSE ERROR===
" + x.xml);

3. Run the validate.js file.

You should see the following alert text appear when you run the file.

errorReason=The element 'x:notDeclared' is used but not declared

This indicates that the doc.xml file is not valid according to doc.xsd, because it contains an undeclared element.

4. Try experimenting with different ways to resolve the parse error, and then run the script again.

For example, you could remove the undeclared element by deleting the following line of text from the doc.xml file.

    <x:notDeclared/>

Alternatively, you could declare the element in the doc.xsd file by adding the following lines directly before the </xs:schema> tag (the last line of the file).

    <xs:element name="notDeclared"/>

---

See Also

XML Schema Examples | XML Schema Element Map | XML Schema Elements
XML Data Types Reference | Primitive XML Data Types | Derived XML Data Types | Data Type Facets | IXMLDOMSchemaCollection/XMLSchemaCache

Other Resources

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Validate an XML Document Against an XML Schema Using C++

To validate an XML document file with an XML Schema definition language (XSD) schema file using C++, you load XML and XSD documents and create a schema cache. The following example illustrates this procedure.

```c++
#include "stdio.h"

#import <5.0.dll>
using namespace MSXML2;

int checkParseError(IXMLDOMParseErrorPtr pError);
void dump_com_error(_com_error &e);

int main(int argc, char* argv[])
{
    CoInitialize(NULL);
    try{

        IXMLDOMParseErrorPtr pError;

        // load the XML file
        // ****** you need to use IXMLDOMDocument2 interface *********
        IXMLDOMDocument2Ptr pXMLDoc;
        HRESULT hr = pXMLDoc.CreateInstance(__uuidof(DOMDocument50));
        pXMLDoc->async = VARIANT_FALSE;

        hr = pXMLDoc->load("books.xml");

        // check on the parser error
        if(hr!=VARIANT_TRUE)
        {
            return checkParseError(pXMLDoc->parseError);
        }

        // load the XSD file
        IXMLDOMDocumentPtr pXSDDoc;
        hr = pXSDDoc.CreateInstance(__uuidof(DOMDocument50));
        pXSDDoc->async = VARIANT_FALSE;
```
hr = pXSDDoc->load("books.xsd");

// check on the parser error
if(hr!=VARIANT_TRUE)
{
    return checkParseError(pXSDDoc->parseError);
}

// create schemacache
IXMLDOMSchemaCollectionPtr pSchemaCache;
hr = pSchemaCache.CreateInstance((__uuidof(XMLSchemaCache50)));
pXMLDoc->schemas = pSchemaCache.GetInterfacePtr();

// hook it up with XML Document
hr = pSchemaCache->add("urn:books", pXSDDoc.GetInterfacePtr());

// call validate
pError = pXMLDoc->validate();
if(pError->errorCode != S_OK)
{
    _bstr_t parseError = _bstr_t("Error code: ")+_bstr_t(pError->Getreason());
    MessageBox(NULL,(char*)parseError,"Parse Error",MB_OK);
    return -1;
}
else
    MessageBox(NULL,"Validation succeeded", "Results",MB_OK);
}
catch(_com_error &e)
{
    dump_com_error(e);
    return 0;
}

int checkParseError(IXMLDOMParseErrorPtr pError)
{
    _bstr_t parseError = _bstr_t("At line ")+_bstr_t(pError->Getline());
    MessageBox(NULL,pError->Getreason(),"Parse Error",MB_OK);
    return -1;
}

void dump_com_error(_com_error &e)
{
    printf("Error\n");
    printf("\atCode = %08lx\n", e.Error());
    printf("\atCode meaning = %s", e.ErrorMessage());
Input file: books.xml

<?xml version="1.0"?>
<x:catalog xmlns:x="urn:books">
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
</x:catalog>

Input file: books.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespaces="urn:books">
  <xsd:element name="catalog" type="b:CatalogData"/>
  <xsd:complexType name="CatalogData">
    <xsd:sequence>
      <xsd:element name="book" type="b:bookdata" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

See Also

Validate an XML Document or Fragment | XML Schema Examples | XML
### Other Resources


An XML Schema is composed of the top-level `schema` element. The `schema` element definition must include the following namespace:

```
http://www.w3.org/2001/XMLSchema
```

The `schema` element contains type definitions (`simpleType` and `complexType` elements) and `attribute` and `element` declarations.

In addition to its built-in data types (such as integer, string, and so on), XML Schema also allows for the definition of new data types using the `simpleType` and `complexType` elements.

**simpleType**

A type definition for a value that can be used as the content (`textOnly`) of an element or attribute. This data type cannot contain elements or have attributes.

**complexType**

A type definition for elements that can contain attributes and elements. This data type can contain elements and have attributes.

**See Also**

[XML Schema Examples] | [XML Schema Element Map] | [XML Schema Elements] | [XML Data Types Reference] | [Primitive XML Data Types] | [Derived XML Data Types] | [Data Type Facets]
Other Resources

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
This topic contains the World Wide Web Consortium (W3C) purchase order examples. The first example is the schema for the purchase order. The second example is the instance document that is validated by this schema example.

**Example: Purchase Order Schema**

The following example shows a schema, po.xsd, that defines a purchase order. This example shows the use of element, and attribute declarations. This example also shows simpleType and complexType definitions.

```xml
xmlns="http://tempuri.org/po.xsd" elementFormDefault="qualified">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Purchase order schema for Example.com.
      Copyright 2000 Example.com. All rights reserved.
    </xs:documentation>
  </xs:annotation>

  <xs:element name="purchaseOrder" type="PurchaseOrderType"/>
  <xs:element name="comment" type="xs:string"/>

  <xs:complexType name="PurchaseOrderType">
    <xs:sequence>
      <xs:element name="shipTo" type="USAddress"/>
      <xs:element name="billTo" type="USAddress"/>
      <xs:element ref="comment" minOccurs="0"/>
      <xs:element name="items" type="Items"/>
    </xs:sequence>
    <xs:attribute name="orderDate" type="xs:date"/>
  </xs:complexType>

  <xs:complexType name="USAddress">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="state" type="xs:string"/>
      <xs:element name="zip" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
Example: Purchase Order Instance Document

The following example shows an instance document, po.xml, for the purchase order schema that is validated by po.xsd in the preceding example.

```xml
<?xml version="1.0"?>
<purchaseOrder xmlns="http://tempuri.org/po.xsd" orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
  </shipTo>
</purchaseOrder>
```
<zip>90952</zip>
</shipTo>
<billTo country="US">
  <name>Robert Smith</name>
  <street>8 Oak Avenue</street>
  <city>Old Town</city>
  <state>PA</state>
  <zip>95819</zip>
</billTo>
<comment>Hurry, my lawn is going wild!</comment>
<items>
  <item partNum="872-AA">
    <productName>Lawnmower</productName>
    <quantity>1</quantity>
    <USPrice>148.95</USPrice>
    <comment>Confirm this is electric</comment>
  </item>
  <item partNum="926-AA">
    <productName>Baby Monitor</productName>
    <quantity>1</quantity>
    <USPrice>39.98</USPrice>
    <shipDate>1999-05-21</shipDate>
  </item>
</items>
</purchaseOrder>

See Also

XML Schema Element Map | XML Schema Reference (XSD) | XML Schema Elements | XML Data Types Reference | Primitive XML Data Types | Derived XML Data Types | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XML Schema Element Map

The following is the purchase order schema example shown in the preceding topic. Click the colored text for more information about a schema element.
See Also

XML Schema Examples
XML Schema Elements

XML Schema elements are grouped by their function: top level elements, particles, multiple XML documents and namespaces, identity constraints, attributes, named attributes, complex type definitions, and simple type definitions.

Top Level Elements

Particles

Multiple XML Documents and Namespaces

Identity Constraints

Attributes

Named Schema Objects

Complex Type Definitions

Simple Type Definitions
# Top Level Elements

The following are elements that appear at the top level of a schema document.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation</td>
<td>Defines an annotation.</td>
</tr>
<tr>
<td>attribute</td>
<td>Declares an attribute.</td>
</tr>
<tr>
<td>attributeGroup</td>
<td>Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.</td>
</tr>
<tr>
<td>complexType</td>
<td>Defines a complex type, which determines the set of attributes and the content of an element.</td>
</tr>
<tr>
<td>element</td>
<td>Declares an element.</td>
</tr>
<tr>
<td>group</td>
<td>Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.</td>
</tr>
<tr>
<td>import</td>
<td>Identifies a namespace whose schema components are referenced by the containing schema.</td>
</tr>
<tr>
<td>include</td>
<td>Includes the specified schema document in the target namespace of the containing schema.</td>
</tr>
<tr>
<td>notation</td>
<td>Contains the definition of a notation to describe the format of non-XML data within an XML document. An XML Schema notation declaration is a reconstruction of XML 1.0 NOTATION declarations.</td>
</tr>
<tr>
<td>redefine</td>
<td>Allows simple and complex types, groups, and attribute groups that are obtained from external schema files to be redefined in the current schema. Defines a simple type, which determines the constraints on and</td>
</tr>
</tbody>
</table>
simpleType information about the values of attributes or elements with text-only content.
### Particles

The following are elements that can have `minOccurs` and `maxOccurs` attributes. Such elements always appear as part of a complex type definition or as part of a named model group.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Allows the elements in the group to appear (or not appear) in any order in the containing element.</td>
</tr>
<tr>
<td>any</td>
<td>Enables any element from the specified namespace(s) to appear in the containing <code>sequence</code> or <code>choice</code> element.</td>
</tr>
<tr>
<td>choice</td>
<td>Allows one and only one of the elements contained in the selected group to be present within the containing element.</td>
</tr>
<tr>
<td>element</td>
<td>Declares an element.</td>
</tr>
<tr>
<td>group</td>
<td>Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.</td>
</tr>
<tr>
<td>sequence</td>
<td>Requires the elements in the group to appear in the specified sequence within the containing element.</td>
</tr>
</tbody>
</table>
**Multiple XML Documents and Namespaces**

The following are elements that bring in schema elements from other namespaces or redefine schema elements in the same namespace.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>import</td>
<td>Identifies a namespace whose schema components are referenced by the containing schema.</td>
</tr>
<tr>
<td>include</td>
<td>Includes the specified schema document in the target namespace of the containing schema.</td>
</tr>
<tr>
<td>redefine</td>
<td>Allows simple and complex types, groups, and attribute groups that are obtained from external schema files to be redefined in the current schema.</td>
</tr>
</tbody>
</table>
# Identity Constraints

The following are elements that are related to identity constraints.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>field</strong></td>
<td>Specifies an XML Path Language (XPath) expression that specifies the value (or one of the values) used to define an identity constraint (<strong>unique</strong>, <strong>key</strong>, and <strong>keyref</strong> elements).</td>
</tr>
<tr>
<td><strong>key</strong></td>
<td>Specifies that an attribute or element value (or set of values) must be a key within the specified scope. The scope of a key is the containing <strong>element</strong> in an instance document. A key must be unique, non-nil, and always present.</td>
</tr>
<tr>
<td><strong>keyref</strong></td>
<td>Specifies that an attribute or element value (or set of values) correspond to those of the specified <strong>key</strong> or <strong>unique</strong> element.</td>
</tr>
<tr>
<td><strong>selector</strong></td>
<td>Specifies an XPath expression that selects a set of elements for an identity constraint (<strong>unique</strong>, <strong>key</strong>, and <strong>keyref</strong> elements).</td>
</tr>
<tr>
<td><strong>unique</strong></td>
<td>Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope. The value must be unique or nil.</td>
</tr>
</tbody>
</table>
### Attributes

The following are elements that define attributes in schemas.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>anyAttribute</code></td>
<td>Enables any attribute from the specified namespace(s) to appear in the containing <code>complexType</code> element or in the containing <code>attributeGroup</code> element.</td>
</tr>
<tr>
<td><code>attribute</code></td>
<td>Declares an attribute.</td>
</tr>
<tr>
<td><code>attributeGroup</code></td>
<td>Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.</td>
</tr>
</tbody>
</table>
# Named Schema Objects

The following are elements that define named constructs in schemas. Named constructs are referred to with a QName by other schema elements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>Declares an attribute. Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.</td>
</tr>
<tr>
<td>attributeGroup</td>
<td>Groups a set of attribute declarations so that they can be incorporated as a group into complex type definitions.</td>
</tr>
<tr>
<td>complexType</td>
<td>Defines a complex type, which determines the set of attributes and the content of an element.</td>
</tr>
<tr>
<td>element</td>
<td>Declares an element. Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.</td>
</tr>
<tr>
<td>group</td>
<td>Specifies that an attribute or element value (or set of values) must be a key within the specified scope. The scope of a key is the containing element in an instance document. A key must be unique, non-nillable, and always present.</td>
</tr>
<tr>
<td>key</td>
<td>Specifies that an attribute or element value (or set of values) correspond to those of the specified key or unique element.</td>
</tr>
<tr>
<td>keyref</td>
<td>Contains the definition of a notation to describe the format of non-XML data within an XML document. An XML Schema notation declaration is a reconstruction of XML 1.0 NOTATION declarations.</td>
</tr>
<tr>
<td>notation</td>
<td>Defines a simple type, which determines the constraints on and</td>
</tr>
</tbody>
</table>
simpleType

information about the values of attributes or elements with text-only content.

unique

Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope. The value must be unique or nil.
# Complex Type Definitions

The following are elements that create complex type definitions.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Allows the elements in the group to appear (or not appear) in any order in the containing element.</td>
</tr>
<tr>
<td>annotation</td>
<td>Defines an annotation.</td>
</tr>
<tr>
<td>any</td>
<td>Enables any element from the specified namespace(s) to appear in the containing sequence or choice element.</td>
</tr>
<tr>
<td>anyAttribute</td>
<td>Enables any attribute from the specified namespace(s) to appear in the containing complexType element or in the containing attributeGroup element.</td>
</tr>
<tr>
<td>appinfo</td>
<td>Specifies information to be used by applications within an annotation element.</td>
</tr>
<tr>
<td>attribute</td>
<td>Declares an attribute.</td>
</tr>
<tr>
<td>attributeGroup</td>
<td>Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.</td>
</tr>
<tr>
<td>choice</td>
<td>Allows one and only one of the elements contained in the selected group to be present within the containing element.</td>
</tr>
<tr>
<td>complexContent</td>
<td>Contains extensions or restrictions on a complex type that contains mixed content or elements only.</td>
</tr>
<tr>
<td>documentation</td>
<td>Specifies information to be read or used by users within an annotation element.</td>
</tr>
<tr>
<td>element</td>
<td>Declares an element.</td>
</tr>
</tbody>
</table>
**extension (simpleContent)**

Contains extensions on `simpleContent`. This extends a simple type or a complex type that has simple content by adding specified attribute(s), attribute groups(s) or `anyAttribute`.

**extension (complexContent)**

Contains extensions on `complexType`.

**group**

Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.

**restriction (simpleContent)**

Defines constraints on a `simpleContent` definition.

**restriction (complexContent)**

Defines constraints on a `complexType` definition.

**sequence**

Requires the elements in the group to appear in the specified sequence within the containing element.

**simpleContent**

Contains extensions or restrictions on a `complexType` element with character data or a `simpleType` element as content and contains no elements.
## Simple Type Definitions

The following are elements that create simple type definitions.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation</td>
<td>Defines an annotation.</td>
</tr>
<tr>
<td></td>
<td>Specifies information to be used by applications within an <strong>annotation</strong></td>
</tr>
<tr>
<td></td>
<td>element.</td>
</tr>
<tr>
<td>appinfo</td>
<td>Specifies information to be read or used by users within an <strong>annotation</strong></td>
</tr>
<tr>
<td></td>
<td>element.</td>
</tr>
<tr>
<td>documentation</td>
<td>Specifies information to be read or used by users within an <strong>annotation</strong></td>
</tr>
<tr>
<td>element</td>
<td>Declares an element.</td>
</tr>
<tr>
<td>list</td>
<td>Defines a collection of a single <strong>simpleType</strong> definition.</td>
</tr>
<tr>
<td>restriction (simpleType)</td>
<td>Defines constraints on a <strong>simpleType</strong> definition</td>
</tr>
<tr>
<td>union</td>
<td>Defines a collection of multiple <strong>simpleType</strong> definitions.</td>
</tr>
</tbody>
</table>

### See Also

- [XML Schema Element Map](#) | [XML Schema Reference (XSD)](#) | [XML Data Types Reference](#) | [Primitive XML Data Types](#) | [Derived XML Data Types](#) | [Data Type Facets](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
all

Allows the elements in the group to appear (or not appear) in any order in the containing element.

```xml
<all
    id = ID
    maxOccurs= 1: 1
    minOccurs=(0 | 1): 1
    {any attributes with non-schema Namespace...}>
Content: (annotation?, element*)
</all>
```

Attributes

id

The ID of this element. The **id** value must be of type ID and be unique within the document containing this element.

Optional.

maxOccurs

The maximum number of times the element can occur. The value must be 1.

Optional.

minOccurs

The minimum number of times the element can occur. The value can be an integer equal to 1. To specify that this element is optional, set this attribute to zero. Default value is 1.

Optional.

Element Information

Number of occurrences

One time

**group, restriction (simpleContent), extension**
Example

The following example defines a complex data type containing an all element with three elements.

```xml
<xs:element name="thing1" type="xs:string"/>
<xs:element name="thing2" type="xs:string"/>
<xs:element name="thing3" type="xs:string"/>
<xs:attribute name="myAttribute" type="xs:decimal"/>
<xs:complexType name="myComplexType">
  <xs:all>
    <xs:element ref="thing1"/>
    <xs:element ref="thing2"/>
    <xs:element ref="thing3"/>
  </xs:all>
  <xs:attribute ref="myAttribute"/>
</xs:complexType>
```

See Also

[XML Schema Reference (XSD)](http) | [XML Schema Elements](http)

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
annotation

 Defines an annotation.

 <annotation
    id = ID
    {any attributes with non-schema Namespace}…>
Content: (appinfo | documentation)*
</annotation>

 Attributes

 id

  The ID of this element. The id value must be of type ID and be unique within the document containing this element.

   Optional.

 Element Information

 Number of occurrences
    One time in parent element.

 Parent elements
    Any element

 Contents
    appinfo, documentation

 Remarks

 An annotation can contain appinfo elements (information to be used by applications) and documentation elements (comments or text to be read or used by users).

 Example

 The following example uses annotation and documentation elements to document a simple type and the enumerations that make up its definition.
<xs:simpleType name="northwestStates">
  <xs:annotation>
    <xs:documentation>States in the Pacific Northwest of US</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value='WA'>
      <xs:annotation>
        <xs:documentation>Washington</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='OR'>
      <xs:annotation>
        <xs:documentation>Oregon</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='ID'>
      <xs:annotation>
        <xs:documentation>Idaho</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>

See Also

XML Schema Reference (XSD) | XML Schema Elements

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**any**

Enables any element from the specified namespace(s) to appear in the containing **sequence** or **choice** element.

```xml
<any
    id = ID
    maxOccurs = (nonNegativeInteger | unbounded) : 1
    minOccurs = nonNegativeInteger : 1
    namespace = "(##any | ##other) | List of (anyURI | (#targetNamespace | ##local)"
    processContents = (lax | skip | strict) : strict
    {any attributes with non-schema Namespace...}>
Content: (annotation?)
</any>
```

**Attributes**

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**maxOccurs**

The maximum number of times the any element can occur on the element. The value can be an integer greater than or equal to zero. To set no limit on the maximum number, use the string "unbounded". Default value is 1.

Optional.

**minOccurs**

The minimum number of times the any element can occur on the element. The value can be an integer greater than or equal to zero. To specify that this any group is optional, set this attribute to zero. Default value is 1.

Optional.

**namespace**

The namespaces containing the elements that can be used. If no namespace
is specified, **any** is the default. If the namespace is specified, it must one of the following.

### any

Elements from any namespace can be present.

Elements from any namespace that is not the target namespace of the parent element containing this element can be present.

### local

Elements that are not qualified with a namespace can be present.

### targetNamespace

Elements from the target namespace of the parent element containing this element can be present.

### List of {URI references, targetNamespace, local}

Elements from a space-delimited list of the namespaces can be present. The list can contain the following: URI references of namespaces, **targetNamespace**, and **local**.

Optional.

**processContents**

An indicator of how an application or XML processor should handle validation of XML documents against the elements specified by this **any** element. If no **processContents** attribute is specified, the default is **strict**. If **processContents** is specified, it must one of the following.

### strict

The XML processor must obtain the schema for the required namespaces and validate any element from those namespaces.

The XML processor attempts to obtain the schema for the required namespaces and validate any element from those namespaces; however, if the schema cannot be obtained, no errors will occur.

### skip

The XML processor does not attempt to validate any elements from the specified namespaces.

Optional.
Element Information

**Number of occurrences** Unlimited

**Parent elements** choice, sequence

**Contents** annotation

Example

The following example shows an element declaration for an element that must contain one or more elements from the XHTML namespace with content processing set to lax.

```xml
<xsd:element name='htmlText'>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any namespace='http://www.w3.org/1999/xhtml'
        minOccurs='1' maxOccurs='unbounded'
        processContents='lax'/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

See Also

[XML Schema Reference (XSD)](http) | [XML Schema Elements](http)

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
anyAttribute

Enables any attribute from the specified namespace(s) to appear in the containing complexType element or in the containing attributeGroup element.

<anyAttribute
   id = ID
   namespace = (##any | ##other) | List of (anyURI | (##targetNamespace | ##local | strict): strict
   processContents = (lax | skip | strict): strict
>{any attributes with non-schema Namespace...}>
Content: (annotation?)
</anyAttribute>

Attributes

id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

namespace
The namespace containing the attributes that can be used. If no namespace is specified, ##any is the default. If the namespace is specified, it must one of the following.

##any
Attributes from any namespace can be present.

##other
Attributes from any namespace that is not the target namespace of the parent element containing this anyAttribute element can be present.

##local
Attributes that are not qualified with a namespace can be present.

##targetNamespace
Attributes from the target namespace of the parent element containing this anyAttribute element can be present.

List of {URI references,
## targetNamespace, ##local

Namespaces can be present. The list can contain the following: URI references of namespaces, ##targetNamespace, and ##local.

Optional.

### processContents

An indicator of how an application or XML processor should handle validation of XML documents against the attributes specified by this anyAttribute element. If no processContents attribute is specified, the default is strict. If processContents is specified, it must one of the following.

- **strict**
  
  The XML processor must obtain the schema for the required namespaces and validate any attribute from those namespaces.

- **lax**
  
  The XML processor attempts to obtain the schema for the required namespaces and validate any attribute from those namespaces; however, if the schema cannot be obtained, no errors will occur.

- **skip**
  
  The XML processor does not attempt to validate any attributes from the specified namespaces.

Optional.

### Element Information

- **Number of occurrences**: Unlimited

- **Parent elements**: complexType, restriction (simpleContent), extension (simpleContent), restriction (complexContent), extension (complexContent), attributeGroup

- **Contents**: annotation

### Example

The following example shows an element declaration for an element that
contains a string and can have any attribute from the target namespace.

```xml
<xs:element name="stringElementWithAnyAttribute">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:anyAttribute namespace="##targetNamespace"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

**See Also**


Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
appinfo

Specifies information to be used by applications within an annotation element.

<appinfo
   source = anyURI>
Content: ({any})*
</appinfo>

Attributes

source
The source of the application information. The source must be a URI reference. Optional.

Element Information

Number of occurrences  Unlimited.
Parent elements       annotation
Contents               Any well-formed XML content.

Remarks

Information supplied within the appinfo element is used by applications for processing instructions.

Example

The following example uses annotation, appinfo, and documentation elements.

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   <xs:element name="State">
      <xs:annotation>
         <xs:documentation>State Name</xs:documentation>
      </xs:annotation>
   </xs:element>
See Also

XML Schema Reference (XSD) | XML Schema Elements

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
attribute

Declares an attribute.

<attribute
default = string
fixed = string
form = (qualified | unqualified)
id = ID
name = NCName
ref = QName
type = QName
use = (optional | prohibited | required): optional
{any attributes with non-schema Namespace...}>
Content: (annotation?, (simpleType?))
</attribute>

Attributes

default
Attribute has a default value. If the attribute is not specified in an instance within an XML document, the attribute has the value given. If the attribute is not present in the instance document, then the attribute should appear in the post-schema-validation infoset. Processors of the schema should act as if the attribute was specified with the default value if it was not actually in the instance document. Default and fixed attributes cannot both be present.

Optional.

fixed
Attribute has a fixed value. If the attribute is present in an instance document, its value must be equal to the fixed value given. If it is not present, then the attribute receives the supplied value. Default and fixed attributes cannot both be present.

Optional.

form
The form for the attribute. The default value is the value of the
**attributeFormDefault** attribute of the schema element containing the attribute. The value must be one of the following strings: "qualified" or "unqualified".

If the value is unqualified, this attribute is not required to be qualified with the namespace prefix and is matched against the no-colon-name (NCName) of the attribute (local name).

If the value is qualified, this attribute must be qualified by combining the **targetNamespace** of the schema and the NCName of the attribute.

Optional.

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**name**

The name of the attribute. The name must be an NCName as defined in the XML Namespaces specification. Required if the containing element is the schema element. Name and ref attributes cannot both be present.

When an XML document is validated against a schema, each attribute in the document is validated against an attribute element in the schema.

Optional.

**ref**

The name of an attribute declared in this schema (or another schema indicated by the specified namespace). The ref value must be a qualified name (QName). The type can include a namespace prefix. Name and ref attributes cannot both be present. If ref is present, simpleType element, form, and type cannot be present.

To declare an attribute using an existing attribute definition within a complex type, use the ref attribute to specify the existing attribute definition.
<xs:attribute name="mybaseattribute">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:maxInclusive value="1000"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<xs:complexType name="myComplexType">
  <xs:attribute ref="mybaseattribute"/>
</xs:complexType>

Optional.

**type**

The name of a built-in data type or a simple type defined in this schema (or another schema indicated by the specified namespace). The type must be a QName. The type can include a namespace prefix. The `type` attribute can only be present when the content does not contain a `simpleType` element.

Optional.

**use**

An indicator of how the attribute is used.

If specified, this attribute must have one of the following values.

- **optional**
  
  Attribute is optional and may have any value. This is the default following are equivalent.

  
  ```xml
    <xs:attribute name="myattr" type="xs:string"/>
    <xs:attribute name="myattr" type="xs:string" use="optional"/>
  ```

  Attribute cannot be used. This attribute is used in a restriction of complex type to prohibit the use of existing attributes.

  ```xml
    <xs:attributeGroup name="A">
      <xs:attribute name="x" type="xs:NCName"/>
      <xs:attribute name="y" type="xs:QName"/>
    </xs:attributeGroup>
  ```

- **prohibited**

  ```xml
    <xs:complexType name="B">
      <xs:complexContent>
        <xs:restriction base="xs:A">
          <xs:attribute name="x" use="required"/>
          <xs:attribute name="y" use="prohibited"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  ```
Attribute must appear once.

The attribute is required and can contain any value allowed by the definition of the attribute.

```xml
<xs:attribute name="myattr" type="xs:string" use="required"/>
```

This attribute is used in a restriction or extension of another complex type, to require that a specified attribute or attributes are present.

```xml
<xsl:attributeGroup name="A">
    <xs:attribute name="x" type="xs:NCName"/>
    <xs:attribute name="y" type="xs:QName"/>
</xs:attributeGroup>

<xsl:complexType name="B">
    <xs:complexContent>
        <xs:restriction base="xs:A">
            <xs:attribute name="x" use="required"/>
            <xs:attribute name="y" use="prohibited"/>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

If the attribute is declared as global (its parent element is `schema`), the attribute is required on all elements in the schema.

Optional.

**Element Information**

**Number of occurrences**

Defined one time in the `schema` element. Referred to multiple times in complex types or attribute groups.

- `attributeGroup`, `schema`, `complexType`, `restriction (simpleContent)`, `extension (simpleContent)`, `restriction (complexContent)`, `extension (complexContent)`

**Parent elements**
Remarks

An attribute declaration associates a name with a type definition, which can be a built-in data type or a simple type.

Attribute declarations can be present as child elements of the schema, complexType, and attributeGroup elements (having global scope) or within complex type definitions. For complex types, attribute declarations can be present as local declarations or references to attributes with global scope.

In addition, attributes can appear by reference within attributeGroup and complexType elements.

Examples

In the following example, an attribute is declared by reference to a built-in type with a default value of test and used in a complexType element.

```xml
<xs:attribute name="mybaseattribute" type="xs:string" default="test"/>
<xs:complexType name="myComplexType">
  <xs:attribute ref="mybaseattribute"/>
</xs:complexType>
```

In the following example, a required attribute is declared directly within a complexType element.

```xml
<xs:complexType name="myComplexType">
  <xs:attribute name="mybaseattribute" type="xs:string" use="required"/>
</xs:complexType>
```

In the following example, an attribute is declared by deriving from the built-in integer type (by restriction) and restricting the range of values to between "60" and "95", inclusive.

```xml
<xs:attribute name="myHolidayLocationTemperature">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="60"/>
      <xs:maxInclusive value="95"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```
In the following example, an attribute is declared as a list containing decimal values. (This allows an attribute such as `shoeSizes="10.5 9 8 11"` to contain a list of the values 10.5, 9, 8, and 11).

```xml
<xs:simpleType name="Sizes">
  <xs:restriction base="xs:decimal">
    <xs:enumeration value="10.5"/>
    <xs:enumeration value="9"/>
    <xs:enumeration value="8"/>
    <xs:enumeration value="11"/>
  </xs:restriction>
</xs:simpleType>

<xs:attribute name="shoeSizes">
  <xs:simpleType>
    <xs:list itemType="Sizes"/>
  </xs:simpleType>
</xs:attribute>
```

**See Also**


**Other Resources**

[W3C Namespaces in XML Recommendation](http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-attribute)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**attributeGroup**

Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.

```xml
<attributeGroup
  id = ID
  name = NCName
  ref = QName
  {any attributes with non-schema Namespace...}>
Content: (annotation?), ((attribute | attributeGroup)*, anyAttribute
</attributeGroup>
```

**Attributes**

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**name**

The name of the attribute group whose attributes are included in an attributeGroup element or complexType element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification. Name and ref attributes cannot both be present. Name attribute can only be present if the attribute group is a child of the schema element.

Optional.

**ref**

The reference name of the attribute group whose attributes are included in an attributeGroup element or complexType element.

Name and ref attributes cannot both be present. Ref attribute can only be present if the attribute group is not a child of the schema element.
This value must be a qualified name (QName).

Optional.

**Element Information**

**Number of occurrences**  Unlimited

**Parent elements**

attributeGroup, complexType, schema, restriction (simpleContent), extension (simpleContent), restriction (complexContent), extension (complexContent)

**Contents**

annotation, attribute, attributeGroup, anyAttribute

**Remarks**

An attribute group can be defined only as a child of the schema element. In this case, the name attribute must be present and contain the attribute, attributeGroup, or anyAttribute elements that make up the attribute group.

An attribute group can be included in an attributeGroup or complexType, in which case, the ref attribute must be present and the attributeGroup element must be empty. See the following example.

The attributeGroup elements can contain other attributeGroup elements.

**Example**

The following example shows an attribute group defined (myAttributeGroup) and used in a complex type (myElementType).

```xml
<xs:attributeGroup name="myAttributeGroup">
    <xs:attribute name="someattribute1" type="xs:integer"/>
    <xs:attribute name="someattribute2" type="xs:string"/>
</xs:attributeGroup>

<xs:complexType name="myElementType">
    <xs:attributeGroup ref="myAttributeGroup"/>
</xs:complexType>
```
The following example shows two attribute groups defined (myAttributeGroupA and myAttributeGroupB) with one containing the other.

```xml
<xs:attributeGroup name="myAttributeGroupA">
  <xs:attribute name="someattribute10" type="xs:integer"/>
  <xs:attribute name="someattribute11" type="xs:string"/>
</xs:attributeGroup>

<xs:attributeGroup name="myAttributeGroupB">
  <xs:attribute name="someattribute20" type="xs:date"/>
  <xs:attributeGroup ref="myAttributeGroupA"/>
</xs:attributeGroup>
```

**See Also**


**Other Resources**

[W3C Namespaces in XML Recommendation](http://www.w3.org/2001/XMLSchema-instance)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
choice

Allows one and only one of the elements contained in the selected group to be present within the containing element.

```xml
<choice
  id = ID
  maxOccurs= (nonNegativeInteger | unbounded) : 1
  minOccurs= nonNegativeInteger : 1
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (element | group | choice | sequence | any)*)
</choice>
```

Attributes

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**maxOccurs**

The maximum number of times the choice can occur. The value can be an integer greater than or equal to zero. To set no limit on the maximum number, use the string "unbounded". Default value is 1.

Optional.

**minOccurs**

The minimum number of times the choice can occur. The value can be an integer greater than or equal to zero. To specify that this choice group is optional, set this attribute to zero. Default value is 1.

Optional.

Element Information

**Number of**

One within **group** and **complexType** elements;
occurrences otherwise, unlimited.

Parent elements

- group, choice, sequence, complexType, restriction (simpleContent), extension (simpleContent), restriction (complexType), extension (complexType)

Contents

- annotation, any, choice, element, group, sequence

Example

The following complex type defines an element with one attribute and one and only one element from a choice of four specified elements.

```xml
<xs:complexType name="chadState">
  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:element ref="selected"/>
    <xs:element ref="unselected"/>
    <xs:element ref="dimpled"/>
    <xs:element ref="perforated"/>
  </xs:choice>
  <xs:attribute name="candidate" type="candidateType"/>
</xs:complexType>
```

See Also

- [XML Schema Reference (XSD)](#) | [XML Schema Elements](#)

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
complexContent

Contains extensions or restrictions on a complex type that contains mixed content or elements only.

```xml
<complexContent
    id = ID
    mixed = Boolean
    {any attributes with non-schema Namespace}…>
Content: (annotation?, (restriction | extension))
</complexContent>
```

Attributes

**id**

The ID of this element. The **id** value must be of type ID and be unique within the document containing this element.

Optional.

**mixed**

An indicator of whether character data is allowed to appear between the child elements of this complexType element. The default is **false**.

This **mixed** attribute can override the **mixed** attribute value specified on the containing complexType element.

Optional.

Element Information

**Number of occurrences**  One time

**Parent elements**  complexType

Optional. [annotation](#)

**Contents**  Required. One and only one of the following elements: [restriction](#), or [extension](#)
Examples

The following example has a complex type, `USAddress`, that derives from another complex type, `address`, by extending the inherited type with two additional elements (state and zipcode).

```xml
<xs:complexType name="address">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="USAddress">
  <xs:complexContent>
    <xs:extension base="address">
      <xs:sequence>
        <xs:element name="state" type="USState"/>
        <xs:element name="zipcode" type="xs:positiveInteger"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

See Also

[XML Schema Reference (XSD)](http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-complexContent)

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
complexType

Defines a complex type, which determines the set of attributes and the content of an element.

<complexType
    abstract = Boolean : false
    block = (#all | List of (extension | restriction))
    final = (#all | List of (extension | restriction))
    id = ID
    mixed = Boolean : false
    name = NCName
    {any attributes with non-schema Namespace...}>
Content: (annotation?, (simpleContent | complexContent | ((group | a choice | sequence)?, ((attribute | attributeGroup)*, anyAttribute?))
</complexType>

Attributes

abstract
An indicator of whether the complex type can be used in an instance document. If this value is true, an element cannot use this complex type directly but must use a complex type derived from this complex type. The default is false.

Optional.

block
The type of derivation. The block attribute prevents a complex type that has the specified type of derivation from being used in place of this complex type. This value can contain #all or a list that is a subset of extension or restriction. The block attribute is only used when, during validation, an instance document overrides the normal type for an element by using the schema-instance:type attribute. The block attribute is capable of stopping elements from selecting complex types that are defined through extension and/or restriction to replace the original type that is specified for the element.

Prevents complex types derived by extension from
extension being used in place of this complex type.

restriction Prevents complex types derived by restriction from being used in place of this complex type.

#all Prevents all derived complex types from being used in place of this complex type.

Optional.

final
The type of derivation. The final attribute prevents the specified type of derivation of this complexType element. This value can contain #all or a list that is a subset of extension or restriction.

extension Prevents derivation by extension.

restriction Prevents derivation by restriction.

#all Prevents all derivation (both extension and restriction).

Optional.

id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

mixed
An indicator of whether character data is allowed to appear between the child elements of this complex type. The default is false.

If simpleContent element is a child element, the mixed attribute is not allowed.

If complexContent element is a child element, this mixed attribute can be overridden by the mixed attribute on the complexContent element.

Optional.

name
The name of the type. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.

Required if the containing element is the `schema` element; otherwise, prohibited.

If specified, the name must be unique among all `simpleType` and `complexType` elements.

Optional.

Element Information

**Number of occurrences**  Unlimited within schema; one time within element.
**Parent elements**  `element`, `redefine`, `schema`  
**Content**  `annotation`, `simpleContent`, `complexContent`, `complexType`, `group`,  
`all`, `choice`, `sequence`, `attribute`, `attributeGroup`, `anyAttribute`  

Remarks

A complex type is essentially a type definition for elements that may contain attributes and elements. An element can be declared with a `type` attribute that refers to a `complexType` element that defines the structure, content, and attributes of that element. (An element can also take a reference to a `simpleType` in its `type` attribute.)

A complex type can contain one and only one of the following elements, which determines the type of content allowed in the complex type.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simpleContent</code></td>
<td>The complex type has character data or a <code>simpleType</code> as content and contains no elements, but may contain attributes.</td>
</tr>
<tr>
<td><code>complexContent</code></td>
<td>The complex type contains only elements or no element content (empty).</td>
</tr>
<tr>
<td><code>group</code></td>
<td>The complex type contains the elements defined in the referenced group.</td>
</tr>
</tbody>
</table>
sequence  The complex type contains the elements defined in the
specified sequence.

choice The complex type allows one of the elements specified in the
choice element.

all The complex type allows any or all of the elements specified
in the all element to appear once.

If **group**, **sequence**, **choice**, or **all** is specified as the child element, the attributes
for the **complexType** can be declared optionally using the following elements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>The complex type contains the specified attribute.</td>
</tr>
<tr>
<td>attributeGroup</td>
<td>The complex type contains the attributes defined in the referenced <strong>attributeGroup</strong>.</td>
</tr>
<tr>
<td>anyAttribute</td>
<td>The complex type can contain any attribute from the specified</td>
</tr>
<tr>
<td></td>
<td>namespace(s).</td>
</tr>
</tbody>
</table>

Any number of **attribute** or **attributeGroup** elements can be used. One instance
of **anyAttribute** can also be used.

If **group**, **sequence**, **choice**, or **all** is specified, the elements must appear in the
following order.

1. **group** | **sequence** | **choice** | **all**
2. **attribute** | **attributeGroup**
3. **anyAttribute**

**Examples**

The following example shows a complex type that contains a simple type (**decimal**) with an attribute and element declaration that uses a complex type.

```xml
<xs:complexType name='internationalShoeSize'>
  <xs:simpleContent>
    <xs:extension base='xs:decimal'>
      <xs:attribute name='sizing' type='xs:string' />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```
The following example shows an instance of the **myShoeSize** element within an XML document.

```xml
<myShoeSize sizing='UK'>10.5</myShoeSize>
```

The following example shows the **myShoeSize** element with the **complexType** defined anonymously within the element. (Note the absence of the **name** attribute.)

```xml
<xs:element name='myShoeSize'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:decimal'>
        <xs:attribute name='sizing' type='xs:string' />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

**See Also**

[XML Schema Reference (XSD)](http) | [XML Schema Elements](http)


**Other Resources**

[W3C Namespaces in XML Recommendation](http)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
documentation

Specifies information to be read or used by users within an annotation element.

```xml
<documentation>
  source = anyURI
  xml:lang = language
  Content: ({any})*
</documentation>
```

Attributes

**source**
The source of the application information. The source must be a URI reference. Optional.

**xml:lang**
The indicator of the language used in the contents.

Optional.

Element Information

- **Number of occurrences**: Unlimited
- **Parent elements**: annotation
- **Contents**: Any well-formed XML content.

Remarks

Information supplied within the documentation element is not used in validation.

Example

For examples, see annotation.
See Also

XML Schema Reference (XSD) | XML Schema Elements

element

Declares an element.

```xml
<element
   abstract = Boolean : false
   block = (#all | List of (extension | restriction | substitution))
   default = string
   final = (#all | List of (extension | restriction))
   fixed = string
   form = (qualified | unqualified)
   id = ID
   maxOccurs = (nonNegativeInteger | unbounded) : 1
   minOccurs = nonNegativeInteger : 1
   name = NCName
   nillable = Boolean : false
   ref = QName
   substitutionGroup = QName
   type = QName
   {any attributes with non-schema Namespace}…>
Content: (annotation?, ((simpleType | complexType)?, (unique | key | keyref)*)
</element>
```

Attributes

**abstract**

An indicator of whether the element can be used in an instance document. If this value is `true`, the element cannot appear in the instance document. Instead, another element whose `substitutionGroup` attribute contains the qualified name (QName) of this element must appear in this element's place. More than one element can reference this element in its `substitutionGroup` attribute.

The default is `false`.

Optional.

**block**

The type of derivation. The `block` attribute prevents an element that has the specified type of derivation from being used in place of this element. This
value can contain `#all` or a list that is a subset of `extension`, `restriction`, or `substitution`.

**extension**
- Prevents elements derived by extension from being used in place of this element.

**restriction**
- Prevents elements derived by restriction from being used in place of this element.

**substitution**
- Prevents elements derived by substitution from being used in place of this element.

**#all**
- Prevents all derived elements from being used in place of this element.

Optional.

**default**
The default value of the element if its content is a simple type or its content is `textOnly`.

The `fixed` and `default` attributes are mutually exclusive.

If the element contains a simple type, this value must be a valid value of that type.

Optional.

**substitutionGroup**
The name of an element for which this element can be substituted. This element must have the same type or a type derived from the type of the specified element.

This attribute can be used on any element if the referring element is declared at the global level (parent is `schema` element).

This value must be a QName.

Optional.
**final**

The type of derivation. The final attribute sets the default value of the final attribute on the element element. This value can contain #all or a list that is a subset of extension or restriction.

**extension**

Prevents elements derived by extension from being used in place of this element.

**restriction**

Prevents elements derived by restriction from being used in place of this element.

**#all**

Prevents all derived elements from being used in place of this element.

Prohibited if the containing element is not the schema element.

Optional.

**fixed**

The predetermined, unchangeable value of the element if its content is a simple type or its content is textOnly.

The fixed and default attributes are mutually exclusive.

Optional.

**form**

The form for the element. The default value is the value of the elementFormDefault attribute of the schema element containing the attribute. The value must be one of the following strings: "qualified" or "unqualified".

If the value is unqualified, this element is not required to be qualified with the namespace prefix.

If the value is qualified, this element must be qualified with the namespace prefix.

Optional.
id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

maxOccurs
The maximum number of times the element can occur within the containing element. The value can be an integer greater than or equal to zero. To set no limit on the maximum number, use the string "unbounded".

Prohibited if the containing element is the schema element.

Optional.

minOccurs
The minimum number of times the element can occur within the containing element. The value can be an integer greater than or equal to zero. To specify that this element is optional, set this attribute to zero.

Prohibited if the containing element is the schema element.

Optional.

name
The name of the element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification. Name and ref attributes cannot both be present.

Required if the containing element is the schema element.

Optional.

nillable
The indicator of whether an explicit nil value can be assigned to the element. This applies to element content and not the attributes of the element. The default is false.

If nillable is true, this enables an instance of the element to have the nil attribute set to true. The nil attribute is defined as part of the XML Schema
namespace for instances.

For example, the following fragment defines a single element with nillable set to true.

```xml
<xsl:element name="myDate" type="xs:date" nillable="true"/>
```

The following fragment uses that element and has an explicit nil value (nil attribute set to true).

```xml
<myDate xsi:nil="true"></myDate>
```

Optional.

**ref**

The name of an element declared in this schema (or another schema indicated by the specified namespace). The ref value must be a QName. The ref can include a namespace prefix.

Prohibited if the containing element is the schema element.

If the ref attribute is present, complexType, simpleType, key, keyref, and unique elements and nillable, default, fixed, form, block, and type attributes cannot be present.

```xml
<xsl:element ref="comment"/>
```

Optional.

**type**

Either the name of a built-in data type, or the name of a simpleType or complexType element defined in this schema (or another schema indicated by the specified namespace). The supplied value must correspond to the name attribute on the simpleType or complexType element that is referenced.

The type and ref attributes are mutually exclusive.

To declare an element using an existing simple type or complex type definition, use the type attribute to specify the existing type.
Optional.

**Element Information**

**Number of occurrences** Number of elements defined in the schema.

**Parent elements**  
schema, choice, all, sequence

**Contents**  
simpleType, complexType, key, keyref, unique

**Remarks**

An element declaration associates a name with a type definition, which can be a built-in data type, a simple type, or a complex type.

Element declarations can be present as child elements of the *schema* element (having global scope) or within complex type definitions. Within complex types, element declarations can be present as local declarations or references to elements with global scope.

Global element declarations must directly define complex types.

A *simpleType* or *complexType* element can be present as a child only if both the *ref* and *type* attributes are not present.

In addition, elements can appear by reference within choice, all, sequence, and *complexType* elements.

**Examples**

The following example contains two elements that can serve as substitutes for another element.

```xml
<xs:element name="cat" type="xs:string"/>
<xs:element name="dog" type="xs:string"/>
<xs:element name="redDog" type="xs:string" substitutionGroup="dog"/>
<xs:element name="brownDog" type="xs:string" substitutionGroup ="dog"/>
```
<xs:element name="pets">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="cat"/>
      <xs:element ref="dog"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

See Also

XML Schema Reference (XSD) | XML Schema Elements


Other Resources

W3C Namespaces in XML Recommendation | XML Schema Namespace for Instances
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
extension (simpleContent)

Contains extensions on simpleContent. This extends a simple type or a complex type that has simple content by adding specified attribute(s), attribute group(s), or anyAttribute.

```xml
<extension
    base = QName
    id = ID
    {any attributes with non-schema Namespace}…>
Content: (annotation?, ((attribute | attributeGroup)*, anyAttribute?)
</extension>
```

Attributes

**base**
The name of a built-in data type, simpleType element, or complexType which has simple content.

The base value must be a qualified name (QName).

Required.

**id**
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

Element Information

**Number of occurrences**  One time

**Parent elements**  simpleContent,
annotation, attribute, attributeGroup,
anyAttribute

**Contents**

Example
The following example extends a defined **simpleType** by adding an enumerated attribute.

```xml
<xs:simpleType name="mountainBikeSize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="FamilyMountainBikes">
  <xs:simpleContent>
    <xs:extension base="mountainBikeSize">
      <xs:attribute name="familyMember">
        <xs:restriction base="xs:string">
          <xs:enumeration value="child"/>
          <xs:enumeration value="male"/>
          <xs:enumeration value="female"/>
        </xs:restriction>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

**See Also**

[XML Schema Reference (XSD)](http) | [XML Schema Elements](http)

For more information, see the W3C XML Schema Part 1: Structures Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-simpleContent::extension.
extension (complexType)

Contains extensions on complexContent.

<extension
    base = QName
    id = ID
    {any attributes with non-schema Namespace}…>
Content: (annotation?, ((group | all | choice | sequence)?, ((attrib
    attributeGroup)*, anyAttribute?)))
</extension>

Attributes

base
The name complexType element.

The base value must be a qualified name (QName).

Required.

id
The ID of this element. The id value must be of type ID and be unique
within the document containing this element.

Optional.

Element Information

Number of occurrences  One time
Parent elements  complexContent
Contents  annotation, attribute, attributeGroup, anyAttribute, choice, all, sequence, group

Example

The following example extends a defined complexType element by adding an
element and an attribute.

```xml
<xs:complexType name="address">
   <xs:sequence>
      <xs:element name="street" type="xs:string" />
      <xs:element name="city" type="xs:string" />
   </xs:sequence>
</xs:complexType>

<xs:complexType name="USAddress">
   <xs:complexContent>
      <xs:extension base="address">
         <xs:sequence>
            <xs:element name="state" type="xs:string" />
         </xs:sequence>
         <xs:attribute name="country" type="xs:string" fixed="US" />
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

**See Also**

[XML Schema Reference (XSD)](http) | [XML Schema Elements](http)

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
field

Specifies an XML Path Language (XPath) expression that specifies the value (or one of the values) used to define an identity constraint (unique, key, and keyref elements).

<field
   id = ID
   xpath = a subset of XPath expression
   {any attributes with non-schema Namespace}…>
Content: (annotation?)
</field>

Attributes

id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

xpath
An XPath expression that is relative to each element selected by the selector of the identity constraint. This expression must identify a single element or attribute whose content or value is used for the constraint. If the expression identifies an element, that element must be of a simple type.

Required.

Element Information

Number of occurrences One time
Parent elements key, keyref, unique
Contents annotation

Remarks
An identity constraint must contain one or more field elements and those field
elements must appear after the selector element.
Selector ::= Path ( '|' Path )*
FieldXPath ::= ('.//')? ( Step '/' )* ( Step | '@' NameTest )
Step ::= '.' | NameTest
NameTest ::= QName | '*' | NCName ':' '*'

Note When using XPath, you must use fully qualified names if the name is
defined with a non-empty targetNamespace. For example,
myNS:localName instead of localName.
Example
The following fragment is a field element that specifies the myID attribute as
the field to use for the identity constraint.
<xs:field xpath="@myID"/>

See Also
XML Schema Reference (XSD) | XML Schema Elements
For more information, see the W3C XML Schema Part 1: Structures


Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**group**

Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.

```xml
<group
   name= NCName
   id = ID
   maxOccurs = (nonNegativeInteger | unbounded) : 1
   minOccurs = nonNegativeInteger : 1
   name = NCName
   ref = QName
   {any attributes with non-schema Namespace}…>
Content: (annotation?, (all | choice | sequence))
</group>
```

**Attributes**

**name**

The name of the group. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.

This attribute is used only when the schema element is the parent of this group element. In this case, the group is a model group to be used by complexType, choice, and sequence elements.

Optional.

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**maxOccurs**

The maximum number of times the element can occur within the containing element. The value can be an integer greater than or equal to zero. To set no limit on the maximum number, use the string "unbounded".
Limited when the group is not a child of the schema element.

Optional.

**minOccurs**
The minimum number of times the element can occur within the containing element. The value can be an integer greater than or equal to zero. To specify that this element is optional, set this attribute to zero.

Limited when the group is not a child of the schema element.

Optional.

**name**
The name of the element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification. Name and ref attributes cannot both be present.

Required if the containing element is the schema element.

Optional.

**ref**
The name of a group declared in this schema (or another schema indicated by the specified namespace). The ref value must be a QName. The ref can include a namespace prefix.

If the ref attribute is present, id, minOccurs, and maxOccurs may be present. Ref and name are mutually exclusive.

To declare a group using an existing group definition, use the ref attribute to specify the existing group definition.

<xs:group ref="globalGroup"/>

Optional.

**Element Information**

**Number of** Unlimited
occurrences

Parent elements

- schema, choice, sequence, complexType, restriction (complexContent), extension (complexContent)

Contents

- annotation, all, choice, sequence

Remarks

Any one of the following elements can be a child of the group element.

choice

Permits one and only one of the elements contained in the group.

sequence

Requires the elements in the group to appear in the specified sequence.

all

Allows the elements in the group to appear (or not appear) in any order in the containing element.

The minOccurs and maxOccurs attributes have the default value 1. A group with neither attribute must appear once and only once in the containing element.

Example

The following example defines a group containing a sequence of three elements and uses the group element in a complex type definition.

```xml
<xs:element name="thing1" type="xs:string"/>
<xs:element name="thing2" type="xs:string"/>
<xs:element name="thing3" type="xs:string"/>

<xs:attribute name="myAttribute" type="xs:decimal"/>

<xs:group name="myGroupOfThings">
  <xs:sequence>
    <xs:element ref="thing1"/>
    <xs:element ref="thing2"/>
    <xs:element ref="thing3"/>
  </xs:sequence>
</xs:group>

<xs:complexType name="myComplexType">
  <xs:group ref="myGroupOfThings"/>
  <xs:attribute ref="myAttribute"/>
```
</xs:complexType>

See Also

XML Schema Reference (XSD) | XML Schema Elements


Other Resources

W3C Namespaces in XML Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
import

Identifies a namespace whose schema components are referenced by the containing schema.

```
<import
  id = ID
  namespace = anyURI
  schemaLocation = anyURI
  {any attributes with non-schema Namespace}...>
Content: (annotation?)
</import>
```

**Attributes**

**id**

The ID of this element. The `id` value must be of type ID and be unique within the document containing this element.

Optional.

**namespace**

The URI reference to the namespace to import.

The `namespace` attribute indicates that the containing schema document may contain qualified references to schema components in that namespace (through one or more prefixes declared with `xmlns` attributes).

If this attribute is absent, the containing schema can contain unqualified references to components in the imported namespace.

Optional.

**schemaLocation**

The URI reference to the location of a schema document for the imported namespace.

If this attribute is absent, the author is allowing the identification of the
schema for the imported namespace to be determined by the XML document that is an instance of the containing schema or the user or application that is processing it.

Optional.

**Element Information**

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>schema</td>
</tr>
<tr>
<td>Contents</td>
<td>annotation</td>
</tr>
</tbody>
</table>

**Remarks**

The difference between the **include** element and the **import** element is that the **import** element allows references to schema components from schema documents with different target namespaces and the **include** element adds the schema components from other schema documents that have the same target namespace (or no specified target namespace) to the containing schema. In short, the **import** element allows you to use schema components from any schema; the **include** element allows you to add all the components of an included schema to the containing schema.

**Example**

The following example shows importing a namespace.

```xml
<?xml version="1.0"?>
<xs:schema elementFormDefault="qualified" targetNamespace="http://www.example.com/IPO"
    <xs:import namespace="http://www.example.com/IPO" />
    <xs:include schemaLocation="example.xsd" />
</xs:schema>
```

**See Also**

[XML Schema Reference (XSD)](#) | [XML Schema Elements](#) | [include](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
include

Includes the specified schema document in the target namespace of the containing schema.

```
<include
   id = ID
   schemaLocation = anyURI
   {any attributes with non-schema Namespace}...>
Content: (annotation?)
</include>
```

**Attributes**

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**schemaLocation**

The URI reference to the location of a schema document to include in the target namespace of the containing schema.

Required.

**Element Information**

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>schema</td>
</tr>
<tr>
<td>Child elements</td>
<td>annotation</td>
</tr>
</tbody>
</table>

**Remarks**

The included schema document must meet one of the following conditions.
- It must have the same target namespace as the containing schema document.
- It must not have a target namespace specified (no `targetNamespace` attribute).

The **import** element allows references to schema components from schema documents with different target namespaces; the **include** element adds the schema components from other schema documents that have the same target namespace (or no specified target namespace) to the containing schema. In short, the **import** element allows you to use schema components from any schema; the **include** element allows you to add all the components of an included schema to the containing schema.

**Example**

For an example, see [import](#).

**See Also**

[XML Schema Reference (XSD)](#) | [XML Schema Elements] | [import](#)

For more information, see the W3C XML Schema Part 1: Structures Recommendation at [http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-include](http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-include).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
key

Specifies that an attribute or element value (or set of values) must be a key within the specified scope. The scope of a key is the containing element in an instance document. A key means that data should be unique within a specified scope, non-nillable, and always present.

<key
   id = ID
   name = NCName
   {any attributes with non-schema Namespace}…>
Content: (annotation?, (selector, field+))
</key>

Attributes

id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

name
The name of the key element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.

The name must be unique within an identity constraint set.

Required.

Element Information

Number of occurrences Unlimited
Parent elements element
Contents annotation, field, selector

Remarks
The **key** element must contain the following elements in order.

**selector**

The selector element contains an XML Path Language (XPath) expression specifying the set of elements across which the values specified by field must be unique.

There must be one and only one selector element.

Each **field** element contains an XPath expression specifying the values (attribute or element values) that must be unique for the set of elements specified by the selector element.

If there is more than one field element, the combination of the field elements must be unique. In this case, the values for a single field element may or may not be unique across the selected elements but the combination of all the fields must be unique.

There must be one or more field element(s).

**Example**

The following example defines a keyref element that corresponds to the key element in this schema.

```xml
<xs:schema	xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="namespace1"
xmlns:r="namespace1"
elementFormDefault="qualified">

<xs:element name="root">
<xs:complexType>
<xs:sequence>

<xs:element name="A" type="r:A" maxOccurs="unbounded">
<xs:keyref name="dummy" refer="r:pNumKey">
<xs:selector xpath="r:part"/>
<xs:field xpath="@ref-number"/>
</xs:keyref>
</xs:element>

</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```
Input: key.xml

<root xmlns="namespace1">
  <A>
    <!-- if the ref-number is not equal to one of the key-number, th
    <part ref-number="1"/>
See Also

XML Schema Reference (XSD) | XML Schema Elements


Other Resources

W3C Namespaces in XML Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
keyref

Specifies that an attribute or element value (or set of values) correspond to those of the specified key or unique element.

```xml
<keyref
  id = ID
  name = NCName
  refer = QName
  {any attributes with non-schema Namespace}…>
Content: (annotation?, (selector, field+))
</keyref>
```

Attributes

id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

name
The name of the keyref element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.

The name must be unique within an identity constraint set.

Required.

refer
The name of a key or unique element defined in this schema (or another schema indicated by the specified namespace). The refer value must be a qualified name (QName). The type can include a namespace prefix.

Required.

Element Information
Number of occurrences One time
Parent elements element
Contents annotation, field, selector

Remarks

The **keyref** element must contain the following elements in order.

The **selector** element contains an XML Path Language (XPath) expression specifying the set of elements across which the values specified by **field** must be unique.

There must be one and only one **selector** element.

Each **field** element contains an XPath expression specifying the values (attribute or element values) that must be unique for the set of elements specified by the **selector** element.

If there is more than one **field** element, the combination of the **field** elements must be unique. In this case, the values for a single **field** element may or may not be unique across the selected elements but the combination of all the fields must be unique.

There must be one or more **field** elements.

Example

For an example, see **key**.

See Also


Other Resources

W3C Namespaces in XML Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
list

Defines a collection of a single simpleType definition.

```xml
<list
   id = ID
   itemType = QName
   {any attributes with non-schema Namespace}…>
Content: (annotation?, (simpleType?))
</list>
```

Attributes

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**itemType**

The name of a built-in data type or simpleType element defined in this schema (or another schema indicated by the specified namespace). The simpleType element containing the list element is derived from the simple type specified by the list value. The list value must be a qualified name (QName). The use of the simpleType element child and the itemType attribute is mutually exclusive.

Not allowed if the content contains a simpleType element. Otherwise, required.

Element Information

**Number of occurrences** One time

**Parent elements** simpleType

**Child elements** annotation, simpleType

Remarks
When a data type is derived from a list data type, the following constraining facets can be used.

- `length`
- `maxLength`
- `minLength`
- `enumeration`
- `pattern`
- `whiteSpace`

The unit of `length` is measured in the number of list items. For example, the following example sets the `maxLength` to 5, limiting the number of items in an instance of the `derivedlistOfIntegers` data type to 5 items in this list. Note that restrictions in a list are distinct from restrictions used in other elements.

Lists are always delimited by white space. For example, deriving a list from a simple type of three enumerations such as "books" "magazine articles" "technical newspapers and business journals" generates a list of eight strings: "books", "magazine", "articles", "technical", "newspapers", "and", "business", "journals" not a list of three strings.

**Example**

The following example shows a `simpleType` that is a list of integers where 5 is the maximum number of items allowed in the list.

```xml
<xs:simpleType name='derivedlistOfIntegers'>
  <xs:restriction base='listOfIntegers'>
    <xs:maxLength value='5'/>
  </xs:restriction>
</xs:simpleType>
```

**Example**

The following example shows a `simpleType` that is a list of integers.

```xml
<xs:simpleType name='listOfIntegers'>
  <xs:list itemType='integer'/>  
</xs:simpleType>
```
Example

The following example shows an instance of an element that has an attribute of type **listOfIntegers**. Each item in the list is of type **integer** and is separated by white space — in this case, a space.

```
<myelement listOfIntegers='1 100 9 4000 0'>
```

See Also


Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
**notation**

Contains the definition of a notation to describe the format of non-XML data within an XML document. An XML Schema notation declaration is a reconstruction of XML 1.0 NOTATION declarations.

```xml
<notation
  id = ID
  name = NCName
  public = Public identifier per ISO 8879
  system = anyURI
  {any attributes with non-schema Namespace}…>
Content: (annotation?)
</notation>
```

**Attributes**

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**name**

The name of the notation element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.

The name must be unique within a schema.

Required.

**public**

A URI reference corresponding to the public identifier. The identifier in the document prolog that gives the name of a DTD or external entity.

Required.

**system**

A URI reference corresponding to the system identifier. A local, system-
dependent identifier for a document, DTD, or external entity. The name must be a URI.

Optional.

**Element Information**

**Number of occurrences** Unlimited

**Parent elements** [schema]

**Contents** [annotation]

**Example**

The following example shows a jpeg notation using a viewer application, viewer.exe.

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:notation name="jpeg" public="image/jpeg" system="viewer.exe"
</xs:schema>
```

**See Also**


**Other Resources**

[W3C Namespaces in XML Recommendation](http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-notation)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
redefine

Allows simple and complex types, groups, and attribute groups that are obtained from external schema files to be redefined in the current schema.

```xml
<redefine
  id = ID
  schemaLocation = anyURI
  {any attributes with non-schema Namespace}…>
Content: (annotation | (simpleType | complexType | group | attributeGroup))*
</redefine>
```

Attributes

**id**

The ID of this element. The **id** value must be of type ID and be unique within the document containing this element.

Optional.

**schemaLocation**

A URI reference to the location of a schema document.

Required.

Element Information

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td><code>schema</code></td>
</tr>
<tr>
<td>Contents</td>
<td><code>annotation</code>, <code>attributeGroup</code>, <code>complexType</code>, <code>group</code>, <code>simpleType</code></td>
</tr>
</tbody>
</table>

Remarks

The **redefine** element requires that the external elements exist in the same target namespace as the redefining schema. Schemas without a namespace can also be
redefined using the **redefine** element and the redefined components become part of the schema’s target namespace.

The definitions within the **redefine** element itself must use their current definitions as the base type definition. Attribute group definitions and model group definitions must include exactly one reference to themselves. Not all of the definitions of the redefined schema document must be redefined.

Redefining elements can cause unexpected results, such as ill-formed definitions, on other type definitions that are based on the redefined definitions.

**Example**

The following example shows a schema, v2.xsd, with elements specified by the v1.xsd. The **personName** type is redefined. According to this schema, elements constrained by the **personName** type may end with a **generation** element.

```xml
v1.xsd:
<xs:complexType name="personName">
  <xs:sequence>
    <xs:element name="title" minOccurs="0"/>
    <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="addressee" type="personName"/>

v2.xsd:
<xs:redefine schemaLocation="v1.xsd">
  <xs:complexType name="personName">
    <xs:complexContent>
      <xs:extension base="personName">
        <xs:sequence>
          <xs:element name="generation" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
<xs:element name="author" type="personName"/>
```

**See Also**
XML Schema Reference (XSD) | XML Schema Elements

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
restriction (simpleType)

Defines constraints on a `simpleType` definition.

```xml
<restriction
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (simpleType?, (minExclusive | minInclusive | maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength | maxLength | enumeration | whiteSpace | pattern))*)
</restriction>
```

Attributes

**base**

The name of a built-in data type, `simpleType` element defined in this schema (or another schema indicated by the specified namespace). The element containing the `restriction` element is derived from the type specified by the base value.

The base value must be a qualified name (QName).

Required.

**id**

The ID of this element. The `id` value must be of type ID and be unique within the document containing this element.

Optional.

Element Information

**Number of occurrences**

One time

**Parent elements**

`simpleType`, `annotation`, `fractionDigits`, `enumeration`, `length`, `maxExclusive`, `maxInclusive`, `maxLength`, `minExclusive`, `minInclusive`

**Contents**

`maxLength`, `minExclusive`, `minInclusive`
Example

The following example shows a simple type definition using restriction. (The simple type is derived from `nonNegativeInteger` and its `maxInclusive` value is set to 100.)

```xml
<xs:simpleType name="freezeboilrangeInteger">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Example

The following example shows a simple type definition that derives from the `integer` built-in type and has all the facets of `integer` (no restrictions).

```xml
<xs:simpleType name="myInteger">
  <xs:restriction base="xs:integer"/>
</xs:simpleType>
```

Example

The following example shows a simple type definition that derives from `string` and is an enumerated type. The set of values is restricted to specified values.

```xml
<xs:simpleType name="mountainbikesize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>
```

See Also

[XML Schema Reference (XSD)](#) | [XML Schema Elements](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
restriction (simpleContent)

Defines constraints on a simpleContent definition.

```xml
<restriction
    base = QName
    id = ID
    {any attributes with non-schema Namespace}…>
Content: (annotation?, (simpleType?,
    (minExclusive | minInclusive | maxExclusive | maxInclusive | totalDigits | fractionDigits | length |
    minLength | maxLength | enumeration | whiteSpace | pattern)*)?,
    ((attribute | attributeGroup)*, anyAttribute?))
</restriction>
```

Attributes

**base**

The name of a built-in data type, `simpleType` element, or `complexType` element defined in this schema (or another schema indicated by the specified namespace). The element containing the restriction element is derived from the type specified by the base value.

**base** is the name of a built-in data type, simple type, or complex type. If a complex type, this type must be of one of the following: built-in data type, simple type, or simple content.

The base value must be a qualified name (QName).

Required.

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

Element Information
**Number of occurrences**
One time

**Parent elements**
- `simpleContent`
- `annotation`, `fractionDigits`, `enumeration`, `length`, `maxExclusive`, `maxInclusive`, `maxLength`, `minExclusive`, `minInclusive`, `minLength`, `pattern`, `simpleType`, `totalDigits`, `whiteSpace`, `attribute`, `attributeGroup`, `anyAttribute`

**Contents (simpleContent)**

**Example**

The following example shows a simpleContent definition that restricts another simpleContent extension. The set of values is restricted to a short list of values and the attribute.

```xml
<xs:simpleType name="mountainbikesize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="FamilyMountainBikeSizes">
  <xs:simpleContent>
    <xs:extension base="mountainbikesize">
      <xs:attribute name="familyMember" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="ChildMountainBikeSizes">
  <xs:simpleContent>
    <xs:restriction base="FamilyMountainBikeSizes">
      <xs:enumeration value="small"/>
      <xs:enumeration value="medium"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

**See Also**

[XML Schema Reference (XSD)](https) | [XML Schema Elements](https)
restriction (complexType)

Defines constraints on a complexContent definition.

```xml
<restriction
    base = QName
    id = ID
    {any attributes with non-schema Namespace}...>
Content: (annotation?, (group | all | choice | sequence)?, ((attribute | attributeGroup)*, anyAttribute?))
</restriction>
```

### Attributes

**base**

The name of a complexType element defined in this schema (or another schema indicated by the specified namespace). The element containing the restriction element is derived from the type specified by the base value.

The base value must be a qualified name (QName).

Required.

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

### Element Information

**Number of occurrences**

One time

**Parent elements**

complexType

group, all, choice, sequence, attribute, attributeGroup, anyAttribute

**Contents**

Example
The following example shows a complex type definition using **restriction**. The complex type, `USAddress`, is derived from a general address complex type and its country element is fixed to `US`.

```xml
<xs:complexType name="address">
  <xs:sequence>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="zipcode" type="xs:integer"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="USAddress">
  <xs:complexContent>
    <xs:restriction base="address">
      <xs:sequence>
        <xs:element name="street" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="zipcode" type="xs:integer"/>
        <xs:element name="country" type="xs:string" fixed="US"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

**See Also**

[XML Schema Reference (XSD)](https) | [XML Schema Elements](https)

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
schema

Contains the definition of a schema.

```xml
<schema
  attributeFormDefault = (qualified | unqualified): unqualified
  blockDefault = (#all | List of (extension | restriction | substi
tu
elementFormDefault = (qualified | unqualified): unqualified
  finalDefault = (#all | List of (extension | restriction | list |
union): ''
  id = ID
  targetNamespace = anyURI
  version = token
  xml:lang = language
  {any attributes with non-schema Namespace}…>
Content: ((include | import | redefine | annotation)*, (((simpleType
complexType | group | attributeGroup) | element | attribute | notati
annotation*))*)
</schema>
```

Attributes

attributeFormDefault

The form for attributes declared in the target namespace of this schema. The value must be one of the following strings: qualified or unqualified. The default is unqualified.

If the value is unqualified, attributes from the target namespace are not required to be qualified with the namespace prefix.

If the value is qualified, attributes from the target namespace must be qualified with the namespace prefix.

This value is the global default for all attributes declared in the target namespace. Individual attributes can override this setting for their local scope using the form attribute.

Optional.

blockDefault
The type of derivation. The **blockDefault** attribute sets the default value of the **block** attribute on **element** and **complexType** elements in the target namespace of this schema. The **block** attribute prevents a complex type (or element) that has the specified type of derivation from being used in place of the inherited complex type (or element). This value can contain **#all** or a list that is a subset of **extension, restriction, or substitution**.

- **extension**
  Prevents complex types derived by extension from being used in place of this complex type.

- **restriction**
  Prevents complex types derived by restriction from being used in place of this complex type.
  Prevents substitution of elements. (An element can declare itself a substitution for another element by specifying the name of the other element in its **substitutionGroup** attribute.)

- **#all**
  Prevents all derived complex types from being used in place of this complex type.

Optional.

**elementFormDefault**
The form for elements declared in the target namespace of this schema. The value must be one of the following strings: **qualified** or **unqualified**. The default is **unqualified**.

If the value is **unqualified**, elements from the target namespace are not required to be qualified with the namespace prefix.

If the value is **qualified**, elements from the target namespace must be qualified with the namespace prefix.

This value is the global default for all elements declared in the target namespace. Individual elements can override this setting for their local scope using the **form** attribute.

Optional.

**finalDefault**
The type of derivation. The **finalDefault** attribute sets the default value of
the final attribute on element, simpleType, and complexType elements in the target namespace of this schema. The final attribute prevents the specified type of derivation of an element, simpleType, or complexType element. For element and complexType elements, this value can contain #all or a list that is a subset of extension or restriction. For simpleType elements, this value can additionally contain list and union.

extension
By default, elements in this schema cannot be derived by extension. Applies to element and complexType elements only.

restriction
Prevents derivation by restriction.

list
Prevents derivation by list. Applies to simpleType elements only.

union
Prevents derivation by union. Applies to simpleType elements only.

#all
#all. By default, elements in this schema cannot be derived by any method.

Optional.

id
The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

targetNamespace
The URI reference of the namespace of this schema. A prefix for the namespace can also be assigned. If no prefix is assigned, the schema components of the namespace can be used with unqualified references.

Optional.

version
The version of the schema.

Optional.

xml:lang
The indicator of the language used in the contents.

Optional.

**Element Information**

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>One time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>(No parent elements)</td>
</tr>
<tr>
<td></td>
<td>include, import, annotation, redefine</td>
</tr>
<tr>
<td>Content</td>
<td>attribute, attributeGroup, element, group, notation, simpleType, complexType</td>
</tr>
<tr>
<td>Other</td>
<td>The include, import, and redefine elements must come before the other elements — except for annotation element, which can appear anywhere.</td>
</tr>
</tbody>
</table>

**Remarks**

The *schema* element is the document element (the top-level element) in a schema definition.

The namespaces specified using the *xmlns* attribute must include the namespace for the schema-defining elements and attributes (*schema, element, and so on*). Any prefix can be used, but it is convenient to make the *schema* namespace the default namespace to avoid having to use a prefix on every XML Schema element.

The *targetNamespace* is the namespace of all schema components in this schema as well as any schemas included using the *include* element. (Included schemas must have the same target namespace as the containing schema or have no target namespace at all.) In the following example, the schema components (element name, type) in the [http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema) namespace are unqualified and those for [http://tempuri.org/myschema](http://tempuri.org/myschema) (mydecimal) are qualified with the *msc* prefix.

```xml
<xs:schema
```
For XML Schema, the namespace is the following:

http://www.w3.org/2001/XMLSchema

In addition, the schema element should also contain namespace declarations for any other schemas that it uses.

**Example**

The following schema uses the XML Schema namespace as the default namespace and defines the target namespace for the schema as http://mynamespace/myschema.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://mynamespace/myschema">
   <!--Put schema content here -->
</xs:schema>
```

**See Also**

XML Schema Reference (XSD) | XML Schema Elements | include

selector

Specifies an XML Path Language (XPath) expression that selects a set of elements for an identity constraint (unique, key, and keyref elements).

```xml
<selector
  id = ID
  xpath = a subset of XPath expression
  {any attributes with non-schema Namespace}…>
Content: (annotation?)
</selector>
```

Attributes

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**xpath**

An XPath expression that is relative to the element being declared. This expression identifies the child elements (of the element being declared) to which the identity constraint applies.

Required.

Element Information

**Number of occurrences** One time

**Parent elements** key, keyref, unique

**Contents** annotation

Remarks

Selector ::= Path ( '|' Path )*  
Path ::= ('.'//')? Step ( '/' Step )*
Step ::= '.' | NameTest
NameTest ::= QName | '*' | NCName '::' '*'

**Note** When using XPath, you must use fully qualified names. For example, `myNS:localName` instead of `localName`.

**Example**

The following fragment is a selector element that selects all expense elements with a field element that selects an approved attribute.

```xml
<selector xpath=".//expense"/>
<field xpath="@approved"/>
```

**See Also**


Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
sequence

Requires the elements in the group to appear in the specified sequence within the containing element.

```xml
<sequence
    id = ID
    maxOccurs = (nonNegativeInteger | unbounded) : 1
    minOccurs = nonNegativeInteger : 1
    {any attributes with non-schema Namespace}…>
Content: (annotation?, (element | group | choice | sequence | any)*)
</sequence>
```

**Attributes**

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**maxOccurs**

The maximum number of times the sequence can occur. The value can be an integer greater than or equal to zero. To set no limit on the maximum number, use the string "unbounded".

Optional.

**minOccurs**

The minimum number of times the sequence can occur. The value can be an integer greater than or equal to zero. To specify that this sequence group is optional, set this attribute to zero.

Optional.

**Element Information**

**Number of**

One time within group; otherwise, unlimited.
occurrences

Parent elements
group, choice, sequence, complexType, restriction (simpleContent), extension (simpleContent), restriction (complexType), extension (complexType)

Contents
annotation, any, choice, element, group, sequence

Example

The following example shows an element (zooAnimals) that can have zero or more of the following elements, elephant, bear, giraffe, in the sequence element.

```xml
<xs:element name="zooAnimals">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="elephant"/>
      <xs:element name="bear"/>
      <xs:element name="giraffe"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

See Also

XML Schema Reference (XSD) | XML Schema Elements

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
simpleContent

Contains extensions or restrictions on a complexType element with character data or a simpleType element as content and contains no elements.

<simpleContent id = ID
   {any attributes with non-schema Namespace}...>
Content: (annotation?, (restriction | extension))
</simpleContent>

Attributes

id

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

Element Information

Number of occurrences
One time

Parent elements
complexType
Optional — annotation

Contents
Required — One and only one of the following elements: restriction(simpleContent), or extension(simpleContent).

Remarks

The simpleContent element enables you to specify an element as containing a simpleType with no elements but enables you to restrict the value of the element's content or extend the element with attributes. For example, the following element declaration does not allow for adding attributes (such as
sizing).

```xml
<xs:element name="shoeSize" type="xs:decimal"/>
```

**Example**

Using the `simpleContent` element, the following element declaration contains a `complexType` element that defines the content of the element as a `decimal` type with a single attribute (`sizing`).

```xml
<xs:element name="shoeSize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="sizing">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="US"/>
              <xs:enumeration value="European"/>
              <xs:enumeration value="UK"/>
              <xs:enumeration value="Japan"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Simple content must be defined in one of the following ways.

- **restriction** element
  - Restricts the range of values for the element to a subset of those for inherited `simpleType`.

- **extension** element
  - Extends the `simpleType` content of the element by adding attributes.

**Examples**

The following example shows an element (`generalPrice`) declared using an anonymous `complexType` (one without a name) with its content defined as a `decimal` data type and with a `currency` attribute.

```xml
<xs:element name="generalPrice"/>
<xs:complexType>
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="currency" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

See Also

XML Schema Reference (XSD) | XML Schema Elements

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**simpleType**

Defines a simple type, which determines the constraints on and information about the values of attributes or elements with text-only content.

```xml
<simpleType
   final = (#all | (list | union | restriction))
   id = ID
   name = NCName
   {any attributes with non-schema Namespace}...>
Content: (annotation?, (restriction | list | union))
</simpleType>
```

**Attributes**

**final**

The type of derivation. The `final` attribute prevents the specified type of derivation of this `simpleType` element. This value can contain `#all` or a list that is a subset of `list`, `union`, or `restriction`.

- **list**
  - Prevents derivation by list.
- **union**
  - Prevents derivation by union.
- **restriction**
  - Prevents derivation by restriction.
- **#all**
  - Prevents all derivation (list, union, restriction).

Optional.

**id**

The ID of this element. The `id` value must be of type ID and be unique within the document containing this element.

Optional.

**name**

The name of the type. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.
If specified, the name must be unique among all `simpleType` and `complexType` elements.

Required if the `simpleType` element is a child of the `schema` element, and not allowed at other times.

**Element Information**

**Number of occurrences** Unlimited
**Parent elements** `attribute`, `element`, `list`, `restriction (simpleType)`, `schema`, `union`
**Contents** `annotation`, `list`, `restriction (simpleType)`, `union`

**Remarks**

Simple types are defined by deriving them from existing simple types (built-in data types and derived simple types). A simple type cannot contain elements and cannot have attributes.

Simple types can be defined in one of the following ways.

- **restriction** Restrictions the range of values for the simple type to a subset of those for inherited simple type.
- **list** Defines a simple type that contains a white space-separated list of values of an inherited simple type.
- **union** Defines a simple type that contains a union of the values of two or more inherited simple types.

A `simpleType` declaration contained within a `complexType` or `attribute` element defines that simple type within the scope of the `complexType` or `attribute` that contains it. If a `simpleType` declaration has the `schema` element as its parent, it has global scope within that schema.

After a simple type is defined, it can be used in an attribute or element declaration or `complexType` (which, in turn, can be used in an element declaration) definition.
Examples

The following examples show simple type definitions using **restriction**, **list**, and **union** elements.

Restriction

```xml
<xs:simpleType name="freezeboilrangeInteger">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

List

```xml
<xs:simpleType name="listOfDates">
  <xs:list itemType="xs:date">
  </xs:list>
</xs:simpleType>
```

Union

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="allframesize">
    <xs:simpleType>
      <xs:union>
        <xs:simpleType>
          <xs:restriction base="roadbikesize"/>
        </xs:simpleType>
        <xs:simpleType>
          <xs:restriction base="mountainbikesize"/>
        </xs:simpleType>
      </xs:union>
    </xs:simpleType>
  </xs:attribute>
</xs:schema>
```

```xml
<xs:simpleType name="roadbikesize">
  <xs:restriction base="xs:positiveInteger">
    <xs:enumeration value="46"/>
    <xs:enumeration value="52"/>
    <xs:enumeration value="55"/>
  </xs:restriction>
</xs:simpleType>
```

```xml
<xs:simpleType name="mountainbikesize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
  </xs:restriction>
</xs:simpleType>
```
See Also

**XML Schema Reference (XSD)** | **XML Schema Elements**

For more information, see the W3C XML Schema Part 1: Structures Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#element-simpleType.

Other Resources

**W3C Namespaces in XML Recommendation**
union

Defines a collection of multiple simpleType definitions.

```xml
<union
  id = ID
  memberTypes = List of QNames
  {any attributes with non-schema Namespace}>
Content: (annotation?, (simpleType*))
</union>
```

Attributes

**id**

The ID of this element. The id value must be of type ID and be unique within the document containing this element.

Optional.

**memberTypes**

The list of names of built-in data types or simpleType elements defined in this schema (or another schema indicated by the specified namespace). The simpleType element containing the union element is derived from the simple types specified by the memberTypes value. The values in memberTypes must be qualified names (QNames).

For simple type union definitions, the list of simple types is the union of the contents of memberTypes (which is itself a list of simple types) and each of the child simpleType element definitions under the union element. See the second example later in this topic.

The memberTypes attribute is opposite of the itemType attribute for the list element which is mutually exclusive to the simpleType element child of the list element.

Optional.

Element Information
**Number of occurrences**  One time

**Parent elements**  
- `simpleType`

**Contents**  
- `annotation`
- `simpleType`

## Example

The following example shows a simple type that is a **union** of two simple types.

```xml
<xs:attribute name="fontsize">
  <xs:simpleType>
    <xs:union memberTypes="fontbynumber fontbystringname"/>
  </xs:simpleType>
</xs:attribute>

<xs:simpleType name="fontbynumber">
  <xs:restriction base="xs:positiveInteger">
    <xs:maxInclusive value="72"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="fontbystringname">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>
```

## Example

The following example shows a simple type definition that is a union of all nonnegative integers and the NMTOKEN "unbounded". (This is from the schema in the W3C XML Schema Part 1: Structures specification.)

```xml
<xs:simpleType name="allNNI">
  <xs:annotation>
    <xs:documentation>for maxOccurs</xs:documentation>
  </xs:annotation>
  <xs:union memberTypes="nonNegativeInteger">
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="unbounded"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```
See Also

XML Schema Reference (XSD) | XML Schema Elements


Other Resources

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**unique**

Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope. The value must be unique or nil.

```xml
<unique
    id = ID
    name = NCName
    {any attributes with non-schema Namespace}…>
Content: (annotation?, (selector, field+))
</unique>
```

**Attributes**

**id**

The ID of this element. The **id** value must be of type ID and be unique within the document containing this element.

Optional.

**name**

The name of the **unique** element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.

The name must be unique within an identity constraint set.

Required.

**Element Information**

**Number of occurrences** One time

**Parent elements** element

**Contents** annotation, field, selector

**Remarks**
The **unique** element must contain the following elements in order.

**selector**

The **selector** element contains an XML Path Language (XPath) expression specifying the set of elements across which the values specified by the **field** elements must be unique.

There must be one and only one **selector** element.

Each **field** element contains an XPath expression specifying the values (attribute or element values) that must be unique for the set of elements specified by the **selector** element.

If there is more than one **field** element, the combination of the **field** elements must be unique. In this case, the values for a single **field** element may or may not be unique across the selected elements, but the combination of all the fields must be unique.

There must be one or more **field** element.

**Example**

The following example defines a schema with a root element of **ordersByCustomer**, which can contain an unbounded sequence of **customerOrders** elements. In the **ordersByCustomer** element, the **unique** element specifies that the **customerID** attribute must be unique across all **customerOrders**.

```xml
<xs:schema targetNamespace="http://tempuri.org/myschema/unique" elem
xmlns="http://tempuri.org/myschema/unique
<xs:complexType name="customerOrderType">
  <xs:sequence>
    <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="itemID" type="xs:string"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```
</xs:sequence>
<xs:attribute name="CustomerID" type="xs:string"/>
</xs:complexType>

<xs:element name="ordersByCustomer">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="customerOrders" type="customerOrderType"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:unique name="oneCustomerOrdersforEachCustomerID">
  <xs:selector xpath="mstns:customerOrders"/>
  <xs:field xpath="@customerID"/>
</xs:unique>
</xs:element>
</xs:schema>

See Also

XML Schema Reference (XSD) | XML Schema Elements


Other Resources

W3C Namespaces in XML Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
XML Data Types Reference

The World Wide Web Consortium (W3C) XML Schema Part 2: DataTypes is the specification for defining data types used in XML Schemas. This specification defines built-in primitive data types, derived data types, and facets.

The following diagram shows the type hierarchy.

See Also

XML Schema Reference (XSD) | Primitive XML Data Types | Derived XML Data Types | Data Type Facets
Other Resources

World Wide Web Consortium (W3C) XML Schema Part 2: Datatypes
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
## Primitive XML Data Types

The following table lists primitive XML schema data types, facets that can be applied to the data type, and a description of the data type.

Facets can only appear once in a type definition except for **enumeration** and **pattern** facets. **Enumeration** and **pattern** facets can have multiple entries and are grouped together.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Facets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>length, pattern, maxLength, minLength, enumeration, whiteSpace</td>
<td>Represents character strings.</td>
</tr>
<tr>
<td>Boolean</td>
<td>pattern, whiteSpace</td>
<td>Represents Boolean values, which are either <strong>true</strong> or <strong>false</strong>.</td>
</tr>
<tr>
<td>decimal</td>
<td>enumeration, pattern, totalDigits, fractionDigits, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace</td>
<td>Represents arbitrary precision numbers.</td>
</tr>
<tr>
<td>float</td>
<td>minExclusive, maxInclusive, maxExclusive, whiteSpace</td>
<td>Represents single-precision 32-bit floating-point numbers.</td>
</tr>
<tr>
<td>double</td>
<td>minExclusive, maxInclusive, maxExclusive, whiteSpace</td>
<td>Represents double-precision 64-bit floating-point numbers.</td>
</tr>
<tr>
<td></td>
<td>enumeration, pattern</td>
<td>Represents a duration of time.</td>
</tr>
<tr>
<td></td>
<td>enumeration, pattern, PnYnMnDTnHnMnS, where nY</td>
<td>The pattern for <strong>duration</strong> is PnYnMnDTnHnMnS, where nY</td>
</tr>
</tbody>
</table>
**duration**

- minInclusive
- minExclusive
- maxInclusive
- maxExclusive
- whiteSpace

represents the number of years, \( nM \) the number of months, \( nD \) the number of days, \( T \) the date/time separator, \( nH \) the number of hours, \( nM \) the number of minutes, and \( nS \) the number of seconds.

Represents a specific instance of time.

The pattern for **dateTime** is \( CCYY-MM-DDThh:mm:ss \) where \( CC \) represents the century, \( YY \) the year, \( MM \) the month, and \( DD \) the day, preceded by an optional leading negative (-) character to indicate a negative number. If the negative character is omitted, positive (+) is assumed. The \( T \) is the date/time separator and \( hh \), \( mm \), and \( ss \) represent hour, minute, and second respectively.

Additional digits can be used to increase the precision of fractional seconds if desired. For example, the format \( ss.ss... \) with any number of digits after the decimal point is supported. The fractional seconds part is optional.

This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or to indicate the time zone. For example, the difference between the local time and Coordinated Universal Time, immediately followed by a sign,
+ or -, followed by the difference from UTC represented as hh:mm (minutes is required). If the time zone is included, both hours and minutes must be present.

**time**

- **enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace**

  Represents an instance of time that recurs every day.

The pattern for **time** is hh:mm:ss.sss with optional time zone indicator.

**date**

- **enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace**

  Represents a calendar date.

The pattern for **date** is CCYY-MM-DD with optional time zone indicator as allowed for **dateTime**.

**gYearMonth**

- **enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace**

  Represents a specific Gregorian month in a specific Gregorian year. A set of one-month long, nonperiodic instances.

The pattern for **gYearMonth** is CCYY-MM with optional time zone indicator.

**gYear**

- **enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace**

  Represents a Gregorian year. A set of one-year long, nonperiodic instances.

The pattern for **gYear** is CCYY with optional time zone indicator as allowed for **dateTime**.

Represents a specific Gregorian date that recurs, specifically a day
<table>
<thead>
<tr>
<th>enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace</th>
</tr>
</thead>
<tbody>
<tr>
<td>of the year such as the third of May. A <strong>gMonthDay</strong> is the set of calendar dates. Specifically, it is a set of one-day long, annually periodic instances.</td>
</tr>
<tr>
<td>The pattern for <strong>gMonthDay</strong> is - - MM-DD with optional time zone indicator as allowed for <strong>date</strong>.</td>
</tr>
<tr>
<td>enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace</td>
</tr>
<tr>
<td>Represents a Gregorian day that recurs, specifically a day of the month such as the fifth day of the month. A <strong>gDay</strong> is the space of a set of calendar dates. Specifically, it is a set of one-day long, monthly periodic instances.</td>
</tr>
<tr>
<td>The pattern for <strong>gDay</strong> is - - - DD with optional time zone indicator as allowed for <strong>date</strong>.</td>
</tr>
<tr>
<td>enumeration, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace</td>
</tr>
<tr>
<td>Represents a Gregorian month that recurs every year. A <strong>gMonth</strong> is the space of a set of calendar months. Specifically, it is a set of one-month long, yearly periodic instances.</td>
</tr>
<tr>
<td>The pattern for <strong>gMonth</strong> is - - MM - - with optional time zone indicator as allowed for <strong>date</strong>.</td>
</tr>
<tr>
<td>length, pattern, maxLength, minLength, enumeration, whiteSpace</td>
</tr>
<tr>
<td>Represents arbitrary hex-encoded binary data. A <strong>hexBinary</strong> is the set of finite-length sequences of binary octets. Each binary octet is encoded as a character tuple, consisting of two hexadecimal</td>
</tr>
</tbody>
</table>
digits ([0-9a-fA-F]) representing the octet code.

**base64Binary**

length, pattern, maxLength, minLength, enumeration, whiteSpace

Represents Base64-encoded arbitrary binary data. A **base64Binary** is the set of finite-length sequences of binary octets.

**anyURI**

length, pattern, maxLength, minLength, enumeration, whiteSpace

Represents a URI as defined by RFC 2396. An **anyURI** value can be absolute or relative, and may have an optional fragment identifier.

**QName**

length, enumeration, pattern, maxLength, minLength, whiteSpace

Represents a qualified name. A qualified name is composed of a prefix and a local name separated by a colon. Both the prefix and local names must be an NCName. The prefix must be associated with a namespace URI reference, using a namespace declaration.

**NOTATION**

length, enumeration, pattern, maxLength, minLength, whiteSpace

Represents a **NOTATION** attribute type. A set of QNames.

**See Also**

[XML Schema Element Map](#) | [XML Data Types Reference](#) | [Derived XML Data Types](#) | [Data Type Facets](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
## Derived XML Data Types

The following table lists derived XML schema data types, facets that can be applied to the derived data type, and a description of the derived data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Facets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalizedString</td>
<td>length, pattern, maxLength, minLength, enumeration, whiteSpace</td>
<td>Represents white space normalized strings. This data type is derived from string.</td>
</tr>
<tr>
<td>token</td>
<td>enumeration, pattern, length, minLength, maxLength, whiteSpace</td>
<td>Represents tokenized strings. This data type is derived from normalizedString.</td>
</tr>
<tr>
<td>language</td>
<td>length, pattern, maxLength, minLength, enumeration, whiteSpace</td>
<td>Represents natural language identifiers (defined by RFC 1766). This data type is derived from token.</td>
</tr>
<tr>
<td>IDREFS</td>
<td>length, maxLength, minLength, enumeration, whiteSpace</td>
<td>Represents the IDREFS attribute type. Contains a set of values of type IDREF.</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>length, maxLength, minLength, enumeration, whiteSpace</td>
<td>Represents the ENTITIES attribute type. Contains a set of values of type ENTITY.</td>
</tr>
<tr>
<td>NMToken</td>
<td>length, pattern, maxLength, minLength, enumeration, whiteSpace</td>
<td>Represents the NMToken attribute type. An NMToken is set of name characters (letters, digits, and other characters) in any combination. Unlike Name and NCName,</td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>NMTOKENS</td>
<td>Represents the NMTOKENS attribute type. Contains a set of values of type NMTOKEN.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Represents names in XML. A Name is a token that begins with a letter, underscore, or colon and continues with name characters (letters, digits, and other characters). This data type is derived from token.</td>
<td></td>
</tr>
<tr>
<td>NCName</td>
<td>Represents noncolonized names. This data type is the same as Name, except it cannot begin with a colon. This data type is derived from Name.</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Represents a reference to an element that has an ID attribute that matches the specified ID. An IDREF must be an NCName and must be a value of an ID.</td>
<td></td>
</tr>
<tr>
<td>IDREF</td>
<td>NMTOKEN has no restrictions on the starting character. This data type is derived from token.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td></td>
</tr>
<tr>
<td>maxLength</td>
<td></td>
</tr>
<tr>
<td>minLength</td>
<td></td>
</tr>
<tr>
<td>enumeration</td>
<td></td>
</tr>
<tr>
<td>whiteSpace</td>
<td></td>
</tr>
</tbody>
</table>
minLength, whiteSpace

element or attribute of type ID within the XML document. This data type is derived from NCName.

**ENTITY**

length, enumeration, pattern, maxLength, minLength, whiteSpace

Represents the ENTITY attribute type in XML 1.0 Recommendation. This is a reference to an unparsed entity with a name that matches the specified name. An ENTITY must be an NCName and must be declared in the schema as an unparsed entity name. This data type is derived from NCName.

**integer**

enumeration, fractionDigits, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, totalDigits, whiteSpace

Represents a sequence of decimal digits with an optional leading sign (+ or -). This data type is derived from decimal.

**nonPositiveInteger**

enumeration, fractionDigits, pattern, minInclusive, minExclusive, maxInclusive, maxExclusive, totalDigits, whiteSpace

Represents an integer that is less than or equal to zero. A nonPositiveInteger consists of a negative sign (-) and sequence of decimal digits. This data type is derived from integer.

**negativeInteger**

enumeration, fractionDigits, pattern, minInclusive, minExclusive, maxInclusive,

Represents an integer that is less than zero. Consists of a negative sign (-) and sequence of decimal digits. This data type is
maxExclusive, totalDigits, whiteSpace
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
totalDigits, whiteSpace
enumeration, fractionDigits, pattern,
minInclusive, minExclusive,
maxInclusive, maxExclusive,
<table>
<thead>
<tr>
<th>Data Type</th>
<th>fractionDigits, pattern, minInclusive, maxInclusive, maxExclusive, totalDigits, whiteSpace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsignedLong</td>
<td>a minimum of zero and maximum of 18446744073709551615. This data type is derived from</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nonNegativeInteger.</td>
<td></td>
</tr>
<tr>
<td>unsignedInt</td>
<td>Represents an integer with a minimum of zero and maximum of 4294967295. This data type is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>derived from unsignedLong.</td>
<td></td>
</tr>
<tr>
<td>unsignedShort</td>
<td>Represents an integer with a minimum of zero and maximum of 65535. This data type is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>derived from unsignedInt.</td>
<td></td>
</tr>
<tr>
<td>unsignedByte</td>
<td>Represents an integer with a minimum of zero and maximum of 255. This data type is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>derived from unsignedShort.</td>
<td></td>
</tr>
<tr>
<td>positiveInteger</td>
<td>Represents an integer that is greater than zero. This data type is derived from</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nonNegativeInteger.</td>
<td></td>
</tr>
</tbody>
</table>

See Also
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Data Type Facets

Simple types (both built-in and derived) have facets. A facet is a single defining aspect that helps determine the set of values for a simple type. For example, length, minInclusive, and maxInclusive are common facets for the built-in data types. All of the facets for a simple type define the set of legal values for that simple type.

A facet is defined as an element. Each facet element has a fixed attribute that is a Boolean value. When a simple type is defined, you can prevent derivatives of that type from modifying the value of specified facets. To prevent modification of a facet, add the fixed attribute to the facet and set its value to true.

Facets can only appear once in a type definition except for enumeration and pattern facets. Enumeration and pattern facets can have multiple entries and are grouped together.

Example

The following example shows a simple type with the fixed attribute set to true which prevents the length from having a value other than 7.

```xml
<xs:simpleType name="Postcode">
  <xs:restriction base="xs:string">
    <xs:length value="7" fixed="true"/>
  </xs:restriction>
</xs:simpleType>
```
List of Constraining Facets

The constraining facets (facets that can be used to constrain the values of simple types), their descriptions, and the built-in data types that they apply to, are listed below.

**enumeration**
Specified set of values. This constrains a data type to the specified values. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-enumeration.

**fractionDigits**
Value with specific maximum number of decimal digits in the fractional part. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-fractionDigits.

**length**
Number of units of length. Units of length depend on the data type. This value must be a **nonNegativeInteger**. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-length.

**maxExclusive**
Upper bound value (all values are less than this value). This value must be the same data type as the inherited data type. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-maxExclusive.

**maxInclusive**
Maximum value. This value must be the same data type as the inherited data type. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-maxInclusive.

**maxLength**
Maximum number of units of length. Units of length depend on the data type. This value must be a **nonNegativeInteger**. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at

**minExclusive**
Lower bound value (all values are greater than this value). This value must be the same data type as the inherited data type. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-minExclusive.

**minInclusive**
Minimum value. This value must be the same data type as the inherited data type. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-minInclusive.

**minLength**
Minimum number of units of length. Units of length depend on the data type. This value must be a **nonNegativeInteger**. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-minLength.

**pattern**
Specific pattern that the data type's values must match. This constrains the data type to literals that match the specified pattern. The pattern value must be a regular expression. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-pattern.

**totalDigits**
Value with specific maximum number of decimal digits. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-totalDigits.

**whiteSpace**
Value must be one of **preserve**, **replace**, or **collapse**. The **whiteSpace** facet cannot be changed for most numeric data types. For more information, see the W3C XML Schema Part 2: Datatypes Recommendation at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#element-whiteSpace.

No normalization is performed; the value is not changed for element content as required by the W3C
XML 1.0 Recommendation.

**replace**

All occurrences of \#x9 (tab), \#xA (line feed) and \#xD (carriage return) are replaced with \#x20 (space).

After the processing implied by **replace**, contiguous sequences of \#x20s are collapsed to a single \#x20, and leading and trailing \#x20s are removed.

**collapse**

See Also

[XML Schema Element Map](#) | [XML Data Types Reference](#) | [Primitive XML Data Types](#) | [Derived XML Data Types](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XML Schema Regular Expressions

XML Schema regular expressions are used with the `pattern` facet to provide constraints or restrictions on a string of characters that conform to the defined pattern.

The `pattern` facet is a constraint on the value space of a data type, achieved by constraining the lexical space to literals that match a specific pattern. The value of `pattern` must be a regular expression.

**Pattern** provides for constraining a value space to values that are denoted by literals that match a specific regular expression.

**Example**

The following example defines a user-derived data type that creates a representation of postal codes in the United States, limited to strings that match a specific regular expression.

**Input file: T.xml**

```xml
<zips xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="t.xsd">
  <zip>12345</zip>
  <zip>12345-1234</zip>
</zips>
```

**XML Schema: T.xsd**

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="zips">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="zip" type="better-us-zipcode"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="better-us-zipcode">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{5}(-[0-9]{4})?"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

There are two types of characters that are used in regular expressions: metacharacters and normal characters. A metacharacter is one of the following characters: ., \, ?, *, +, {, }, (, ), [ or ]. These characters have special meaning in regular expressions, but can be escaped to from atoms that denote the sets of strings containing only themselves. An escaped metacharacter has the behavior of a normal character.

A normal character is any XML character that is not a metacharacter. In regular expressions, a normal character is an atom that denotes the singleton set of strings containing only itself.

The following topics describe the XML Schema regular expressions.

- Atom
- Normal Character
- Piece
- Quantifier
- Branch
- Regular Expression
- Character Classes
- Character Class Escapes

To understand the relationships in XML Schema Regular Expressions, see the XML Schema Regular Expressions Reference Chart.

See Also

XML Schema Reference (XSD) | XML Schema Regular Expressions Reference Chart | Atom | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XML Schema Regular Expressions Reference Chart

The following chart shows the relationship of XML Schema Regular Expressions.

\[
\text{regExp} ::= \text{branch} \left(\text{'}|\right) \text{branch} \right) *
\]
\[
\text{branch} ::= \text{piece} *
\]
\[
\text{piece} ::= \text{atom} \text{ quantifier}?
\]
\[
\text{quantifier} ::= [?*+] | (\{'\text{quantity}'\}')
\]
\[
\text{quantity} ::= \text{quantRange} \mid \text{quantMin} \mid \text{QuantExact}
\]
\[
\text{quantRange} ::= \text{QuantExact} ',' \text{QuantExact}
\]
\[
\text{quantMin} ::= \text{QuantExact} ','
\]
\[
\text{QuantExact} ::= [0-9]+ 
\]
\[
\text{atom} ::= \text{Char} \mid \text{charClass} \mid (\{'\text{regExp}'\}')
\]
\[
\text{Char} ::= [^\."?*+()|] | #x5B#x5D
\]
\[
\text{charClass} ::= \text{charClassExpr} \mid \text{charClassExpr}
\]
\[
\text{charClassExpr} ::= [\,' \text{charGroup} ']
\]
\[
\text{charGroup} ::= \text{posCharGroup} \mid \text{negCharGroup} \mid \text{charClassSub}
\]
\[
\text{posCharGroup} ::= (\text{charRange} \mid \text{charClassEsc}) +
\]
\[
\text{negCharGroup} ::= [^]* \text{posCharGroup}
\]
\[
\text{charClassSub} ::= (\text{posCharGroup} \mid \text{negCharGroup}) '-' \text{charClassExpr}
\]
\[
\text{charRange} ::= \text{seRange} \mid \text{XmlCharRef} \mid \text{XmlCharIncDash}
\]
\[
\text{seRange} ::= \text{charOrEsc} '-' \text{charOrEsc}
\]
\[
\text{XmlCharRef} ::= (\{'&\#' \text{[0-9]+} ';\}' \mid \{'\&\#x\text{[0-9a-fA-F]+} ';\}'
\]
\[
\text{charOrEsc} ::= \text{XmlChar} \mid \text{SingleCharEsc}
\]
\[
\text{XmlChar} ::= [^\#/x2D#x5B#x5D]
\]
\[
\text{XmlCharIncDash} ::= [^\#/x5B#x5D]
\]
\[
\text{charClassEsc} ::= (\text{SingleCharEsc} \mid \text{MultiCharEsc} \mid \text{catEsc} \mid \text{complEsc}
\]
\[
\text{SingleCharEsc} ::= '\text{'} [\text{nrt}\text{.} \text{.} ?*+(){}|#x2Dx5B#x5D]
\]
\[
\text{catEsc} ::= '\text{p'}\{\text{charProp}\}'
\]
\[
\text{complEsc} ::= '\text{P'}\{\text{charProp}\}'
\]
\[
\text{charProp} ::= \text{IsCategory} \mid \text{IsBlock}
\]
\[
\text{IsCategory} ::= \text{Letters} \mid \text{Marks} \mid \text{Numbers} \mid \text{Punctuation} \mid \text{Separators}
\]
\[
\text{Symbols} \mid \text{Others}
\]
\[
\text{Letters} ::= 'L' [ultmo]?
\]
\[
\text{Marks} ::= 'M' [nce]?
\]
\[
\text{Numbers} ::= 'N' [dlo]?
\]
\[
\text{Punctuation} ::= 'P' [cdseifo]?
\]
\[
\text{Separators} ::= 'Z' [s1p]?
\]
\[
\text{Symbols} ::= 'S' [mcko]?
\]
\[
\text{Others} ::= '0' [cfon]?
\]
\[
\text{IsBlock} ::= 'Is' [a-zA-Z0-9#x2D]+
MultiCharEsc::='.' | ('\' [sSiIcCdDwW])

See Also

XML Schema Regular Expressions
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Atom

A normal character, a character class, or a parenthesized regular expression.

Syntax

\[
\text{atom :: = Char | charClass | ('(\text{regExp}\'))}
\]

The following table shows valid atoms \( R \) and the description for their set of strings \( L(R) \).

<table>
<thead>
<tr>
<th>Valid atoms ( R ) for normal characters ( c ), character classes ( C ), and regular expressions ( S )</th>
<th>Set of strings ( L(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c )</td>
<td>The single string consisting only of normal characters.</td>
</tr>
<tr>
<td>( C )</td>
<td>All strings in ( L(C) ).</td>
</tr>
<tr>
<td>( (S) )</td>
<td>All strings in ( L(S) ).</td>
</tr>
</tbody>
</table>

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Character Classes | Normal Character | Data Type Facets
Normal Character

Any XML character that is not a metacharacter. In regular expressions, a normal character is an atom that denotes the singleton set of strings containing only itself.

Syntax

Char ::= [^\.?*+|{}\x5b\x5d]

Note that a normal character can be represented either as itself or by using a character reference.

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Atom | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Piece

An atom, either a normal character, a character class, or a parenthesized regular expression, possibly followed by a quantifier.

Syntax

piece :: = atom quantifier?

The following table shows valid pieces \( R \) and their set of strings \( L(R) \).

<table>
<thead>
<tr>
<th>Valid pieces ( R ) for atoms ( S ) and non-negative integers ( n ) and ( m ) such that ( n\leq m )</th>
<th>Set of strings ( L(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>All strings in ( L(S) ).</td>
</tr>
<tr>
<td>( S? )</td>
<td>The empty string, and all strings in ( L(S) ).</td>
</tr>
<tr>
<td></td>
<td>All strings in ( L(S?) ) and all strings ( ST ) with ( S ) in ( L(S*) ) and ( T ) in ( L(S) ).</td>
</tr>
<tr>
<td>( S* )</td>
<td>All concatenations of zero or more strings from ( L(S) ).</td>
</tr>
<tr>
<td></td>
<td>All strings ( ST ) with ( S ) in ( L(S) ) and ( T ) in ( L(S*) ).</td>
</tr>
<tr>
<td>( S+ )</td>
<td>All concatenations of one or more strings from ( L(S) ).</td>
</tr>
<tr>
<td></td>
<td>All strings ( ST ) with ( S ) in ( L(S) ) and ( T ) in ( L(S{n-1, m-1}) ).</td>
</tr>
<tr>
<td>( S{n, m} )</td>
<td>All sequences of at least ( n ), and at most ( m ), strings from ( L(S) ).</td>
</tr>
<tr>
<td></td>
<td>All strings in ( L(S{n,n}) ).</td>
</tr>
</tbody>
</table>
\[ S\{n\} \] All sequences of exactly \( n \) strings from \( L(S) \).

All strings in \( L(S\{n\}S*) \).

\[ S\{n,\} \] All sequences of exactly \( n \) strings from \( L(S) \).

All strings \( ST \) with \( S \) in \( L(S?) \) and \( T \) in \( L(S\{0, m-1\}) \).

\[ S\{0, m\} \] All sequences of at most \( m \), strings from \( L(S) \).

\[ S\{0,0\} \] The set containing only the empty string.

**See Also**

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Atom | Data Type Facets | Quantifier
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Quantifier

One of the following characters or strings: ?, *, +, {n,m}, or {n,}. For more information about these characters or strings, see Piece.

Syntax

quantifier :: = [?*+] | ('{quantity}')

Syntax

quantity :: = quantRange | quantMin | QuantExact

Syntax

quantRange :: = QuantExact ',' QuantExact

Syntax

quantMin :: = QuantExact ','

Syntax

QuantExact :: = [0-9]+

Example

The following are examples of allowed data for regex.

<table>
<thead>
<tr>
<th>Regex</th>
<th>Allowed data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a * x</td>
<td>x, ax, aax, aaax…</td>
</tr>
<tr>
<td>a ? x</td>
<td>ax, x</td>
</tr>
<tr>
<td>a + x</td>
<td>ax, aax, aaax…</td>
</tr>
<tr>
<td>(a</td>
<td>b) + x</td>
</tr>
</tbody>
</table>
See Also

XML Schema Regular Expressions | XML Schema Regular Expressions
Reference Chart | Data Type Facets | Piece
Branch

Consists of zero or more pieces combined.

Syntax

branch ::= piece*

The following table shows valid branches $R$ and their set of strings $L(R)$.

<table>
<thead>
<tr>
<th>Valid branches $R$ for all pieces $S$ and branches $T$</th>
<th>Set of strings $L(R)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>All strings in $L(S)$.</td>
</tr>
<tr>
<td>$ST$</td>
<td>All strings $ST$ with $S$ in $L(S)$ and $T$ in $L(T)$.</td>
</tr>
</tbody>
</table>

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions
Reference Chart | Data Type Facets | Piece
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Regular Expression

A sequence of characters that denote a set of strings (also represented as $L(R)$). When used to constrain a lexical space, a regular expression asserts that only strings in the set of strings are valid literals for values of that type.

A regular expression is composed of zero or more branches that are separated by the "or" (|) character.

Syntax

\
regExp :: = branch ('|' branch) * 

The following table shows valid regular expressions $R$ and their set of strings $L(R)$.

<table>
<thead>
<tr>
<th>Valid regular expressions $R$ for all branches $S$ and regular expressions $T$</th>
<th>Set of strings $L(R)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(empty string)</td>
<td>The set containing only the empty string.</td>
</tr>
<tr>
<td>S</td>
<td>All strings in $L(S)$.</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
</tr>
</tbody>
</table>

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Branch | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Character Classes

A character class is an atom that identifies a set of characters. The set of strings denoted by a character class contains one single character string for each character in the set of characters.

The following are character classes.

- Character Class
- Character Class Expression
- Character Group
- Positive Character Group
- Negative Character Group
- Character Class Subtraction
- Character Range

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Atom | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Character Class

Either a character class escape or a character class expression.

Syntax

charClass ::= charClassEsc | charClassExpr

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Character Class Escapes | Character Class Expression | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Character Class Expression

A character group surrounded by left bracket ([]) and right bracket (]) characters. For all character groups $G$, $G$ is a valid character class expression that identifies the set of characters.

Syntax

charClassExpr ::= '][' charGroup ']

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Character Group | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**Character Group**

Either a positive character group, a negative character group, or a character class subtraction.

**Syntax**

charGroup ::= posCharGroup | negCharGroup | charClassSub

**See Also**

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Data Type Facets | Positive Character Group | Negative Character Group | Character Class Subtraction
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Positive Character Group

Consists of one or more character ranges or character class escapes that are concatenated together. A positive character group identifies the set of characters containing all of the characters in all of the sets that are identified by its constituent ranges or escapes.

**Syntax**

$$\text{posCharGroup ::= ( charRange | charClassEsc ) +}$$

The following table shows valid positive character groups and their descriptions.

<table>
<thead>
<tr>
<th>Valid positive character groups $G$ for all character ranges $R$, character class escapes $E$, and positive character groups $P$</th>
<th>Set of characters $C(G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>All characters in $C(R)$.</td>
</tr>
<tr>
<td>$E$</td>
<td>All characters in $C(E)$.</td>
</tr>
<tr>
<td>$RP$</td>
<td>All characters in $C(R)$ and all characters in $C(P)$.</td>
</tr>
<tr>
<td>$EP$</td>
<td>All characters in $C(E)$ and all characters in $C(P)$.</td>
</tr>
</tbody>
</table>

**See Also**

[XML Schema Regular Expressions](#) | [XML Schema Regular Expressions](#) | [Reference Chart](#) | [Character Class Escapes](#) | [Character Range](#) | [Data Type Facets](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Negative Character Group

A positive character group that is preceded by the caret (^) character. For all positive character groups $P$, $^P$ is a valid negative character group, and $c(^P)$ contains all XML characters that are not in $C(P)$. 

Syntax

$$\text{negCharGroup ::= } '^' \ \text{posCharGroup}$$

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Character Group | Data Type Facets | Positive Character Group
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Character Class Subtraction

A character class expression subtracted from a positive character group or negative character group using the subtraction (-) character.

Syntax

```
charClassSub ::= ( posCharGroup | negCharGroup ) '-' charClassExpr
```

For any positive character group or negative character group $G$, and any character class expression $C$, $G-C$ is a valid character class subtraction that identifies the set of all characters in $C(G)$ that are not also in $C(C)$.

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Negative Character Group | Data Type Facets | Positive Character Group
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Character Range

A character range $R$ identifies a set of characters $C(R)$ containing all XML characters with Universal Character Set (UCS) code points in a specified range.

Syntax

charRange ::= seRange | XmlCharRef | XmlCharIncDash

Syntax

seRange ::= charOrEsc '-' charOrEsc

Syntax

XmlCharRef ::= ( '&#' [0-9]+ ';' ) | ('&#x' [0-9a-fA-F]+ ';')

Syntax

charOrEsc ::= XmlChar | SingleCharEsc

Syntax

XmlChar ::= [^\#x2D#x5B#x5D]

Syntax

XmlCharIncDash ::= [^\#x5B#x5D]

A single XML character is a character range that identifies the set of characters containing only itself. All XML characters are valid character ranges except as follows:

- The left bracket (\[), right bracket (\]), and backslash (\) character range are not valid character ranges.
- The caret (^) character is a valid character range at the beginning of a
positive character group only if it is part of a negative character group.

- The subtraction (−) character is a valid character range only at the beginning or end of a positive character group.

A character range may also be written in the form s-e, identifying the set that contains all XML characters with UCS code points that are greater than or equal to the code point of s, but not greater than the code point of e.

The form s-e is a valid character range if:

- s is a single character escape, or an XML character.
- s is not "\".
- s is the first character in a character class expression, then s is not "^".
- e is a single character escape, or an XML character.
- e is not "\" or "[;".
- the code point of e is greater than or equal to the code point of s.

**Note** The code point of a single character escape is the code point of the single character in the set of characters that it identifies.

**See Also**

[XML Schema Regular Expressions](#) | **XML Schema Regular Expressions Reference Chart** | Negative Character Group | Data Type Facets | Positive Character Group
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Character Class Escapes

A character class escape is a short sequence of characters that identifies a predefined character class.

The following are character class escapes.

- Character Class Escape
- Single Character Escape
- Category Escape
- Categories
- Block Escape
- Multi-Character Escape

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Character Class | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Character Class Escape

A short sequence of characters that identifies a predefined character class.

Syntax

charClassEsc ::= ( SingleCharEsc | MultiCharEsc | catEsc | complEsc

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Character Class | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Single Character Escape

Identifies a set containing only one character, usually because that character is difficult or impossible to write directly into a regular expression.

Syntax

SingleCharEsc ::= '\' [nrt\|.?+(){}] \#x2D\#x5B\#x5D\#x5E]

The following table shows valid single character escapes and their descriptions.

<table>
<thead>
<tr>
<th>Valid single character escapes</th>
<th>Set of characters C(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>The new line character (\xA).</td>
</tr>
<tr>
<td>\r</td>
<td>The return character (\xD).</td>
</tr>
<tr>
<td>\t</td>
<td>The tab character (\x9).</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>^</td>
<td>^</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
</tr>
<tr>
<td>[</td>
<td>[</td>
</tr>
<tr>
<td>]</td>
<td>]</td>
</tr>
</tbody>
</table>

See Also
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schem
Category Escape

A set that contains all characters that have property $X$ that can be identified with a category escape $\p{X}$. The complement of this set is specified with the category escape $\P{X}$. ($[\P{X}]=[\^\p{X}]$).

Syntax

catEsc ::= '\p{ charProp }'

Syntax

complEsc ::= '\P{ charProp }'

Syntax

charProp ::= IsCategory | IsBlock

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Categories

The Unicode Character Database specifies a number of possible values for the General Category property and provides mappings from code points to specific character properties. The tables below specify the recognized values of the General Category property.
IsCategory

Syntax

IsCategory ::= Letters | Marks | Numbers | Punctuation | Separators
Letters

Syntax

Letters ::= 'L' [ultmo]?

The following table shows the properties for letters.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>All letters</td>
</tr>
<tr>
<td>Lu</td>
<td>Uppercase</td>
</tr>
<tr>
<td>Ll</td>
<td>Lowercase</td>
</tr>
<tr>
<td>Lt</td>
<td>Titlecase</td>
</tr>
<tr>
<td>Lm</td>
<td>Modifier</td>
</tr>
<tr>
<td>Lo</td>
<td>Other</td>
</tr>
</tbody>
</table>
Marks

Syntax

Marks ::= 'M' [nce]?

The following table shows the properties for marks.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>All marks</td>
</tr>
<tr>
<td>Mn</td>
<td>Nonspacing</td>
</tr>
<tr>
<td>Mc</td>
<td>Space combining</td>
</tr>
<tr>
<td>Me</td>
<td>Enclosing</td>
</tr>
</tbody>
</table>
Numbers

Syntax

Numbers ::= 'N' [dlo]?

The following table shows the properties for numbers.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>All numbers</td>
</tr>
<tr>
<td>Nd</td>
<td>Decimal digit</td>
</tr>
<tr>
<td>Nl</td>
<td>Letter</td>
</tr>
<tr>
<td>No</td>
<td>Other</td>
</tr>
</tbody>
</table>
**Punctuation**

**Syntax**

\texttt{Punctuation ::= 'P' [cdseifo]?

The following table shows the properties for punctuation.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>All punctuation</td>
</tr>
<tr>
<td>Pc</td>
<td>Connector</td>
</tr>
<tr>
<td>Pd</td>
<td>Dash</td>
</tr>
<tr>
<td>Ps</td>
<td>Open</td>
</tr>
<tr>
<td>Pe</td>
<td>Close</td>
</tr>
<tr>
<td>Pi</td>
<td>Initial quote (may behave like \textit{Ps} or \textit{Pe} depending on usage)</td>
</tr>
<tr>
<td>Pf</td>
<td>Final quote (may behave like \textit{Ps} or \textit{Pe} depending on usage)</td>
</tr>
<tr>
<td>Po</td>
<td>Other</td>
</tr>
</tbody>
</table>
Separators

Syntax

Separators ::= 'Z' [slp]?

The following table shows the properties for separators.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>All separators</td>
</tr>
<tr>
<td>Zs</td>
<td>Space</td>
</tr>
<tr>
<td>Zl</td>
<td>Line</td>
</tr>
<tr>
<td>Zp</td>
<td>Paragraph</td>
</tr>
</tbody>
</table>
Symbols

Syntax

Symbols ::= 'S' [mcko]?

The following table shows the properties for symbols.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>All symbols</td>
</tr>
<tr>
<td>Sm</td>
<td>Math</td>
</tr>
<tr>
<td>Sc</td>
<td>Currency</td>
</tr>
<tr>
<td>Sk</td>
<td>Modifier</td>
</tr>
<tr>
<td>So</td>
<td>Other</td>
</tr>
</tbody>
</table>
Others

Syntax

Others ::= '0' [cfon]?

The following table shows the properties for others.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>All others</td>
</tr>
<tr>
<td>Cc</td>
<td>Control</td>
</tr>
<tr>
<td>Cf</td>
<td>Format</td>
</tr>
<tr>
<td>Co</td>
<td>Private use</td>
</tr>
<tr>
<td>Cn</td>
<td>Not assigned</td>
</tr>
</tbody>
</table>

**Note**  The Cs property is not included here. The Cs property identifies surrogate characters that do not occur at the level of character abstraction that XML instance documents use.

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Block Escape

The Unicode Character Database groups code points into a number of blocks such as Basic Latin (ASCII) and so on. The set containing all characters that have block name X (with all white space stripped) can be identified with a block escape \p{IsX}. The complement of this set is specified with the block escape \P{IsX}.

For more information, see the Appendix F Regular Expressions in the W3C XML Schema Datatypes specification (http://www.w3.org/TR/xmlschema-2).

Syntax

IsBlock ::= 'Is' [a-zA-Z0-9#x2D]+

Example

The following example shows a block escape to identify the ASCII character for Basic Latin.

\p{IsBasicLatin}

See Also

XML Schema Regular Expressions | XML Schema Regular Expressions Reference Chart | Data Type Facets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Multi-Character Escape

Provides a simple way to identify a commonly used set of characters.

**Syntax**

MultiCharEsc ::= '.' | ('\' [sSiIcCdDwW])

<table>
<thead>
<tr>
<th>Character Sequence</th>
<th>Equivalent character class</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>[\n\r]</td>
</tr>
<tr>
<td>\s</td>
<td>[#x20\n\r]</td>
</tr>
<tr>
<td>\S</td>
<td>[^\s]</td>
</tr>
<tr>
<td>\i</td>
<td>The set of initial name characters, those matched by Letter</td>
</tr>
<tr>
<td>\I</td>
<td>[^\i]</td>
</tr>
<tr>
<td>\c</td>
<td>The set of name characters, those matched by NameChar.</td>
</tr>
<tr>
<td>\C</td>
<td>[^\c]</td>
</tr>
<tr>
<td>\d</td>
<td>\p{Nd}</td>
</tr>
<tr>
<td>\D</td>
<td>[^\d]</td>
</tr>
<tr>
<td></td>
<td>[#x0000=#x10FFFF]-</td>
</tr>
<tr>
<td>\w</td>
<td>[\p{P}\p{Z}\p{C}] (all characters except the set of punctuation, separator, and other characters)</td>
</tr>
<tr>
<td>\W</td>
<td>[^\w]</td>
</tr>
</tbody>
</table>

**Note** The regular expression language defined here does not attempt to provide a general solution to regular expressions over the Universal Character Set (UCS) character sequences. The language is targeted at the support of "Level 1" features as defined in the Unicode Regular Expressions Guidelines (http://www.unicode.org/unicode/reports/tr18/).

**See Also**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
# XML Schema Instance Attributes

The XML Schema definition language (XSD) defines four attributes for use in XML instance documents. These attributes are in the `http://www.w3.org/2001/XMLSchema-instance` namespace. These attributes are often prefixed with `xsi`, although any prefix can be used and mapped to the `http://www.w3.org/2001/XMLSchema-instance` namespace URI.

The following table describes the XML Schema instance attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>When true, specifies that an element has missing content.</td>
</tr>
<tr>
<td></td>
<td>Associates a schema document that has no target namespace with an instance document.</td>
</tr>
<tr>
<td>noNamespaceSchemaLocation</td>
<td>Associates a schema document that has a target namespace with an instance document.</td>
</tr>
<tr>
<td>schemaLocation</td>
<td>Specifies the type of an element.</td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>

## See Also

[XML Schema Reference](#) | [XML Schema Elements](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
nil

A Boolean value that indicates if an element should contain content.

<xsi:nil="boolean">

Values

true
The element must be empty and the element declaration in the schema must have an annotation where the nillable attribute is set to true.

false
The element cannot be empty. This is the default value.

Example

This example uses an XML Schema, person.xsd, and an instance document, person.xml. The schema contains a type Person that has element declarations for name and height elements. The height element declaration includes a nillable attribute with a value of true. It also contains an element declaration that maps the person element, with its name and height child elements that map to the com:People namespace. The height element includes an xsi:nil attribute with its value set to true to indicate that the element should not contain content.

<!-- person.xsd -->
<x:schema xmlns:xs= "http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:contoso-com:People"
    xmlns:ns="urn:contoso-com:People">

    <xs:complexType name="Person">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="height" type="xs:double" nillable="true"
        </xs:sequence>
    </xs:complexType>

    <xs:element name="Person" type="ns:Person" />
</xs:schema>

<!--person.xml -->
<p:person xmlns:p="urn:contoso-com:People">
    <name>John</name>
    <height xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
</p:person>

See Also

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
noNamespaceSchemaLocation

The noNamespaceSchemaLocation attribute references an XML Schema document that does not have a target namespace.

<xsi:noNamespaceSchemaLocation="anyURI" />

Attributes

anyURI

   The location of a schema that contains unqualified (a schema without a namespace) schema constructs.

Remarks

An XML Schema is not required to have a namespace. To specify the location for an XML Schema that does not have a target namespace, use the noNamespaceSchemaLocation attribute. The XML Schema referenced in this attribute cannot have a target namespace. Because this attribute does not take a list of URLs, you can only specify one schema location.

According to the World Wide Web Consortium (W3C) XML Schema Recommendation, XML instance documents can have both xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes specified.

Example

The following example shows the use of the xsi:noNamespaceSchemaLocation attribute on an instance document that contains unqualified elements.

 <name>John</name>
 <height>59</height>
</person>

See Also
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
schemaLocation

The **schemaLocation** attribute references an XML Schema document that has a target namespace.

```
<xsi:schemaLocation="list of anyURI" />
```

**Attributes**

anyURI (list)

The location of a schema that contains qualified (a schema with a namespace) schema constructs. The first URI reference in each pair is a namespace name, and the second is the location of a schema that describes that namespace.

**Remarks**

This attribute associates an XML Schema document that has a target namespace with an instance document.

An XML Schema is not required to have a namespace. To specify the location for an XML Schema that does not have a target namespace, use the `noNamespaceSchemaLocation` attribute. The XML Schema referenced in this attribute cannot have a target namespace. Because this attribute does not take a list of URLs, you can only specify one schema location.

Associates a schema document that have a target namespace with an instance document. Multiple pairs of URI references can be listed, each with a different namespace name part.

According to the World Wide Web Consortium (W3C) XML Schema Recommendation, XML instance documents can have both `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes specified. In addition, you can list the same namespace several times.

**Example**
The following example shows the use of `xsi:schemaLocation` attribute that provides location information for several XML Schema documents.

```xml
<p:Person
    xmlns:p="http://contoso.com/People"
    xmlns:v="http://contoso.com/Vehicles"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://contoso.com/People
                         http://contoso.com/schemas/people.xsd
                         http://contoso.com/schemas/Vehicles
                         http://contoso.com/schemas/vehicles.xsd
                         http://contoso.com/schemas/People
                         http://contoso.com/schemas/people.xsd">
    <name>John</name>
    <age>28</age>
    <height>59</height>
    <v:Vehicle>
        <color>Red</color>
        <wheels>4</wheels>
        <seats>2</seats>
    </v:Vehicle>
</p:Person>
```

**See Also**

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
type

Specifies the type of an element. This attribute labels an element as a particular type, even though there might not be an element declaration in the schema binding that element to the type.

<xsi:type="QName"/>

Attributes

QName
The name of the data type that is being substituted for the element's declared data type.

Remarks

The type attribute is also used when derived complex types are used in instance documents instead of the expected base type.

Example

The following example shows the use of type attribute. The example uses a schema document, person.xsd, and an instance document, person.xml. The schema document contains a base type Person, a derived type Employee, and an element declaration person. The instance document shows the use of the xsi:type attribute to specify that the person element in the urn:contoso-com:People namespace is of the type Employee in the same namespace.

<!-- person.xsd -->
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema"
   targetNamespace="urn:contoso-com:People"
   xmlns:ns="urn:contoso-com:People">
   <x:s:element name="person" type="ns:Person"/>
   <x:s:complexType name="Person">
     <x:s:sequence>
       <x:s:element name="name" type="xs:string"/>
       <x:s:element name="height" type="xs:double"/>
     </x:s:sequence>
   </x:s:complexType>
</x:schema>
<xs:complexType>
  <xs:complexContent>
    <xs:extension base="ns:Person">
      <xs:sequence>
        <xs:element name="jobDescription" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- person.xml -->
<p:Person
  xmlns:p="urn:contoso-com:People"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="p:Employee">
  <name>John</name>
  <height>59</height>
  <jobDescription>manager</jobDescription>
</p:Person>

See Also

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XML-Data Reduced (XDR) schema is a subset of ideas described in the XML-Data specification, specifically the World Wide Web Consortium (W3C) XML-Data Note and the Document Content Description (DCD) initiative for XML.

MSXML 2.0 and later provide XDR schema support.

The following topics provide more information about XDR schemas.

- Referencing XDR Schemas in XML Documents
- Defining Elements and Attributes
- Content Model
- Using XDR Schema Data Types
- Extensibility
- XDR Schemas and the DOM
- XDR Schemas and Namespaces
- Using Inline XDR Schemas
- Representing the XDR Schema as a DTD

Other Resources

- XML-Data Note
- Document Content Description (DCD) for XML
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Referencing XDR Schemas in XML Documents

To reference an XML-Data Reduced (XDR) schema from an XML document, include a schema namespace declaration, as in the following XML document.

Example

The following example shows an XML document that references an XDR schema, yourschema.xml.

```xml
<book xmlns="x-schema:yourschema.xml">
  <title>Presenting XML</title>
  <author>Richard Light</author>
</book>
```

In MSXML 2.0 and later, a schema can be included inline within the document that references it. The pound sign (#) before the schema name indicates that the schema is inline.

Example

The following example shows an XML document that references an inline XDR schema.

```xml
<root xmlns:data="x-schema:#inLineSchema">
  <Schema name="#inLineSchema"
    xmlns="urn:schemas-microsoft-com:xml-data">
    <ElementType name="myelement"/>
  </Schema>
  <data:myelement>inline example</data:myelement>
</root>
```

See Also

Using Inline XDR Schemas
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Defining Elements and Attributes

When defining schema for a particular class of documents, you specify which elements and attributes are allowed in a complying XML document, and how those elements and attributes are related to each other. In the XML-Data Reduced (XDR) schema, specifying an `ElementType` element and an `AttributeType` element defines the elements and attributes, respectively. You can then declare an instance of an element or an attribute using `element` or `attribute` element tags.

Example

The following example shows an XDR schema that defines four elements: `title`, `author`, `pages`, and `book` using the `ElementType` element. The definition of the `book` element includes the content model for that element, which specifies that each `book` element contains `title`, `author`, and `pages` child elements. This content model is specified using the `element` element along with the `type` attribute that references the element type defined earlier.

The schema also declares an attribute, "copyright", for the `book` element by using the `AttributeType` element, which defines an attribute type, and then declares it using the `attribute` element.

```xml
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="title"/>
  <ElementType name="author"/>
  <ElementType name="pages"/>
  <ElementType name="book" model="closed">
    <element type="title"/>
    <element type="author"/>
    <element type="pages"/>
  </ElementType>
  <AttributeType name="copyright"/>
  <attribute type="copyright"/>
</ElementType>
</Schema>
```

Data file
Specifying global AttributeType elements

You can specify the AttributeType element globally by placing it outside the context of any ElementType element. This allows multiple elements to share the definition of a common attribute.

Example

The following example shows an XDR schema with multiple elements that share the definition of the "copyright" attribute.

```xml
<?xml version="1.0"?>
<Schema xmlns="schemas-microsoft-com:xml-data">
  <ElementType name="title"/>
  <ElementType name="author"/>
  <ElementType name="pages"/>
  <AttributeType name="copyright"/>
  <ElementType name="book" model="closed">
    <element type="title"/>
    <element type="author"/>
    <element type="pages"/>
    <attribute type="copyright"/>
  </ElementType>
</Schema>
```

Data file

```xml
<r xmlns:a="x-schema:abovefile.xml">
  <a:book copyright="">
    <a:title/>
    <a:author/>
    <a:pages/>
  </a:book>
</r>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Content Model

The content model describes the content structure of elements and attributes in the XML document. To describe the content structure, you can use the `model`, `minOccurs`, `maxOccurs`, `order`, `content`, `minLength`, `maxLength`, `default`, and `type` attributes.

In this section, the content model of an element is described first and then the content model of an attribute is described. Attributes used to describe the content model are introduced.
Specifying the Content Model of an Element

When specifying the content model of an element:

- Describe the child elements and attributes that can appear in the element.
- Specify whether the element can include text and elements.
- Specify the order in which the child elements can appear in an instance of the element.

Example

The following example describes the content model and elements for an XDR schema, BookSchema.xml.

```xml
<s:Schema xmlns:s="urn:schemas-microsoft-com:xml-data">
  <s:ElementType name="title" content="mixed"/>
  <s:ElementType name="authors" content="textOnly"/>
  <s:ElementType name="pages" content="textOnly"/>
  <s:ElementType name="book" order="seq" content="eltOnly">
    <s:element type="pages"/>
    <s:element type="title"/>
    <s:element type="authors"/>
  </s:ElementType>
</s:Schema>
```

The content model for the title, author, and pages elements is straightforward. Because the content attribute specifies the elements as textOnly, these elements contain only text and nothing else (that is, no child elements).

The content model for the book element is more complex. The content attribute for the book element is eltOnly. This value means that the book element can contain only the elements specified (title, author, and pages) in the schema. Furthermore, for each book element instance in the document, the child elements must be in the order specified in the schema, validated by setting the order attribute to the value of seq.

The following is a valid instance of the XML document.

```xml
<x:book xmlns:x="x-schema:BookSchema.xml">
  <x:pages>474</x:pages>
```
Alert  Elements with content="empty" and model="open" are not allowed.

The following paragraphs describe the attributes used by the content model.

**The model Attribute (Open and Closed Content Models)**

The content model attribute of an element can have the values open or closed. In an open content model, an element in an XML document can have additional child elements and attributes that are not declared in the XDR schema that the document references. Conversely, a closed model is one in which the document cannot include any information that does not follow the rules specified in the referenced schema.

By default, the content model for an XDR schema is open. This provides an XDR schema with the extensibility that is not present in a document type definition (DTD), which is a closed model. When using a DTD, a document cannot include any information that does not follow the rules specified in the referenced DTD.

For example, consider the following document fragment.

```xml
<x:book xmlns:x="x-schema:BookSchema.xml"
    xmlns:y="urn:some-new-namespace">
    <x:title y:id="123">Applied XML: A Toolkit for Programmers</x:title>
    <x:authors>Alex Ceponkus and Faraz Hoodbhoy</x:authors>
    <y:publisher>Wiley Computer Publishing</y:publisher>
</x:book>
```

This fragment refers to the BookSchema.xml schema presented earlier. Because this schema specifies an open content model (the default), this document fragment is valid even though it has additional elements and attributes not specified in BookSchema.xml (such as the id attribute in the title element and the publisher child element, both of which are defined in the urn:some-new-namespace namespace).

In an open content model, the following constraints apply:

- Content that breaks the existing content model cannot be added or removed.
For example, BookSchema.xml defines the **book** element as a sequence of three elements. Therefore, you must first provide that exact sequence of elements before adding any open content. Thus, you cannot remove the **pages** element, nor can you provide two **title** elements next to each other. Doing so will cause validation to fail.

- Undeclared elements can be added as long as they are defined in a different namespace.
- After satisfying the content model for the schema, other elements can be added. For example, an XML document will validate even if a second **title** element is added after the **pages** element.

Specify a closed content model if you do not want the default value of **open** to be used. In this case, you use the **model** attribute for the **ElementType**, as in the following example.

```xml
<x:ElementType name="book" model="closed">
```

This indicates that a **book** element can only contain the content specified — the **title**, **author**, and **pages** elements — in the schema. Using this setting, the extended elements in the preceding XML fragment would prevent the document from being validated.

### The content Attribute

When describing the content model, you use the **content** attribute to specify whether an element can contain only text (**content="textOnly"**), only elements (**content="eltOnly"**), a mixture of text and elements (**content="mixed"**), or nothing at all (**content="empty"**). The default value for this attribute is **mixed**.

If the **ElementType** has a data type specified (such as **date**, **number**, and so on), the element is assumed to contain only text (the **textOnly** value is implied for the **content** attribute).

For example, the following XDR schema defines a **book** element that contains three child elements: **title**, **author**, and **pages**. These child elements can contain only text because the **content** attribute for each of these elements is **textOnly**. The **book** element can contain only elements and no text because its **content** attribute is **eltOnly**. This **book** element uses a closed content model (**model="closed"**). Thus, the **book** element can contain only these three child
elements and no text or additional subelements and attributes.

```xml
<?xml version="1.0"?>
<s:Schema xmlns:s="urn:schemas-microsoft-com:xml-data">
  <s:ElementType name="title" content="textOnly" />
  <s:ElementType name="author" content="textOnly" />
  <s:ElementType name="pages" content="textOnly" />
  <s:ElementType name="book" content="eltOnly" model="closed">
    <s:element type="title" />
    <s:element type="author" />
    <s:attribute type="pages" />
    <s:AttributeType name="copyright" />
  </s:ElementType>
</s:Schema>
```

If the `content` attribute for an element is **empty**, that element cannot contain any text or child elements; however, it can have attributes. A **mixed** element, on the other hand, can contain text and child elements.

**The minOccurs and maxOccurs Attributes**

You can specify how many times a child element can appear within its parent element by using the **minOccurs** and **maxOccurs** attributes.

```xml
<element type="Item" maxOccurs="*" />
```

The **maxOccurs** attribute is a constraint rule, specifying the maximum number of times that a child element may appear. Valid values for **maxOccurs** are 1 and "*"; the "*" indicates that an unrestricted number of elements can appear. The default value for **maxOccurs** is "1"; however, when **content="mixed"**, the default value is "*".

In a similar way, you can specify a minimum number of times a child element can appear with **minOccurs**. The valid values for **minOccurs** are "0" and "1". For example, to make a child element optional in the instance of a parent element, set **minOccurs** to "0". The default value for **minOccurs** is "1".

In the following schema, the **author** element sets **maxOccurs** to "*" while the **pages** element sets the **minOccurs** attribute to "0".

```xml
<?xml version="1.0"?>
```
Based on the preceding example schema, the following is a valid document instance.

```xml
<root>
  <book>
    <title>C Programming</title>
    <author>Author A</author>
    <author>Author B</author>
    <pages>300</pages>
  </book>
  <book>
    <title>Java Programming</title>
    <author>Author C</author>
  </book>
</root>
```

**The minLength and maxLength Attributes**

You can specify a data type for an element or attribute by using the `urn:schemas-microsoft-com:datatypes` namespace. The `minLength` and `maxLength` attributes defined in this namespace can be used to constrain the length of a string, number, bin.hex, or bin.base64 data type.

- For string and number data types, `maxLength` specifies the maximum number of characters allowed, while `minLength` specifies the minimum number of characters.
- For bin.hex and bin.base64, `maxLength` sets the maximum number of bytes of the binary object, while `minLength` sets the minimum number of bytes.
Regardless of the data type, both length attributes are inclusive. That is, the data can be as long as the length set by `maxLength`, but no longer, and as short as the length set by `minLength`, but no shorter.

The `maxLength` and `minLength` attributes are enforced at parse time and runtime. Valid parent elements are `ElementType`, `AttributeType`, and `datatype`.

The following XDR schema shows an example of how to use these length attributes. This schema specifies the content model for the `userID`, `password`, and `LoginInfo` elements. However, only the content model of the `password` element specifies a minimum and maximum length (6 to 8 characters) for the required string.

```xml
<?xml version="1.0"?>
<s:Schema xmlns:s="urn:schemas-microsoft-com:xml-data"
         xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <s:AttributeType name="userID"
                   xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:type="string" />
  <s:AttributeType name="password"
                   dt:type="string"
                   dt:minLength="6"
                   dt:maxLength="8"/>
  <s:ElementType name="LoginInfo">
    <s:attribute type="userID" />
    <s:attribute type="password" />
  </s:ElementType>
</s:Schema>
```

Using the preceding schema, these two documents are valid instances.

```xml
<LoginInfo userID="1" password="xyz123" />
<LoginInfo userID="2" password="" />
```

Although the password attribute value in the second instance is less than the `minLength`, the instance is still valid because a password value specified as "" is treated the same as if the password attribute is not specified.

**The order Attribute**

When specifying the content model, you use the `order` attribute to specify how sequences of elements can appear in the document instance. You can assign the `order` attribute the value of `seq`, `one`, or `many`.
The **seq** value indicates that the enclosed elements must appear in the same order as they appear in the schema, as in the following example.

```xml
<ElementType name="PurchaseOrder" order="seq">
    <element type="PONumber"/>
    <element type="PODate"/>
    <element type="ShipAddress"/>
</ElementType>
```

The **one** value specifies that only one of the child elements defined in an **ElementType** can appear. For example, to specify that an **Item** element can contain either a **product** element or a **backOrderedProduct** element, but not both, the schema can be specified as follows.

```xml
<ElementType name="Item" order="one">
    <element type="product"/>
    <element type="backOrderedProduct"/>
</ElementType>
```

The **many** value specifies that the child elements can appear in any order, and in any quantity.

The default value for the **order** attribute depends on the content model in use. When the **content** attribute is not set or set to the value of **mixed**, the default value for **order** is **many**. When the **content** attribute is set to **eltOnly**, the default value for **order** is **seq**. Unexpected problems can occur when the **order** attribute is not set and the default value is inserted.

For example, in this schema the **order** attribute is not specified.

```xml
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
    <ElementType name="PONumber"/>
    <ElementType name="PODate"/>
    <ElementType name="ShipAddress"/>
    <ElementType name="product"/>
    <ElementType name="backOrderedProduct"/>
    <ElementType name="PurchaseOrder">
        <element type="PONumber" maxOccurs="1"/>
        <element type="PODate" minOccurs="1"/>
        <element type="ShipAddress" minOccurs="1" maxOccurs="*"/>
        <element type="Item"/>
    </ElementType>
    <ElementType name="Item" content="eltOnly">
        <element type="product"/>
    </ElementType>
</Schema>
```
The schema appears to the parser as follows.

```xml
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
    <ElementType name="PONumber"/>
    <ElementType name="PODate"/>
    <ElementType name="ShipAddress"/>
    <ElementType name="product"/>
    <ElementType name="backOrderedProduct"/>
    <ElementType name="PurchaseOrder" content="mixed" order="many">
        <element type="PONumber" maxOccurs="1" />
        <element type="PODate" minOccurs="1" />
        <element type="ShipAddress" minOccurs="1" maxOccurs="*" />
        <element type="Item"/>
    </ElementType>
    <ElementType name="Item" content="eltOnly" order="seq">
        <element type="product" />
        <element type="backOrderedProduct" />
    </ElementType>
</Schema>
```

Using the preceding schema will not produce the expected validation of the document. Now multiple instances of the element can be entered without an error occurring, as shown in the following XML document example.

```xml
<?xml Version="1.0"?>
<PurchaseOrder xmlns="x-schema:PurchaseOrder.xml " schemaLocation="h
 <!-- The following lines would validate against the schema -->
   <PONumber>1234</PONumber>
   <PONumber>1235</PONumber>
   <ShipAddress>555 Nowhere Blvd</ShipAddress>
<!-- The following lines would not validate against the schema -->
   <Item>
     <backOrderedProduct>firstB0product</backOrderedProduct>
     <product>firstprod</product>
   </Item>
</PurchaseOrder>
```

The `order` attribute is valid for either an `ElementType` or `group` element.

**The `<group>` Element**
The `group` element enables you to specify constraints on a subset of child elements. This can be quite useful when specifying elements within a schema.

The `group` element accepts the `order`, `minOccurs`, and `maxOccurs` attributes.

For example, the following schema defines the `Item` element as containing a `group` element with two child elements, `product` and `backOrderedProduct`. Because this `group` element sets its `order` attribute to `one`, only one of these child elements can appear in the `Item` element. This prevents the `Item` element from having both a `product` and a `backOrderedProduct` element. The `Item` element can have only one of these child elements.

```xml
<ElementType name="Item">
  <group order="one">
    <element type="product" />
    <element type="backOrderedProduct" />
  </group>
  <element type="quantity"/>
  <element type="price"/>
</ElementType>
```

Using the preceding schema, the following document is a valid instance.

```xml
<Item>
  <product>CD</product>
  <quantity>100</quantity>
  <price>10</price>
</Item>
<Item>
  <product>FloppyDisk</product>
  <quantity>100</quantity>
  <price>1</price>
</Item>
```
Specifying the Content Model of an Attribute

The **AttributeType** element specifies the type of attribute used within elements. Using this element, you can even specify whether the attribute is required for the element, as in the following.

```xml
<AttributeType name="shipTo" dt:type="idref" required="yes"/>
```

The **attribute** element specifies instances of an attribute defined within the **AttributeType** element. You use the **attribute** element within an **ElementType** element.

Attributes are more limited in some ways than elements. For example, attributes cannot contain child elements, and you cannot require attributes to appear in any particular order; nor can you pose alternatives, such as a "product" or a "backOrderedProduct". You can specify whether an attribute is required or optional, but an attribute can appear only once per element.

At the same time, attributes have the following capabilities that elements do not:

- Attributes can limit their legal values to a small set of strings, as in the following example.
  ```xml
  <AttributeType name="priority" dt:type="enumeration" dt:values=""/>
  ```

- Attributes indicate a value to be inferred if the attribute is omitted from an element (that is, the attribute's default value), as in the following example.
  ```xml
  <AttributeType name="quantity" dt:type="int">
  <attribute type="quantity" default="1"/>
  ```

Although different element types can have attributes with the same name, these attributes are independent and unrelated.

**Specifying the Default Value of an Attribute**

To specify the default value of an attribute, use the **default** attribute. You specify this attribute on **AttributeType** and **attribute** elements in the schema.

For example, the following schema assigns the default value of **Seattle** to the
City attribute.

```xml
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="Customer">
    <AttributeType name="CustomerID" />
    <AttributeType name="ContactName" />
    <AttributeType name="City" default="Seattle" />
    <attribute type="CustomerID" />
    <attribute type="ContactName" />
    <attribute type="City" />
  </ElementType>
</Schema>
```

If you have a document instance that has a Customer element with a missing City attribute, the parser assumes the default value (Seattle) for the attribute and validates the document. For example, consider the following document instance.

```xml
<Customer CustomerID="ALFKI" ContactName="Maria Anders" City="London"
<Customer CustomerID="ANATR" ContactName="Ana Trujillo" />
```

The customer ALFKI specifies a city (London), so the default value is ignored. On the other hand, the customer ANATR has no City attribute, so it receives the default value (Seattle).

The behavior is slightly different if the schema specifies both a default attribute and a required attribute. For example, the following AttributeType specifies the City attribute as required with a default value of Seattle.

```xml
<AttributeType name="City" default="Seattle" required="yes" />
```

The Customer element is required to have a City attribute and it must have Seattle as its value.
Using XDR Schema Data Types

In an XML-Data Reduced (XDR) schema, you can specify the data types of elements and attributes. In addition to supporting the attribute types found in a document type definition (DTD), the XDR schema provides support for a range of data types.

The data types indicate the format of the data and enable processing specific to that data type in the XML Document Object Model (DOM). Also, the XML parser uses the data type information for validation of the type.

To specify data types in the XDR schema, the schema must include the data type namespace, `urn:schemas-microsoft-com:datatypes`.

Following is the minimum XDR schema with the namespace declarations.

```xml
<?xml version="1.0" ?>
<Schema name="MySchema" xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
    ...
</Schema>
```

For a list of the data types supported in XDR schemas, see XDR Schema Data Types.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Specifying Data Types

Although XML-Data Reduced (XDR) schema support in MSXML 2.0 and later versions allow data types to be specified within attributes, only the following data types are supported within attributes by MSXML and the Document Object Model (DOM):

- string
- id
- idref
- idrefs
- nmtoken
- nmtokens
- enumeration

In MSXML 2.0, the list of data types that can apply to elements did not include the attribute types. These restrictions were lifted in MSXML 2.5 (released with Microsoft Windows® 2000 and Internet Explorer 5.01) although attribute type data types other than id, idref, and idrefs on elements were not validated. With MSXML 2.6 and later, attributes and elements have equivalent functionality.

Example A — Specifying the Data Type in the Schema

The following schema examples specify data types, such as string, nmtoken, int, float, Boolean, and so on, when declaring an element or attribute.

In this example, the dt:type attribute declares the data type of the att1 attribute.

```xml
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
     xmlns:dt="urn:schemas-microsoft-com:datatypes" >
   <AttributeType name="att1" dt:type="int"/>
   <ElementType name="Element1">
     <element type="Element1"/>
     <attribute type="att1"/>
   </ElementType>
</Schema>
<?xml version="1.0"?>
```
Alternatively, you can specify the data type using the `<datatype>` element as in this example.

```
<AttributeType name="att1">
  <datatype dt:type="int"/>
</AttributeType>
```

Example B — Specifying id/idref/idrefs Types in the Schema

The following example illustrates how the `id`, `idref`, and `idrefs` attribute data types of a document type definition (DTD) are specified in XDR schema. The schema describes the `Employee` and `Order` elements.

Because an employee takes orders, the `OrderList` attribute of the `Employee` element is declared as `idrefs` type, referring to the `OrderID` attribute of the `Order` element. Conversely, because an order is taken by an employee, the `EmployeeID` attribute of the `Order` element is declared as `idref` type, referring to the `EmployeeID` attribute of the `Employee` element.

```xml
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Order">
    <AttributeType name="OrderID" dt:type="id"/>
    <AttributeType name="OrderDate"/>
    <AttributeType name="EmployeeID" dt:type="idref"/>
    ...<attribute type="OrderID"/>
    ...<attribute type="EmployeeID"/>
  </ElementType>
  <ElementType name="Employee">
    <AttributeType name="EmployeeID" dt:type="id"/>
    <AttributeType name="LastName"/>
    ...<attribute type="EmployeeID"/>
    ...<attribute type="LastName"/>
    <AttributeType name="OrderList" dt:type="idrefs"/>
    ...<attribute type="OrderList"/>
  </ElementType>
</Schema>
```

The following is a roughly equivalent DTD.
An instance of the XML fragment is shown in the following example. The XML fragment shows a list of order IDs as a value of the `OrderList` attribute in the `Employee` element. If any of these values do not exist as the value of the `OrderID` attribute of the `Order` element, an error will be raised.

In a similar way, the `EmployeeID` attribute of the `Order` element is an `idref` attribute, referring to the `EmployeeID` attribute of the `Employee` element, thus forming an intra-document link. The `id` and `idref` relationships are similar to the primary key and foreign key relationships in relational databases.

```
<Order OrderID="O1" OrderDate="1996-07-17T00:00:00" EmployeeID="Emp1"/>
<Order OrderID="O2" OrderDate="1996-08-01T00:00:00" EmployeeID="Emp1"/>
<Order OrderID="O3" OrderDate="1996-08-07T00:00:00" EmployeeID="Emp1"/>
<Order OrderID="O4" OrderDate="1996-08-20T00:00:00" EmployeeID="Emp2"/>
<Order OrderID="O5" OrderDate="1996-08-20T00:00:00" EmployeeID="Emp2"/>
...<Employee EmployeeID="Emp1"
  LastName="Davolio"
  OrderList="O1 O2 O3">
</Employee>
<Employee EmployeeID="Emp2"
  LastName="Fuller"
  OrderList="O4 O5">
</Employee>
...```

**Example C — Specifying the Data Type in an XML Document Instance**

You can specify the data type of an element without requiring the use of a schema. To do this, you can use the `dt:dt` attribute or declare the data type as an element.
The following document instance shows how to specify the data type using the `dt:dt` attribute. In this example, the `dt:dt` attribute specifies the `int` data type for the `Pages` element.

```xml
<BookInfo xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <Title>XML for Fertilizer</Title>
  <Pages dt:dt="int">123</Pages>
</BookInfo>
```

Now, consider the following document instance. It uses the `dt:dt` attribute as in the preceding example. In this case, the attribute specifies the data type of the `size` element.

```xml
<shoes xmlns:dt="urn:schemas-microsoft-com:datatypes" id="f1121" sizes="mens">
  <sizes id="mens">
    <size dt:dt="int">8</size>
    <size dt:dt="int">10</size>
    <size dt:dt="int">12</size>
  </sizes>
</shoes>
```

However, you can rewrite the preceding instance so that you declare the data types as elements, as shown in the following example.

```xml
<shoes xmlns:dt="urn:schemas-microsoft-com:datatypes" id="f1121" sizes="mens">
  <sizes id="mens">
    <size dt:dt="int">8</size>
    <size dt:dt="int">10</size>
    <size dt:dt="int">12</size>
  </sizes>
</shoes>
```

Following are some guidelines for working with data type elements:

- The `dataType` property on data type elements returns the data type of the element. To return the typed value of the element, use the `nodeTypedValue` property. Using the `nodeTypedValue` property on an element that has no data type returns the string value of the element.
- The parser validates the contents of data type elements against the type of the element. If the parser does not support a data type, it disregards that data type and no error occurs.
- A data type element cannot have child elements.
Example D — Using Enumerations

In the following example, the schema specifies enumeration as the data type for an attribute.

```xml
<AttributeType name = "Attr1"
    dt:type="enumeration" dt:values="a b c"/>
```

In MSXML 2.6 and later, elements also support the enumeration data type.

```xml
<ElementType name = "Ele1"
    dt:type="enumeration" dt:values="a b c"/>
```

The following element is a valid instance for the preceding type declaration.

```xml
<Ele1>b</Ele1>
```

You can also specify the enumeration data type on an instance of the element. For example, the following element, Ele1, contains a type declaration and declares its values.

```xml
<Ele1 dt:dt="enumeration" dt:values="a b c"
     xmlns:dt="urn:schemas-microsoft-com:datatypes">b</Ele1>
```

See Also

dataType Property | nodeTypedValue Property
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Extensibility

Unlike a document type definition (DTD), XML-Data Reduced (XDR) schemas are extensible. That is, they are built on an open content model. Schema authors are free to add their own elements and attributes to XDR schema documents. For example, you can add additional constraints to a declaration for a pages element. The following sample code declares the pages element and assigns it an int data type. Extended tags from the myExt namespace are used to augment this information with an added limitation that books must have a minimum of 50 pages and a maximum of 100 pages.

```xml
<ElementType name="pages" xmlns:myExt="urn:myschema-extensions">
  <datatype dt:type="int"/>
  <myExt:min>50</myExt:min>
  <myExt:max>100</myExt:max>
</ElementType>
```

Although validation will only check that the value of a particular pages element is an integer, your application can use the information provided by the added elements in the myExt namespace to perform additional validation.

Use of a namespace is a requirement for adding elements and attributes to a schema. For more information about namespaces, see Using Namespaces in Documents.
Referencing Other Schemas

Custom schemas can be built out of standard parts. For example, suppose that a definition for an Address element from another schema has met your needs. With namespaces, this schema can be used directly with a reference to the original schema rather than copying the definition.

To reference an element defined in another schema, you must first use a namespace declaration to introduce the other schema. This declaration consists of an attribute beginning with "xmlns:" followed by a name of your choice, called the prefix, and a value that is the Uniform Resource Identifier (URI) of the other schema.

```xml
<ElementType name="purchaseOrder"
    xmlns:abcde="http://contoso.com/stuff.xml">
    <shipTo/>
</ElementType>
```

The schema has a namespace URI of "http://contoso.com/stuff.xml" and a prefix of "abcde". The namespace URI identifies the other schema precisely; the prefix is a local name that you now use to refer to that schema and to the elements it defines as though they were defined in your schema. If the other schema has a suitable shipTo element defined, you can use it by writing the following.

```xml
<ElementType name="shipTo"
    xmlns:abcde="http://contoso.com/stuff.xml">
    <element type="abcde:Address"/>
</ElementType>
```

In the type attribute of the first child element declaration for the value "abcde:Address", the prefix "abcde" is matched to the enclosing namespace declaration; thus, "abcde:Address" has the effect of meaning "the Address element" as defined in the schema "http://contoso.com/stuff.xml".

However, this has an effect on the XML document as well. For example, in the following purchase order document you must also reference the other schema namespace.

```xml
<PurchaseOrder xmlns:abcde="http://contoso.com/stuff.xml">
    <shipTo>
        <abcde:Address>
```
You can determine that the Address element is defined by the schema at http://contoso.com/stuff.xml. When you build specialized schemas from more widely understood standard parts, like the preceding Address element, those portions can often be processed by standard software components. For example, if you have an Extensible Stylesheet Language (XSL) template available for the Address element, as defined in the schema http://contoso.com/stuff.xml, you can reuse this general purpose address styling code anywhere that Address appears.

The same prefix is used in both the schema and in the document instance: in the schema for all purchase orders, and in a specific purchase order. However this is not necessary, because the prefix is only a local alias for the namespace URI, and you can use any prefix in the document instance. You are only required to use the same prefix in a document when you declare the namespace and when you use elements from that namespace, as in the following example.
The declaration of a namespace is scoped to the element on which it appears. It makes the namespace prefix available, and associates it with the namespace URI for that element and anything it contains, unless overridden by another, enclosed namespace declaration. Therefore, because the namespace is only used on the Address element, the namespace declaration can appear on the Address element instead, as follows.

```xml
<PurchaseOrder>
  <shipTo>
    <z:Address xmlns:z="http://contoso.com/stuff.xml">
      <z:name>Alice Smith</z:name>
      <z:street>123 Maple Street</z:street>
      <z:city>Mill Valley</z:city>
      <z:state>CA</z:state>
      <z:zip>90952</z:zip>
    </z:Address>
  </shipTo>
</PurchaseOrder>
```
Omitting Prefixes

You can also omit a prefix to a namespace, which can make for more legible documents. In the following example, by omitting the prefix from the namespace declaration, you indicate that any child element without a prefix is defined by the specified namespace.

```xml
<PurchaseOrder>
  <shipTo>
    <Address xmlns="http://contoso.com/stuff.xml">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
    </Address>
  </shipTo>
</PurchaseOrder>
```

Namespace prefixes are only for intra-document bookkeeping. The corresponding namespace URI is what is important, so all four of the `Address` elements in the preceding examples mean exactly the same thing.

Although the `Address` element has been identified as being defined by the schema `http://contoso.com/stuff.xml`, `PurchaseOrder` has not been similarly tied to a schema. To do this, assuming a schema named `http://fabrikam.com/PO.xml` defines a `PurchaseOrder`, you can write the following.

```xml
<PurchaseOrder xmlns="http://fabrikam.com/PO.xml">
  <shipTo>
    <Address xmlns="http://contoso.com/stuff.xml">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
    </Address>
  </shipTo>
  <orderDate>1999-05-20</orderDate>
  <shipDate>1999-05-25</shipDate>
  <comments>
    Get these things to me in a hurry, my lawn is going wild!
  </comments>
</PurchaseOrder>
```
<comments>
<Items>
<Item>
  <productName>Lawnmower, model BUZZ-1</productName>
  <quantity>1</quantity>
  <price>148.95</price>
</Item>
<Item>
  <productName>Baby Monitor, model SNOOZE-2</productName>
  <quantity>1</quantity>
  <price>39.98</price>
</Item>
</Items>
</PurchaseOrder>

This declares that the PurchaseOrder element and all elements it contains are defined within the schema http://fabrikam.com/PO.xml, except for Address and the elements it contains, which are defined within xmlns="http://contoso.com/stuff.xml. In both cases, explicit prefixes are omitted.
Multiple Namespaces

You can declare multiple namespaces on a single element. You can also use several prefixes to refer to the same namespace. The following example demonstrates this can be done and shows that this document is functionally equivalent to the previous one.

```xml
<PurchaseOrder xmlns:a="http://fabrikam.com/PO.xml"
    xmlns:b="http://contoso.com/stuff.xml"
    xmlns:c="http://contoso.com/stuff.xml">
    <a:shipTo>
        <b:Address>
            <c:name>Alice Smith</c:name>
            <b:street>123 Maple Street</b:street>
            <c:city>Mill Valley</c:city>
            <b:state>CA</b:state>
            <c:zip>90952</c:zip>
        </b:Address>
    </a:shipTo>
    <a:orderDate>1999-05-20</a:orderDate>
    <a:shipDate>1999-05-25</a:shipDate>
    <a:comments>
        Get these things to me in a hurry, my lawn is going wild!
    </a:comments>
    <a:Items>
        <a:Item>
            <a:productName>Lawnmower, model BUZZ-1</a:productName>
            <a:quantity>1</a:quantity>
            <a:price>148.95</a:price>
        </a:Item>
        <a:Item>
            <a:productName>Baby Monitor</a:productName>
            <a:quantity>1</a:quantity>
            <a:price>39.98</a:price>
        </a:Item>
    </a:Items>
</a:PurchaseOrder>
```
Open Content Model

If a document schema has elements with an open content model, you can include elements not mentioned in the original schema, provided they are assigned to a specific namespace, either with a default namespace in scope or with a prefix. You can also include undefined attributes in an element, provided the attribute is qualified by a prefix and the prefix is in scope. For example, suppose that a third schema defines an element called **trackingInformation**. Because **PurchaseOrder** is defined as having an open content model, you can simply add this element directly to the instance.

```xml
<PurchaseOrder xmlns="http://fabrikam.com/PO.xml">
  <shipTo>
    <Address xmlns="http://contoso.com/stuff.xml">
      <name>Denise Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
    </Address>
  </shipTo>
  <orderDate>1999-05-20</orderDate>
  <shipDate>1999-05-25</shipDate>
  <comments>Get these things to me in a hurry. My lawn is going wild!</comments>
  <Items>
    <Item>
      <productName>Lawnmower, model BUZZ-1</productName>
      <quantity>1</quantity>
      <price>148.95</price>
    </Item>
    <Item>
      <productName>Baby Monitor, model SNOOZE-2</productName>
      <quantity>1</quantity>
      <price>39.98</price>
    </Item>
  </Items>
  <trackingInformation xmlns="http://example.microsoft.com/tracking.
    <route>1234-5678</route>
    <signature>32143912850-3959159=1395</signature>
  </trackingInformation>
</PurchaseOrder>
```
Because namespace qualification is required for any undefined attribute or element, the `trackingInformation` element would not have been valid if its `xmlns` attribute had been omitted.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XDR Schemas and the DOM

When you use XML-Data Reduced (XDR) schema on an instance document, the schema definition of the elements and attributes in the instance document can be accessed using the definition property of the Document Object Model (DOM). The definition property of the IXMLDOMNode returns the type declaration that corresponds to the node just as it does in an external schema. The definition property does not work with inline schemas. The definition property returns the element in the schema that declared the type for the current element or attribute. For an element, definition will return an ElementType element; for an attribute, it will return an AttributeType element. Although the schema's DOM is read-only and cannot be modified, it can be traversed, using all the functionality normally exposed through the DOM.

Consider the following example schema (Bookschema.xml) and XML documents.

Bookschema.xml

<?xml version="1.0"?><Schema xmlns="urn:schemas-microsoft-com:xml-data"><ElementType name="title"/><ElementType name="author"/><ElementType name="pages"/><AttributeType name="price"/><ElementType name="book" model="closed"><element type="title"/><element type="author"/><element type="pages"/><attribute type="price"/></ElementType></Schema>

Book.xml

Return the root element, the `a:book` node.

```javascript
var bookNode = doc.documentElement;
```

Use the `definition` property on `bookNode` to get the `ElementType` definition for the `a:book` element.

```javascript
var bookDef = bookNode.definition;
```

The following code returns the following element.

```xml
<ElementType xmlns="urn:schemas-microsoft.com:xml-data" name="book"
  <element type="title"/>
  <element type="author"/>
  <element type="pages"/>
</ElementType>
```

Using `bookDef`, you can then access other schema information.

The following code selects element (0) in the `childNodes` element.

```javascript
var title = bookDef.childNodes(0);
```

The following code returns the definition for the `title` element.

```xml
<element xmlns="urn:schemas-microsoft.com:xmldata" type="title">
```

The following code selects element (1) in the `childNodes` element.

```javascript
var author = bookDef.childNodes(1).xml;
```

The following code returns the definition for the `author` element.

```xml
<element xmlns="urn:schemas-microsoft.com:xmldata" type="author">
```

The `validate` method of the `DOMDocument` attempts to validate the XML document to the schema or schemas specified in the document.

The `validateOnParse` property of the `DOMDocument` controls the use of schemas for validation. Setting `validateOnParse` to `False` does not stop schemas from being loaded and processed. Doing this does prevent document validation, but the default attributes, `ids` and `idrefs`, and data types are all processed according to the schema.
Multiple schemas can appear in the same instance document as long as they conform to the ordering rules and their IDs do not conflict.

The `resolveExternals` property of the `DOMDocument` controls whether the parser resolves the document to an external schema. Setting `resolveExternals` to `False` will stop the parsing of "x-schema:" namespaces.

The `SchemaCache` is an object that allows the programmatic loading of schemas and association of those schemas with a namespace URI. When a document sees a namespace URI and looks for a schema, the first place it looks is the `SchemaCache` associated with the document. Therefore, a `SchemaCache` can be used to override x-schema external references as well as x-schema:## internal references.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XDR Schemas and Namespaces

Microsoft XML Core Services (MSXML) version 2.0 or later enables XML-Data Reduced (XDR) schemas to be referenced directly from a namespace declaration. This allows the MSXML parser to validate the XML document against the rules or grammar defined in the schema.

It also allows the object model to expose data types and to allow navigation to the schema document through the definition property. For more information, see XDR Schemas and the DOM.

A namespace declaration can reference a schema by including the URL to the schema document prefaced by the "x-schema:" string. For example, to reference an XDR schema for books from an XML file conforming to this schema, write the following.

```xml
  <title>Presenting XML</title>
  <author>Richard Light</author>
  <pages>334</pages>
</book>
```

Here the default namespace refers to the sample book grammar used in Defining Elements and Attributes. The namespace Uniform Resource Identifier (URI) is prefixed with "x-schema:" to indicate to MSXML that there is a schema corresponding to this namespace. The remainder of the URI is the schema's location. The path or URI to the schema file must be valid, or a parse error will occur. A schema can also be created as part of the XML document.

See Also

namespaces Property | schemas Property | Using Inline XDR Schemas
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Using Inline XDR Schemas

Instead of using an external schema file, MSXML version 3.0 or later allows you to use inline schemas. These schemas are useful when it is inconvenient to physically separate the schema and the instance (XML document). For example, if you generate XML-Data Reduced (XDR) schema from data retrieved from a database, it may be more convenient to put the schema and the data in the same file.

Note Inline schemas are not supported in MSXML 2.5 and earlier versions. To avoid an error on the root element, use the Xmlinst.exe Replace Mode Tool.

Example

The following XML document contains an inline schema against which the Student element is validated. The name of the schema, mySchema, is indicated through the name attribute on the Schema element.

```
<Root>
<Schema name="mySchema" xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">
   <ElementType name="age" dt:type="int"/>
   <ElementType name="Student" model="closed" content="eltOnly">
      <element type="age"/>
   </ElementType>
</Schema>

Data file

<a:Student xmlns:a="x-schema:#mySchema">
   <a:age>10</a:age>
</a:Student>
</Root>
```

An element is associated with an inline schema just as it is with an external schema, through the "x-schema:" syntax when defining that element's namespace identifier. For an external schema, the URI or URL that follows "x-schema:" indicates the location of the external schema file. With an inline
schema, that which follows "x-schema:" indicates the name of the schema within the XML instance document (indicated by the # sign followed by the name).

The "#" syntax following "x-schema:" refers to a schema within the local document only. This syntax cannot be used to reference a schema within an external document. For example, "x-schema:URI#name" is not allowed.

The inline schema associated with an element must precede that element within the document. When validating against a schema, if the name indicated through an element's namespace declaration does not correspond to an already parsed schema, an error will occur. An error will also occur when an element is associated with a schema that does not exist.
## Inline Schemas Persisted from ADO

You can use MSXML 3.0 to parse inline schemas produced by Microsoft ActiveX® Data Objects (ADO), but they will not be validated.

### Example

```xml
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
     xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
     xmlns:rs='urn:schemas-microsoft-com:rowset'
     xmlns:z='#RowsetSchema'>
  <s:Schema id='RowsetSchema'>
    <s:ElementType name='row'>
      <s:attribute type='CustomerID'/>
      <s:extends type='rs:rowbase'/>
    </s:ElementType>
    <s:AttributeType name='CustomerID' rs:number='1'
                      rs:nullable='true' rs:write='true'>
      <s:datatype dt:type='string' dt:maxLength='5'/>
    </s:AttributeType>
  </s:Schema>
  <rs:data>
    <z:row CustomerID='...'/>
  </rs:data>
</xml>
```

MSXML 3.0 essentially ignores the following elements and attributes, and the remainder of the schema is parsed as if it were a valid schema.

<table>
<thead>
<tr>
<th>Elements</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>extends</td>
<td>dt:max</td>
</tr>
<tr>
<td></td>
<td>dt:maxExclusive</td>
</tr>
<tr>
<td></td>
<td>dt:maxLength</td>
</tr>
<tr>
<td></td>
<td>dt:min</td>
</tr>
<tr>
<td></td>
<td>dt:minExclusive</td>
</tr>
</tbody>
</table>

Validation against the remainder of the schema cannot occur because the schema does not contain a **name** attribute and because the reference to the schema is not prefaced by "x-schema:". MSXML 3.0 views that reference as a Uniform Resource Name (URN) with no special significance.
See Also

definition Property | validateOnParse Property
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Representing the XDR Schema as a DTD

The following is an example of representing the XML-Data Reduced (XDR) schema with a document type definition (DTD). However, the representation is not exact.

- Schemas and DTDs use different content models. When the model is defined as open, the element can include `ElementType` elements, `AttributeType` elements, and mixed content not specified in the content model. When the model is defined as closed, the element cannot include elements or mixed content not specified in the content model. The DTD uses a closed content model; schemas use an open content model.
- Possible data types for elements and attributes are different. In the following code sample, the data types parameter, `entity`, lists the data types for both the attributes and elements of the `dt:type` attribute for the `datatype` element. In MSXML 2.0 and later, support for these data types is limited. The attributes only support the first 10 types (which map to DTD attribute types) and elements only support from "string" through "uuid"; therefore an element cannot behave like an `id` or `idref`.
- Placement of `xmlns` namespace declarations cannot be modeled very well using a DTD. In a real schema, the `xmlns:dt` attributes can go anywhere. Any other `xmlns` declaration can also appear because schemas follow an open content model. This way you can add extended information in your schemas for your own use.
- The `maxOccurs` attribute can only have the values "1" or "*" and this cannot be modeled using a DTD enumeration because "*" is not a valid name token.
- `ElementType` and `AttributeType` elements can only have one `datatype` child element. DTDs cannot model this sort of ordinality, particularly when other child elements are allowed.

<!--
The possible element datatypes according to
-->
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schema
XDR Schema Reference

Microsoft® XML Core Services (MSXML) versions 2.0 and later include support for XML-Data Reduced (XDR) schemas.

XDR schema is based on the World Wide Web Consortium (W3C) XML-Data Note. XDR schema is implemented as a subset of the XML-Data submission that corresponds to the feature set proposed for the Document Content Description (DCD) for XML.

Microsoft is actively involved in defining the emerging W3C XML Schema standard, XML Schema definition language (XSD). MSXML and several other Microsoft products, such as Microsoft BizTalk™ Server, Microsoft Office 2000, and Microsoft SQL Server™ 2000, continue to support XDR schema.

Unlike the unique document type definition (DTD) syntax, XDR schema allows you to define the rules governing the relations between elements and attributes using a standard XML instance syntax, which can be parsed and managed using XML applications. XDR schema also adds support for namespaces, data types, and features like range constraints.

The definitions in an XDR schema are contained within the top-level Schema element. The Schema element definition must come from the namespace xmlns="urn:schemas-microsoft-com:xml-data". The namespace declaration is an XML processing instruction as defined in the W3C Namespace in XML. It is convenient to make the Schema namespace the default namespace to avoid having to prefix all definitions within a schema. This documentation uses the convention in which the Schema namespace is the default namespace and omits a prefix from all XDR schema element and attribute names.

The Schema element in an XDR schema document should also contain namespace declarations for any other schemas, such as the namespace that defines the built-in data types for XDR schema.

```xml
<Schema name="myschema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatyp
</Schema>
```
The **Schema** element can contain any number of top-level declarations. The principal XDR schema elements include **ElementType**, which declares element types, and **AttributeType**, which declares attribute types. These elements must have an explicit **name** attribute that is unique within the schema. The XDR schema **element** and **attribute** elements can appear within the **ElementType** element to constrain the content and attributes that can appear in elements of the named type. The **ElementType** attribute **order**, and the **group** element, allow you to further establish finely-grained definitions. The **minOccurs** and **maxOccurs** attributes allow you to define the number of occurrences of an element or group.

The content for an element of a given element type can be defined as one of four categories: empty (no content), text only, subelements only, or a mixture of text and subelements. This choice is expressed in the **ElementType** attribute named **content**, which can take the values **empty**, **textOnly**, **eltOnly**, and **mixed**, respectively.

Element types and attribute types can constrain their values and contents to be instances of a particular data type, referenced from the **datatype element** namespace. To use the **datatype** namespace in a schema, it must be declared within the **Schema** element. The **dt:type** attribute can then be used to reference data types; in Microsoft Internet Explorer version 5.0 and later, element and attribute types can take rich data types. For more information about data types, see the **XML Data Types Reference**.

**See Also**

[ElementType](#) | [AttributeType](#) | [element](#) | [attribute](#) | [datatype](#) | [XDR Schema Data Types Reference](#) | [XDR Schema Developer's Guide](#) | [XML Reference](#) | [Schema](#)

**Other Resources**

[XML-Data Note](#) | [Document Content Description for XML](#) | [Namespaces in XML Recommendation](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
# XDR Schema Elements

The following table describes the XML-Data Reduced (XDR) schema elements.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>Refers to a declared attribute type that can appear within the scope of the named <strong>ElementType</strong> element.</td>
</tr>
<tr>
<td>AttributeType</td>
<td>Defines an attribute type for use within the XDR schema <strong>Schema</strong> element.</td>
</tr>
<tr>
<td>datatype</td>
<td>Specifies the data type for the <strong>ElementType</strong> or <strong>AttributeType</strong> element.</td>
</tr>
<tr>
<td>description</td>
<td>Provides documentation about an <strong>ElementType</strong> or <strong>AttributeType</strong> element.</td>
</tr>
<tr>
<td>element</td>
<td>Refers to a declared element type that can appear within the scope of the named <strong>ElementType</strong> element.</td>
</tr>
<tr>
<td>ElementType</td>
<td>Defines an element type for use within the XDR schema <strong>Schema</strong> element.</td>
</tr>
<tr>
<td>group</td>
<td>Organizes content into a group to specify a sequence.</td>
</tr>
<tr>
<td>Schema</td>
<td>Identifies the start of a schema definition.</td>
</tr>
</tbody>
</table>
XDR Schema Data Types Support

MSXML 2.0 and later provides support for XDR schema data types. For more information, see the XDR Schema Data Types Reference.

See Also

XDR Schema Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
attribute

The XML-Data Reduced (XDR) schema element that refers to a declared attribute type that can appear within the scope of the named ElementType element.

```xml
<attribute
default="default-value"
type="attribute-type"
[required="{yes | no}"]>
```

Attributes

default
The default value for the attribute. This specified default takes precedence over any default provided on the AttributeType element it refers to.

type
The name of an AttributeType element defined in this schema (or another schema indicated by the specified namespace). The supplied value must correspond to the name attribute on the AttributeType element. The type can include a namespace prefix.

required
An indicator of whether the attribute must be present on the element. Optional if the required attribute is present on the referenced AttributeType.

Element Information

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Number of attribute types specified in the schema.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ElementType</td>
</tr>
<tr>
<td>Child elements</td>
<td>(No child elements.)</td>
</tr>
</tbody>
</table>

Remarks

When the required attribute is set to yes and the default attribute specifies a
default value, the supplied default value must always be the value; documents containing other attribute values are invalid.

When the **required** attribute is set to **yes** and no default is specified, each element whose type is declared to have the attribute must supply its value.

**See Also**

[AttributeType](#) | [XDR Schema Elements](#) | [XDR Schema Data Types Reference](#)
AttributeType

Defines an attribute type for use within the XML-Data Reduced (XDR) schema Schema element.

```xml
<AttributeType
default="default-value"
dt:type="primitive-type"
dt:values="enumerated-values"
name="idref"
required="{yes | no}"/>
```

Attributes

**default**

The default value for the attribute. The default value must be legal for that attribute instance. For example, when the attribute is an enumerated type, the default value must appear in the values list.

**dt:type**

The data type for this attribute type. In the implementation provided with Microsoft® Internet Explorer 5.0, an attribute can take one of the following types: entity, entities, enumeration, id, idref, idrefs, nmtoken, nmtokens, notation, or string. When the type enumeration is selected, the dt:values attribute should also be supplied, listing the allowed values. For more information about data types, see the XDR Schema Data Types Reference.

**dt:values**

A list of the possible values when dt:type is set to enumeration.

**name**

The name of the attribute type. This attribute is required. References to this attribute type within an ElementType definition are made in the schema with the attribute element. The name supplied here corresponds to the type attribute of the attribute element.

**required**

An indicator of whether the attribute must be present on the element.

Element Information
Number of occurrences Unlimited.

Parent elements ElementType, Schema
Child elements Datatype, description

Element Properties

<table>
<thead>
<tr>
<th>TAG</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name=&quot;idref&quot;</td>
<td><strong>ElementType</strong> and <strong>AttributeType</strong> elements must have names.</td>
</tr>
<tr>
<td>required='yes'</td>
<td>This attribute must occur. If a default value is also given, the value must match the default.</td>
</tr>
<tr>
<td>dt:values</td>
<td>Specifies the values for enumeration types.</td>
</tr>
<tr>
<td>default</td>
<td>Specification of the default value for the attribute. This can apply to any type of attribute. The default value must be legal for that attribute instance. (For example, in <strong>enumeration</strong> the default value must appear in the values list.)</td>
</tr>
</tbody>
</table>

Remarks

An **AttributeType** declaration contained within an **ElementType** element defines that attribute type within the scope of the **ElementType** that contains it.

In XDR schema, as in XML 1.0, the order of appearance of attributes in an element is not constrained and a given attribute can appear no more than once in an element.

Example

The following example shows an enumeration and its document type definition (DTD) equivalent.

```xml
<AttributeType name="colors" dt:type="enumeration"
    dt:values="red green blue">
<!ATTLIST x colors (red | green | blue)>
```

Example
The following example declares an attribute type called `myattr`. This declaration constrains `myattr` attributes to contain only a name token.

```xml
<AttributeType name="myattr" dt:type="nmtoken"/>
```

See Also

[XDR Schema Elements](#) | [XDR Schema Data Types Reference](#) | attribute
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
**datatype**

Specifies the data type for the **ElementType** or **AttributeType** element.

```
<datatype
dt:type="datatype">
```

**Attributes**

**dt:type**
The data type.

```
<ElementType name="colors">
  <datatype dt:type="char"/>
</ElementType>
```

**Element Information**

**Number of occurrences** One time.

**Parent elements**  **AttributeType**, **ElementType**

**Child elements**  (No child elements.)

**Remarks**

The **datatype** functionality is an extension offered by XML-Data Reduced (XDR) schema that cannot be specified using the document type definition (DTD) syntax.

**See Also**

[XDR Schema Elements](#) | [XDR Schema Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
description

Provides documentation about an ElementType or AttributeType element.

<description>

Element Information

Number of occurrences  One time.
Parent elements  AttributeType, ElementType, Schema, group, element, attribute
Child elements  Any well-formed XML content.

Remarks

Information supplied within the description element is not used in validation, but provides a standardized way to supply information that can be retrieved by tools.

Example

The following example shows use of the description element providing information about the element.

(ElementType name="Element")
<AttributeType name="att"/>

(ElementType name="Elements" model="closed">
  <element type="Element" minOccurs="1" maxOccurs="1">
    <description>element description</description>
  </element>

    <attribute type="att">
      <description>attribute description</description>
    </attribute>

</ElementType>

See Also
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
element

The XML-Data Reduced (XDR) schema element that refers to a declared element type that can appear within the scope of the named ElementType element.

<element
type="element-type"
[ minOccurs="{0 | 1}" ]
[ maxOccurs="{1 | *}" ]
>

Attributes

type
The name of an ElementType element defined in this schema (or another schema indicated by the specified namespace). The supplied value must correspond to the name attribute on the ElementType element. The type can include a namespace prefix.

minOccurs
Optional. The minimum number of times the reference element type can occur on the element. The following values can be assigned to this attribute.

0 When the minimum value is zero, the specified element is not required; the element is optional.
1 The specified element must occur at least once.

maxOccurs
Optional. The maximum number of times the element can occur on the element. The following values can be assigned to this attribute.

1 Can occur a maximum of one time.
* An unlimited number of occurrences is allowed.

Element Information
**Number of occurrences**  The number of element types defined in the schema.

**Parent elements**  ElementType, group

**Child elements**  (No child elements.)

**Remarks**

**ElementType** declarations can constrain the content and attributes that appear in elements of the named type by referring to other element type or attribute type declarations.

The **minOccurs** and **maxOccurs** attributes have the default value 1. An element with neither attribute specified uses these default values and therefore must appear once and only once in a content model.

**See Also**

[XDR Schema Elements]  [XDR Schema Data Types Reference]
**ElementType**

Defines an element type for use within the XML-Data Reduced (XDR) schema Schema element.

```xml
<ElementType 
    content="{empty | textOnly | eltOnly | mixed}" 
    dt:type="datatype" 
    model="{open | closed}" 
    name="idref" 
    order="{one | seq | many}">

Attributes

**content**

An indicator of whether the content must be empty or can contain text, elements, or both. The following values can be assigned to this attribute. The default value for the content attribute is mixed. When the content attribute is set to mixed, the order attribute requires a many value.

- **empty** The element cannot contain content.
- **textOnly** The element can contain only text, not elements.
- **eltOnly** The element can contain only the specified elements. It cannot contain any free text.
- **mixed** The element can contain a mix of named elements and text. The default value is mixed. If the value of the content attribute is mixed, the minOccurs and maxOccurs attributes will not trigger a validation error if the number of child elements is outside the specified boundary.

**Note** Elements that have content="empty" or "textOnly" can not have model="open".

**Example**

The following example shows use of the content attribute with different values.
Example

The following example shows the content attribute for an equivalent document type definition (DTD).

empty:  <!ELEMENT x EMPTY>
textOnly:  <!ELEMENT x (#PCDATA)>
eltOnly:  <!ELEMENT x y>
mixed:  <!ELEMENT x (#PCDATA | q | r)*>

dt:type
The data type of the element. The valid data types are defined in the XDR Schema Data Types Reference.

model
An indicator of whether the content can include only what is defined in the content model. The default is open.

<ElementType name="x" model="open"/>

When the model is defined as open, the element can include additional elements or attributes not declared explicitly in the content model. These additional tags can come from the same or a different namespace. If they are in the same namespace, there must be a corresponding ElementType or AttributeType definition for them in the schema.

When the model is defined as closed, the element cannot include elements and mixed content not specified in the content model. The DTD uses a closed model.

name
The name of the element. This attribute is required. If this element type is declared as a valid child of another element type, this name is specified
within an **element** element.

```
<ElementType name="x">
</ElementType>
```

The DTD equivalent is as follows.

```
<!ELEMENT x EMPTY>
```

### order

An indicator of how the elements are to appear. This indicator can have the following values.

- **one**
  
  Permits only one of a set of elements. For a document to correctly validate when the **one** attribute is specified, the **model** attribute for the **ElementType** must be specified as **closed**.

- **seq**
  
  Requires the elements to appear in the specified sequence.

- **many**
  
  Permits the elements to appear (or not appear) in any order. If you specify **many** for the **order** attribute, **maxOccurs** values are no longer relevant during validation.

The **seq** value is required to specify valid sequences. For example, it can be used to specify when a particular sequence, such as x1, y1 or x2, y2, is valid but no other possible combinations are valid. The **seq** value serves the same role as parentheses in a DTD. The default value is **seq**.

The following example demonstrates the **one** setting. The value of the **model** attribute must be **closed** when **one** is specified for the **order** attribute.

```
<ElementType name="z" content="eltOnly" order="one" model="closed">
  <element type="x" minOccurs="1" maxOccurs="1"/>
  <element type="y" minOccurs="1" maxOccurs="1"/>
</ElementType>
```

The following example demonstrates the **seq** setting.

```
<ElementType name="x" content="eltOnly" order="one" model="closed">
  <group order="seq" content="eltOnly" model="closed">
    <element type="x1"/>
    <element type="y1"/>
  </group>
</ElementType>
```
The following example demonstrates the **many** setting.

```xml
<ElementType name="x" content="eltOnly" order="many" model="closed">
  <element type="q"/>
  <element type="r"/>
</ElementType>
```

The **many** setting of the **order** attribute makes the **minOccurs** and **maxOccurs** attributes irrelevant during validation. Rather than using the **many** setting for the **order** attribute in conjunction with **minOccurs** and **maxOccurs**, use the **seq** setting instead. For example, consider the following schema, which sets **order**="seq".

```xml
<?xml version="1.0"?>
  <ElementType name="PublisherID" model="closed" dt:type="ID" content="textOnly"/>
  <ElementType name="PublisherName" model="closed" dt:type="string" content="textOnly"/>
  <ElementType name="Publisher" model="closed" content="eltOnly" order="many">
    <element type="PublisherID"/>
    <element type="PublisherName"/>
  </ElementType>
  <ElementType name="Book" model="closed" content="eltOnly" order="seq">
    <element type="Publisher" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

If the following XML document is validated against the preceding schema, the validation fails because the number of **Publisher** elements exceeds the specified **maxOccurs** value of 1. However, if the **order** value is set to **many**, the
validation will incorrectly succeed because the maxOccurs="1" setting will be ignored.

```xml
<?xml version='1.0'?>
<Book xmlns="x-schema:orderschema.xml">
  <Publisher>
    <PublisherID>P1</PublisherID>
    <PublisherName>GG&G</PublisherName>
  </Publisher>
  <Publisher>
    <PublisherID>P2</PublisherID>
    <PublisherName>North</PublisherName>
  </Publisher>
</Book>
```

**Element Information**

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td><strong>Schema</strong></td>
</tr>
<tr>
<td>Child elements</td>
<td><code>attribute</code>, <code>AttributeType</code>, <code>datatype</code>, <code>description</code>, <code>element</code>, <code>group</code></td>
</tr>
</tbody>
</table>

**Element Properties**

<table>
<thead>
<tr>
<th>TAG</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name=&quot;idref&quot;</td>
<td><strong>ElementType</strong> and <strong>AttributeType</strong> elements must have names. Elements or mixed content not specified in the content model is permitted in this element. To subclass a particular definition, this setting must be open (when inheritance is implemented in the future). Not valid on an <strong>ElementType</strong> with content=&quot;textOnly&quot;.</td>
</tr>
<tr>
<td>Model=&quot;open&quot;</td>
<td>Elements or mixed content not specified in the content model is not permitted in this element.</td>
</tr>
<tr>
<td>Model=&quot;closed&quot;</td>
<td>The element may not contain anything.</td>
</tr>
<tr>
<td>content=&quot;empty&quot;</td>
<td>The element may contain only text, not elements. If</td>
</tr>
</tbody>
</table>
model="open", the element may still contain text and other unnamed elements.

The element may contain a mixture of named elements and intermingled text. If model="open", the element may still contain text and other unnamed elements. Implies order='many' and maxOccurs="*"!

* order='one'
Permits only one of a set of elements.

* order='seq'
Indicates that the elements must appear in the listed sequence. This is similar to the default for a list of elements, however this is necessary in order to group sequences, as in the example where either x1, y1 or x2, y2 is legal but where no other combinations are. Serves the same role as parentheses in a DTD.

order='many'
The elements may appear (or not appear) in any order.

dt:type=datatype
Specifies that the element should contain an integer. For a list of primitive data types, see the XDR Schema Data Types Reference.

Remarks

The term "element type" refers to the element type of which all elements sharing a name are instances. Element types are declared in schemas; elements occur in documents. Element types are declared with the ElementType element type.

Example

```
<ElementType name="x">
   <!-- element content declarations go here -->
</ElementType>
```

See Also

XDR Schema Elements | XDR Schema Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
The XML-Data Reduced (XDR) schema element that organizes content into a group to specify a sequence.

```xml
<group
    maxOccurs="{1 | *}"  
    minOccurs="{0 | 1}"  
    order="{one | seq | many}" >
```

**Attributes**

**maxOccurs**
The maximum number of times the group can occur. The following values can be assigned to this attribute.

- `1` Occurs a maximum of one time.
- `*` An unlimited number of occurrences is allowed.

**minOccurs**
The minimum number of times the group can occur. The following values can be assigned to this attribute.

- `0` Not required; the group is optional.
- `1` Must occur at least once.

**order**
A constraint on the sequence of elements (and other contained groups) within this group. The following values can be assigned to this attribute.

- `one` Permits only one instance of each element contained in the group. This corresponds to the "|" (vertical bar) symbol in the document type definition (DTD).

Allowed as part of **ElementType** and **group**.
Requires the elements in the group to appear in the specified sequence. This is the default value.

**seq**

Allowed as part of **ElementType** and **group**.

Permits the elements in the group to appear (or not appear) in any order.

**many**

Allowed as part of **ElementType** and **group**.

The **seq** setting is required to specify valid sequences. For example, it can be used to specify when a particular sequence, such as \( x_1, y_1 \) or \( x_2, y_2 \), is valid but no other possible combinations are valid. The **seq** value serves the same role as parentheses in a DTD.

### Element Information

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td><strong>ElementType</strong></td>
</tr>
<tr>
<td>Child elements</td>
<td>element, description</td>
</tr>
</tbody>
</table>

### Element Properties

<table>
<thead>
<tr>
<th>TAG</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>order='one'</td>
<td>Permits only one of a set of elements. Indicates that the elements must appear in the listed sequence. This is similar to the default for a list of elements, however this is necessary in order to group sequences, as in the example where either ( x_1, y_1 ) or ( x_2, y_2 ) is legal but where no other combinations are. Serves the same role as parentheses in a DTD.</td>
</tr>
<tr>
<td>* order='seq'</td>
<td>The elements may appear (or not appear) in any order.</td>
</tr>
<tr>
<td>order='many'</td>
<td>Allows precise definition of cardinality. '*(^t)' on maxOccurs means unlimited.</td>
</tr>
</tbody>
</table>
Remarks

The order attribute takes the same values as the order attribute of the ElementType element.

The minOccurs and maxOccurs attributes have the default value 1. A group with neither attribute must appear once and only once in a content model.

Examples

The following example demonstrates the one setting.

```xml
<ElementType name="z" order="one">
  <element type="x">
  <element type="y">
  </ElementType>
</ElementType>
```

The following represents a legal instance of the schema.

```xml
<z>
  <x/>
  <y/>
</z>
```

The following example demonstrates the seq setting.

```xml
<ElementType name="x" order="one">
  <group order="seq">
    <element type="x1">
      <element type="y1">
      </element>
    </group>
    <group order="seq">
      <element type="x2">
        <element type="y2">
        </element>
      </group>
  </group>
</ElementType>
```

The following two examples represent legal instances of this schema.

```xml
<x>
  <x1/>
  <y1/>
</x>
```
and:

```xml
<x>
  <x2/>
  <y2/>
</x>
```

The following example demonstrates the `many` setting.

```xml
<Element Type name="x" content="eltOnly" order="many">
  <element type="q">
    <element type="r"/>
  </element>
</ElementType>
```

The following five examples represent all legal instances for this schema.

```xml
<x> </x>
<x> <q> </x>
<x> <r> </x>
<x> <q> <r> </x>
<x> <r> <q> </x>
```

**See Also**

[XDR Schema Elements](#) | [XDR Schema Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Schema

The XML-Data Reduced (XDR) schema element that identifies the start of a schema definition.

```xml
<Schema
    name="schema-name"
    xmlns="namespace">
```

**Attributes**

- **name**
  - The name of the schema.
- **xmlns**
  - One or more namespaces for use with the schema.

**Element Information**

- **Number of occurrences**
  - One time.
- **Parent elements**
  - (No parent elements.)
- **Child elements**
  - `AttributeType`, `ElementType`, `description`

**Remarks**

The XDR schema `Schema` element is the document element (the top-level element) in a schema definition.

The namespaces specified using the `xmlns` attribute must include the namespace for the schema-defining elements (`Schema`, `ElementType`, and so on). Any prefix can be used, but it is convenient to make the `Schema` namespace the default namespace to avoid having to use a prefix on every XML schema element.

The XDR schema `Schema` element in an XML schema should also contain namespace declarations for any other schemas, such as the namespace that
defines the built-in data types for the XML schema.

**Example**

```xml
<Schema name="fooSchema" xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">
</Schema>
```

**See Also**

[XDR Schema Elements](#) | [XDR Schema Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
XDR Schema Data Types Reference

In addition to the string and other primitive types defined by the World Wide Web Consortium (W3C) XML 1.0 Recommendation, Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office supports a rich set of data types. It also supports conversions between the primitive types and these rich data types.

Data types are referenced from the data type namespace "urn:schemas-microsoft-com:datatypes". To use data types within an XML schema, declare the data type namespace within the schema document. This documentation assumes that the data type namespace has been assigned the dt prefix, as shown in the following example.

```xml
<Schema name="myschema"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
<!-- ... -->
</Schema>
```

See Also

XDR Schema Data Types | Primitive XDR Schema Data Types | Supported Data Type Conversions | XDR Schema Developer's Guide | XDR Schema Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
# XDR Schema Data Types

The following table shows the data types used in XML-Data Reduced (XDR) schemas.

<table>
<thead>
<tr>
<th>XDR Data Type</th>
<th>Description</th>
<th>Corresponding type in DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin.base64</td>
<td>MIME-style Base64 encoded binary BLOB (binary large object).</td>
<td>Not used</td>
</tr>
<tr>
<td>bin.hex</td>
<td>Hexadecimal-encoded binary data.</td>
<td>Not used</td>
</tr>
<tr>
<td>boolean</td>
<td>A value of either 0 or 1. Example: 0==&quot;false&quot; and 1==&quot;true&quot;</td>
<td>Not used</td>
</tr>
<tr>
<td>char</td>
<td>A number corresponding to the Unicode representation of a single character. A date in a subset of the ISO 8601 format, without the time data.</td>
<td>Not used</td>
</tr>
<tr>
<td>date</td>
<td>A date in a subset of the ISO 8601 format, without the time data. Example: &quot;1994-11-05&quot;</td>
<td>Not used</td>
</tr>
<tr>
<td>dateTime</td>
<td>A date in a subset of the ISO 8601 format, with optional time and no optional zone. Fractional seconds can be as precise as nanoseconds. Example: &quot;1988-04-07T18:39:09&quot;</td>
<td>Not used</td>
</tr>
<tr>
<td>dateTime.tz</td>
<td>A date in a subset of the ISO 8601 format, with optional time and optional zone. Fractional seconds can be as precise as nanoseconds. Example: &quot;1988-04-07T18:39:09-08:00&quot;</td>
<td>Not used</td>
</tr>
</tbody>
</table>

A reference to an unparsed entity. Requires a
entity matching <!ENTITY> declaration for each instance of the entity. Not used

entities A list of entities delimited by white space. Not used
Similar to nmtoken, but with an explicit list of allowed values (supported on attributes only).

type

 enumeration Example: "Red Blue Green"

A number with no more than 14 digits to the left of the decimal point and no more than 4 to the right. Not used

fixed.14.4 Example: 9999.0044

A real number, with no limit on the digits (2.2250738585072014E-308 - 1.7976931348623157E+308). Not used

float Example: 111, 3.14, -123.456E+10

A one-byte integer, with optional sign (-128 to 127). Not used

i1 Example: 1, 127, -128

A two-byte integer with optional sign (-32768 to 32767). Not used

i2 Example: 1, 703, -32768

A four-byte integer with optional sign (-2147483648 to 2147483647). Not used

i4 Example: 1, 703, -32768, 148343, -1000000000

An eight-byte integer with optional sign (-9223372036854775808 to
id

A value that identifies an attribute as an id type attribute. The id values must be unique throughout the document.

In a document, idref(s) attributes refer to an id type attribute, thus forming a relationship similar to primary key and foreign key in relational databases.

Example: Cust1

秭ref

A value corresponding to an id type, thus enabling intra-document links.

Example: Cust1

秭refes

Similar to idref, except it contains multiple id type values separated by white space.

Example: Cust1 Cust2 Cust3

int

A signed integer.

Example: 11123, -123

秭token

Values that conform to the rules of the name token.

Example: Cust1

秭tokens

Similar to nmtoken, except it can have a list of nmtoken values separated by white space.

Example: Cust1 Cust2 Cust3
<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>notation</td>
<td>A NOTATION type</td>
<td>Not used</td>
</tr>
<tr>
<td>number</td>
<td>A number with no limits on the digits (2.2250738585072014E-308 - 1.7976931348623157E+308).</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Example: 111, 3.14, -123.456E+10</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>Same as float but only four-byte encoding (1.17549435E-38 - 3.40282347E+38).</td>
<td>Not used</td>
</tr>
<tr>
<td>r8, float</td>
<td>A floating point number. This data type only supports 15 digits of precision (2.2250738585072014E-308 - 1.7976931348623157E+308).</td>
<td>Not used</td>
</tr>
<tr>
<td>string</td>
<td>A string.</td>
<td>CDATA</td>
</tr>
<tr>
<td></td>
<td>Example: &quot;This is a string&quot;</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>A time in a subset of the ISO 8601 format with no date and no time zone.</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Example: &quot;08:15:27&quot;</td>
<td></td>
</tr>
<tr>
<td>time.tz</td>
<td>A time in a subset of the ISO 8601 format with no date but optional time zone.</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Example: &quot;08:15:27-05:00&quot;</td>
<td></td>
</tr>
<tr>
<td>ui1</td>
<td>A one-byte unsigned integer (0 to 255).</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Example: 1, 255</td>
<td></td>
</tr>
<tr>
<td>ui2</td>
<td>A two-byte unsigned integer (0 – 65535).</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>Example: 1, 255, 65535</td>
<td></td>
</tr>
<tr>
<td>ui4</td>
<td>A four-byte unsigned integer (0 – 4294967296).</td>
<td>Not used</td>
</tr>
</tbody>
</table>
Example: 1, 703, 3000000000

An eight-byte unsigned integer (0 – 18446744073709551615).

ui8
Example: 148343343434

A Uniform Resource Identifier (URI).

uri
Example: "urn:schemas-microsoft-com:Office9"

Hexadecimal digits representing octets with optional embedded hyphens that are ignored.

uuid
Example: 333C7BC4-460F-11D0-BC04-0080C7055A83

Not used

See Also

XDR Schema Data Types Reference | Primitive XDR Schema Data Types | Supported Data Type Conversions | XDR Schema Developer's Guide | XDR Schema Reference
Primitive XDR Schema Data Types

The World Wide Web Consortium (W3C) XML 1.0 Recommendation defines enumerated types (notations and enumerations) and a set of tokenized types. These types are referred to as primitive types in this Microsoft® XML documentation.

The primitive types include the following, defined in Section 3.3.1 of the W3C XML 1.0 Recommendation.

- **entity**: Represents the XML ENTITY type.
- **entities**: Represents the XML ENTITIES type.
- **enumeration**: Represents an enumerated type (supported on attributes only).
- **id**: Represents the XML ID type.
- **idref**: Represents the XML IDREF type.
- **idrefs**: Represents the XML IDREFS type.
- **nm-token**: Represents the XML NMTOKEN type.
- **nm-tokens**: Represents the XML NMTOKENS type.
- **notation**: Represents a NOTATION type.
- **string**: Represents a string type.

**See Also**

[XDR Schema Data Types Reference](#) | [XDR Schema Data Types](#) | [Supported Data Type Conversions](#) | [XDR Schema Developer's Guide](#) | [XML Schema Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
# Supported Data Type Conversions

The following table lists the data type conversions supported by the MSXML 2.0 and later.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Variant type</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>VT_BSTR, VT_BOOL, VT_CY, VT_DATE, VT_VARIANT, VT_DECIMAL, VT_U1I, VT_I2, VT_I4, VT_R4, VT_R8, VT_EMPTY, VT_NULL, VT_ARRAY</td>
</tr>
</tbody>
</table>
VT_I4, VT_EMPTY, VT_NULL
VT_BSTR, VT_BOOL, VT_CY, VT_VARIANT,
ui8     VT_DECIMAL, VT_UI1, VT_I2, VT_I4, VT_EMPTY, VT_NULL
r4      VT_BSTR, VT_CY, VT_VARIANT, VT_DECIMAL, VT_I2,
        VT_I4, VT_R4, VT_EMPTY, VT_NULL
r8      VT_BSTR, VT_CY, VT_VARIANT, VT_DECIMAL, VT_I2,
        VT_I4, VT_R4, VT_R8, VT_EMPTY, VT_NULL
uuid    VT_BSTR, VT_VARIANT
uri     VT_BSTR, VT_VARIANT
bin.hex VT_BSTR, VT_VARIANT, VT_ARRAY
char    VT_BSTR, VT_VARIANT, VT_I2, VT_EMPTY, VT_NULL

See Also

XDR Schema Data Types Reference | XDR Schema Data Types | Primitive XDR Schema Data Types | XDR Schema Developer's Guide | XDR Schema Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
# DTD Developer's Guide

The following sections provide more information about document type definitions (DTDs).

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Starter Kit</strong></td>
<td>This section provides an introduction to DTDs, explains some of the benefits of DTDs, and demonstrates how to create a simple DTD.</td>
</tr>
<tr>
<td><strong>Concepts</strong></td>
<td>This section provides in-depth information about DTDs.</td>
</tr>
<tr>
<td><strong>How Do I?</strong></td>
<td>This section provides a series of tutorials that demonstrate common DTD tasks.</td>
</tr>
</tbody>
</table>

**See Also**

[DTD Reference](#)
Starter Kit

This starter kit is designed to provide tools and information for working with document type definitions (DTDs).

- What is a DTD?
- Introduction to DTD Syntax
- Authoring DTDs

See Also

Concepts | How Do I?
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
What is a DTD?

Document type definitions (DTDs) provide a way of writing markup rules that specify how XML documents can be validly created. When XML 1.0 was originally specified, the DTD syntax (which is not XML-based) was inherited mainly from earlier markup languages, such as SGML and HTML.

Support for DTDs has been part of SGML for many years, and unlike XML schemas, DTDs have been widely available for use with XML since the XML 1.0 specification was first published in 1996. Therefore, DTDs are still a widely available and supported grammar for authoring and defining XML document types. Almost all validating XML parsers in wide release provide some built-in support for DTDs when validating and parsing XML documents.

DTDs, however, have the following limitations:

- Because the syntax is different from XML syntax, it can be difficult to read and write DTDs.
- DTDs provide very limited support for native data types, and no support for extending and defining custom data types. DTDs only provide for character data as a native data type.
- DTDs do not support XML namespaces well. Although you can write DTD statements that will allow you to declare an element name that includes namespace prefixes, the process by which DTDs are parsed does not enforce adherence to a namespace. Therefore, if a particular namespace URI must be used within a document.

As an alternative to DTDs, you can describe the XML document structure using XDR-based or XSD-based schemas, which offer the same benefits as DTDs but also resolve some of the limitations of DTDs.

DTDs are mainly useful when you are working with legacy XML applications that require them and do not support XML schemas.

See Also

Introduction to DTD Syntax | Authoring DTDs
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Introduction to DTD Syntax

DTDs use a specialized non-XML vocabulary, which includes the following grammar for writing and declaring markup rules that define a specific type of XML document structure:

- **ATTLIST declarations**
- **ELEMENT declarations**
- **ENTITY declarations**
- **NOTATION declarations**

DTD declaration statements can be added internally as a section within the `<!DOCTYPE>` declaration of the XML document. Alternatively, you can use a resource URI to point to an external DTD file.

For example, the following is an internal DTD that could be added to the sample XML file *(Books.xml)* to describe and validate its contents.

**Internal DTD**

```xml
<!DOCTYPE catalog [  
<!ELEMENT catalog (book+) >  
<!ELEMENT book (author, title, genre, price, publish_date, description) >  
<!ATTLIST book id ID #REQUIRED >  
<!ELEMENT author (#PCDATA) >  
<!ELEMENT title (#PCDATA) >  
<!ELEMENT genre (#PCDATA) >  
<!ELEMENT price (#PCDATA) >  
<!ELEMENT publish_date (#PCDATA) >  
<!ELEMENT description (#PCDATA) >  
]>  
```

The first line identifies "catalog" as the document type *(DOCTYPE)*, which also happens to be the name of the root element. After this, various elements are defined for the "catalog" document type.

A set of open ([]) and close (]) brackets contain this DTD as an internal section within the DOCTYPE statement. This section needs to be inserted at the top of the *books.xml* sample file so that it can be read as a set of directives for the XML parser to use when validating and parsing the remaining XML portion of
The DTD declaration statements are made within these brackets:

- The first ELEMENT statement declares the document element, in this case the `<catalog>` element. It states that `<catalog>` must include one or more `<book>` elements as children, as indicated by a trailing + sign.
- The second ELEMENT statement declares the `<book>` element, and that its contents are restricted to the following six child elements (which must be used in this order): `<author>`, `<title>`, `<genre>`, `<price>`, `<publish_date>`, and `<description>`. The `<catalog>` and `<book>` elements both exemplify what is known as a *structured-element content model*, where these elements only permit certain specified elements as their contents.
- Next, an ATTLIST statement declares a required `id` attribute for use with the `<book>` element.
- The remaining ELEMENT statements in the DTD declare that for each of the six element children for `<book>`, only text is allowed as content. This is specified by the `#PCDATA` keyword, which means parsed character data. By contrast, elements that allow either text or a combination of text and other markup have a *mixed content model*.

**See Also**

[What is a DTD?](#) | [Authoring DTDs](#)
Authoring DTDs

DTDs are usually authored and maintained using either a text editor or an application designed specifically for DTD authoring.
**DTD Editing Tools**

For writing relatively simple and short DTDs, or for making isolated changes to an existing DTD, you can use Notepad or a similar program that allows you to edit and save files as text. For more extensive DTD editing, the following are some third party applications you might want to consider evaluating or purchasing:

- TurboXML – For more information, see [TIBCO Extensibility](#).
- XML Spy – For more information, see [XML Spy](#).
- Near & Far Designer – For more information, see [Open Text](#).
Authoring a solid DTD presents a number of design challenges. For more information about DTDs, you can consult the following references:


These books include detailed or advanced discussions of issues that affect DTD design. You might find them helpful if you need to learn about DTDs in more depth.

**See Also**

[What is a DTD?](#) | [Introduction to DTD Syntax](#)
Concepts

The document type definition (DTD) provides a language for defining the structure and valid content of an XML document. This section introduces some of the fundamental concepts for working with DTDs and using them to define XML.

- Declaring DTDs
- Implementing DTDs

See Also

Starter Kit | How Do I?
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Declaring DTDs

The World Wide Web Consortium (W3C) XML 1.0 specification specifies the use of document type definitions (DTDs) with XML documents. Although a document must be well formed in order to be fully and successfully parsed, not all XML parsers validate XML against a DTD or schema.

To use a DTD with an XML document, you need to first declare it in the document type declaration statement. To do so, you can use either of the following methods, separately or in combination:

- Insert the markup declarations that make up the DTD as part of the document type declaration.
- Declare the location of a file through a URI reference that points to an external file, typically a text file named using the .dtd file extension.

For more information about declaring DTDs, see DOCTYPE declaration.
Declaring an External DTD for System Use

To reference a personal external DTD, you need to provide a Web URL for the location of the DTD. For example, the following declares a document type called "catalog" with the DTD in a separate file (MyCatalog.dtd) published on an intranet server (MyServer).

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE catalog SYSTEM "http://MyServer/MyCatalog.dtd">
```

While the DTD might be available and downloaded to others located on the same local network, the SYSTEM keyword helps make it clear to the XML parser that the URL is the only method to use to locate the DTD. In this example, if the Web service on MyServer is offline, the DTD will not be found and the document cannot be validated when parsed.

If you are using or providing your DTD as a separate downloadable file that will be copied to the same local folder with your XML document, a simpler way to write the URI reference is just to include the file name, like this:

```xml
<!DOCTYPE catalog SYSTEM "MyCatalog.dtd">
```

Optionally, you can include path information to help Windows locate the external DTD file on either a local drive or a UNC file share, as in the following examples:

```xml
<!DOCTYPE catalog SYSTEM "C:\temp\MyCatalog.dtd">
<!DOCTYPE catalog SYSTEM "\\MyServer\MyShare\MyCatalog.dtd">
```
Naming a Public External DTD

If your DTD file is for use with XML documents that are published openly on the Internet, using a standard public naming convention will allow others to find and locate the file. To reference a public external DTD, you typically will use the PUBLIC keyword in combination with a public identifier and a URI reference, as shown in the following example:

```
<!DOCTYPE rootElement PUBLIC "PublicIdentifier" "URIreference">
```

This allows an XML application to first attempt to look up the public identifier and resolve it to the correct DTD to use for validating the document. The naming convention widely used for the public identifier is called a formal public identifier (FPI) and consists of the following fields:

- Type
- Owner
- Label
- Language

**Type**

The `type` is a "+" or "-" sign. The "+" sign indicates that the file is approved by a standards body such as the International Standards Organization (ISO). The "-" sign indicates the file is not a recognized standard. Type is the first character in the example below.

```
+//Owner Name//DTD Description_of_DTD//EN//
```

**Owner**

The `owner` identifies the person or organization that wrote and maintains the DTD. Owner is highlighted in the example below.

```
+//Owner Name//DTD Description_of_DTD//EN//
```

**Label**
The *label* gives a description of the DTD. It begins with a space after the characters "DTD". Label is highlighted in the example below.

```xml
+//Owner Name//DTD Description_of_DTD//EN//
```

**Language**

The *language* is a two-character abbreviation of the language of the XML document to which the DTD applies. The letters EN are used to specify the English language. Language is highlighted in the example below.

```xml
+//Owner Name//DTD Description_of_DTD//EN//
```
Declaring a Public External DTD

To declare a public external DTD, you use the standard naming convention for the public identifier that names and refers to the DTD. As a backup, a static URL can be placed after the public identifying name and used if the public name fails to resolve.

The declaration in the XML document will look similar to the following:

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE root element SYSTEM "+//Owner Name//DTD Description_of_DTD//EN//"
"http://MyBackupServer/My.dtd"
```

See Also

Implementing DTDs
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
Implementing DTDs

A DTD can be implemented as a single marked section in the `DOCTYPE` declaration, or as a single external DTD file. In some cases, a DTD can become quite large. When this happens, additional steps are required to make the DTD maintainable.

The following are the primary ways to implement maintainable and reusable DTDs:

- [Using Parameter Entities](#)
- [Modularizing the DTD](#)
Using Parameter Entities

Parameter entities are a way of declaring and using entity contents internally within the DTD itself. Though not required, parameter entities can be helpful if you are working on a large DTD that has a content structure you want to reuse and apply elsewhere in the DTD.

For example, if you were writing a DTD to generate various e-business documents to be sent to customers in e-mail message format, you might declare a parameter entity such as the one in the following example.

<!ENTITY % email "from, to, subject, message" >
<!ELEMENT WelcomeMessage (%email;) >
<!ELEMENT MonthlyBilling (%email;) >
<!ELEMENT PaymentConfirmation (%email;) >

In this example, an ENTITY statement first declares a parameter entity called "email", making its contents available for reuse in other markup declarations in the same DTD. The three ELEMENT declarations that follow it can then apply the value of the entity by referencing it as "%email;". This specifies a common and standard definition of how to mark up e-mail for documents created using the DTD that contains these statements.

If you later decided to redo or extend this DTD to support optional CC: and BCC: fields, you could revise the ENTITY declaration as follows:

<!ENTITY % email "from, to, cc?, bcc?, subject, message" >

All of the other elements that refer to this entity and use its contents will inherit the change.
Modularizing a DTD

Parameter entities allow you to reuse the value of a single DTD declaration to other locations in the same DTD, but in some cases you might have other reasons to partition a section of a large external DTD into a separate file that can be linked to the main DTD (.dtd) file. This type of implementation is known as a modularized DTD.

Extensible HTML (XHTML) is one example of where a modular DTD implementation is helpful. XHTML defines HTML as an XML document type. This allows a document that conforms to XHTML to be processed as either HTML or XML. One of the key differences between XML and HTML is the support for built-in or predefined general entities. HTML supports a large number of these, while XML supports a very limited number of character and entity references natively. Therefore, if you are using the standard XHTML DTD or adapting it to your own DTD, you might notice that it uses parameter entities to declare separate DTD modules, as shown in the following example:

```xml
<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin 1 for XHTML//EN"
   "xhtml-lat1.ent">
%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols for XHTML//EN"
   "xhtml-symbol.ent">
%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special for XHTML//EN"
   "xhtml-special.ent">

The highlighted values refer to three separate files (xhtml-lat1.ent, xhtml-symbol.ent, and xhtml-special.ent) that are DTD modules included as part of the XHTML 1.0 strict DTD. Because each of these sets of definitions is a fairly long and coherent section of the DTD, it makes sense in this case that they have been divided out into separate files for each of the three categories of HTML character entities.

Modular DTDs are a good fit when a DTD might be too large to be manageable
as a single file. However, modular DTDs can, if overused, have an impact on parsing performance and document load times. Usually, a single layer of modularity (a main DTD module with direct links to all child modules) provides the right balance.

See Also

Declaring DTDs
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
How Do I?

This section provides the following tutorials that demonstrate common DTD tasks.

- Check for DTD Errors
- Validate a Document Against Its DTD

See Also

Starter Kit | Concepts
Check for DTD Errors

If a DTD is used within or linked to an XML document, a quick way to check for errors in the DTD is to try loading an XML document file that uses it in Internet Explorer.

Example

The following are an external DTD file with an error (a single missing close parentheses) and an XML file that links to and uses this DTD.

DTD File (errored.dtd)

```xml
<!ELEMENT catalog (book+) >
<!ELEMENT book (author, title, genre, price, publish_date, descr)>
<!ATTLIST book id ID #REQUIRED>
<!ELEMENT author (#PCDATA) >
<!-- The following line has the error -->
<!ELEMENT title (#PCDATA) >
<!ELEMENT genre (#PCDATA) >
<!ELEMENT price (#PCDATA) >
<!ELEMENT publish_date (#PCDATA) >
<!ELEMENT description (#PCDATA) >
```

XML File (books-using-errored.xml)

```xml
<?xml version="1.0"?>
<!DOCTYPE catalog SYSTEM "errored.dtd">
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
</catalog>
```

Try It!
1. Open Notepad.
2. Copy the errored.dtd. Paste it into the Notepad window.
3. From the File menu, click Save As. Save the file as errored.dtd to a folder on your computer.
5. From the File menu, click Save As. Save the file as books-using-errored.xml to the same folder in which you saved errored.dtd.
6. Open books-using-errored.xml in Internet Explorer.

Output

Results should appear in the browser window similar to the following, indicating an error in the DTD.

<P STYLE='line-height:15.0pt'><SPAN STYLE='font-size:13.0pt;font-family:Verdana'>The XML page cannot be displayed</SPAN></P>
<P STYLE='line-height:11.0pt'><SPAN STYLE='font-size:8.0pt;font-family:Verdana'>Cannot view XML input using style sheet. Please correct the error and then button, or try again later. </SPAN></P>
<P STYLE='line-height:11.0pt'><B><SPAN STYLE='font-size:8.0pt;font-family:Verdana'>Mixed content model cannot contain this character. Error processing resource 'file:///C:/temp/books.dtd'. Line 6, Position 37 </SPAN></B></P>

If you correct the error in errored.dtd by adding the missing close parentheses to line 6, you can successfully reload the books-using-errored.xml file.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
Validate a Document Against Its DTD

When using Internet Explorer to load files, you can check for DTD errors, but to use your DTD to validate an XML file, you need to set the value of validateOnParse to True prior to loading a file into the XML DOM. (Internet Explorer leaves this property set to False as its default for loading XML file types.)

Example

The following VBScript example includes the following:

- An external DTD file.
- An XML file, **books.xml**, that is well-formed but contains a validation error.
- A Windows batch file written in VBScript. This file loads and validates either an XML file or a folder tree containing XML file types, and returns the validation results.

DTD File (**books.dtd**)

```xml
<!ELEMENT catalog (book+) >
<!ELEMENT book (author, title, genre, price, publish_date, description) >
<!ATTLIST book id ID #REQUIRED >
<!ELEMENT author (#PCDATA) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT genre (#PCDATA) >
<!ELEMENT price (#PCDATA) >
<!ELEMENT publish_date (#PCDATA) >
<!ELEMENT description (#PCDATA) >
```

XML File (**books-well-formed.xml**)

```xml
<?xml version="1.0"?>
<!DOCTYPE catalog SYSTEM "books.dtd">
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
  </book>
</catalog>
```
<!-- The following line is not valid according to books.dtd. -->

```xml
<book>
  <cost>44.95</cost>
  <publish_date>2000-10-01</publish_date>
  <description>An in-depth look at creating applications with XML.</description>
</book>
</catalog>

**VBScript File (msval.vbs)**

Option Explicit

' Initialize global objects and variables.
Dim fso, f, file, folder, filepath, dir, coll
Dim fspec, strFileName, shortName, strResult
Dim LineOfEquals, strFile, strFiles, strFileExt
Dim Files, StartingFolder, WshShell, strDesktop
Dim S, SubFolders, SubFolder, procFilesCount
Dim xmlDoc, state
Dim sLogPath

' Set global constants and variables.
Const OpenFileForAppending = 8
LineOfEquals = "=============================================" & vbCrLf

set WshShell = WScript.CreateObject("WScript.Shell")
strDesktop = WshShell.SpecialFolders("Desktop")
Set fso = CreateObject("Scripting.FileSystemObject")
sLogPath = strDesktop & "\msval.txt"

Sub ShowHelp
  WScript.Echo vbCrLf & _
  "About:" & Chr(9) & "Msval.vbs is an XML file validator." & vbCrLf
  vbCrLf & _
  "Syntax:" & Chr(9) & "msval [input_file_or_folder]" & vbCrLf & _
  vbCrLf & _
  "Examples:" & vbCrLf & vbCrLf & _
  Chr(9) & "msval my.xml" & vbCrLf & _
  Chr(9) & "msval C:\MyFolderContainingXML" & vbCrLf & _
  Chr(9) & "msval ..\..\MyFolderContainingXML" & vbCrLf & vbCrLf & _
  "Notes:" & Chr(9) & "If XML file is specified, results are " & _
  "returned in a console message." & vbCrLf & vbCrLf & _
  Chr(9) & "If a folder is specified, a report file, Msval.txt," & _
  " is generated" & vbCrLf & _
  Chr(9) & "on your desktop and validation results are recursive" & _
  " for XML" & vbCrLf & _
  Chr(9) & "files found in the specified folder and all of its" & _
  " subfolders." & vbCrLf
  Exit Sub
```
End Sub

Sub ErrorOut
    Wscript.Echo (vbCrLf & "Status: MSVAL failed." + vbCrLf)
    Wscript.Quit
End Sub

Sub ValidateAsXmlFile
    Set xmlDoc = CreateObject("Msxml2.DOMDocument.5.0")
    xmlDoc.validateOnParse = True
    xmlDoc.async = False
    xmlDoc.load(strFileName)
    Select Case xmlDoc.parseError.errorCode
        Case 0
            strResult = "Valid: " & strFileName & vbCrLf
        Case Else
    End Select
    ' Create log file for storing results when validating multiple files.
    Set f = fso.OpenTextFile(sLogPath, OpenFileForAppending)
    f.WriteLine strResult
    f.Close

    ' Increment processed files count.
    procFilesCount = procFilesCount + 1
    ' Release DOM document object
    Set xmlDoc = Nothing
End Sub

Function WalkSubfolders(Folder)
    Dim strFolder, currentFolder, strCurPath
    Set currentFolder = fso.GetFolder(Folder)
    strCurPath = currentFolder.Path
    strFolder = vbCrLf & LineOfEquals & vbCrLf & "Folder: " & strCurPath & vbCrLf & LineOfEquals & vbCrLf

    ' Open the log file and append current subfolder.
    Set f = fso.OpenTextFile(sLogPath, OpenFileForAppending)
    f.Write strFolder
    f.Close
    strFolder = ""
    Set Files = currentFolder.Files
If Files.Count <> 0 Then
    ' Walk the collection. If the file is XML, ' load and validate it.
    For Each File In Files
        strFileName = fso.GetAbsolutePathName(File)
        strFileExt = Right(strFileName, 4)
        Select Case strFileExt
            ' Process all known XML file types.
            Case ".xml" ValidateAsXmlFile
            Case ".xsl" ValidateAsXmlFile
            Case ".xsd" ValidateAsXmlFile
            Case Else
        End Select
    Next
End If

' Open the log file and append file list from current subfolder.
Set f = fso.OpenTextFile(sLogPath, OpenFileForAppending)
f.Write strFiles
f.Close
strFiles = ""

Set SubFolders = currentFolder.SubFolders

If SubFolders.Count <> 0 Then
    For Each SubFolder In SubFolders
        strFolder = strFolder & WalkSubfolders(SubFolder)
    Next
    strFolder = strFolder & vbCrLf
End If

End Function

Sub WriteEOFSummary
    Set f = fso.OpenTextFile(sLogPath, OpenFileForAppending)
    strResult = vbCrLf & LineOfEquals & _
                 "Processing completed at " & Now & vbCrLf & _
                 procFilesCount & " files processed" & vbCrLf & _
                 LineOfEquals
    f.Write strResult
    f.Close
    strResult = "Results written to " & sLogPath & vbCrLf & _
                 "Files processed: " & procFilesCount & vbCrLf & _
                 vbCrLf & "Do you want to view the results now?"
    MsgBox strResult, vbYesNo, "MSVAL: Processing completed"
    If vbYes Then
        WshShell.Run("%windir%\notepad " & sLogPath)
    End If
End Sub

Function ProcessStandAloneFile(sFile)
Dim basename, str, xdoc
Set f = fso.GetFile(fspec)
basename = f.Name
' Load XML input file & validate it
Set xdoc = CreateObject("Msxml2.DOMDocument.5.0")
xdoc.validateOnParse = True
xdoc.async = False
xdoc.load(fspec)
If xdoc.parseError.errorCode = 0 Then
    str = basename & " is valid"
ElseIf xdoc.parseError.errorCode <> 0 Then
    str = basename & " is not valid" & vbCrLf & _
        xdoc.parseError.reason & " URL: " & Chr(9) & _
        xdoc.parseError.url & vbCrLf & "Code: " & Chr(9) & _
        xdoc.parseError.errorCode & vbCrLf & "Line: " & _
        Chr(9) & xdoc.parseError.line & vbCrLf & _
        "Char: " & Chr(9) & xdoc.parseError.linepos & vbCrLf & _
        "Text: " & Chr(9) & xdoc.parseError.srcText
    End If
End Function

Sub Main
' Initialize files count
procFilesCount = 0

' Get the folder to scan for files.
If Wscript.Arguments.Length > 0 Then
    fSpec = Wscript.Arguments.Item(0)
    fSpec = fSpec & "\"
Else
    ShowHelp
    WScript.Quit
End If

fspec = fso.GetAbsolutePathName(fspec)
If fso.FileExists(fspec) Then
    strResult = ProcessStandAloneFile(fspec)
    Wscript.Echo strResult
    Wscript.Quit
ElseIf fso.FolderExists(fspec) Then
    ' Executes a 'DIR' command into a collection.
    Set dir = fso.GetFolder(fspec)
    Set coll = dir.Files
    ' Create the log file on the user's desktop.
    Set f = fso.CreateTextFile(sLogPath, 1)
    strResult = vbCrLf & LineOfEquals & sLogPath & _
        " at " & Now & vbCrLf & LineOfEquals & vbCrLf
    f.Write strResult
    f.Close
Try It!

1. Open Notepad.
3. From the File menu, click Save As. Save the file as books.dtd to a folder on your computer.
5. From the File menu, click Save As. Save the file as books-well-formed.xml to the same folder in which you saved books.dtd.
7. From the File menu, click Save As. Save the file as msval.vbs to the same folder you used for saving files in steps 3 and 5.
8. If needed, open Windows Explorer and browse to the folder used in steps 1 through 6.
9. To validate books-well-formed.xml, drag and drop it on msval.vbs.

Output

A message box should appear with content similar to the following, indicating that books-well-formed.xml is not valid according to its DTD, books.dtd.

books-well-formed.xml is not valid
Element content is invalid according to the DTD/Schema.
Expecting: price.
The script can also perform validation on multiple XML files with .xml, .xsd, or .xsl extensions. It can be used at the command prompt to perform batch validation as well. For more information, double-click the msval.vbs on your machine to view command help.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
## DTD Reference

This section describes the syntax for the four types of defining statements available for authoring document type definitions (DTDs).

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATTLIST (Attribute List)</strong></td>
<td>Declares a list of XML attributes. These attributes are qualified by name, data type, implied default values, and the name of any elements that permit their use.</td>
</tr>
<tr>
<td><strong>ELEMENT</strong></td>
<td>Declares an XML element type name and its valid sub-elements (&quot;children&quot;).</td>
</tr>
<tr>
<td><strong>ENTITY</strong></td>
<td>Declares special character references, text macros (much like a C/C++ #define statement), and other repetitive content from external sources (like a C/C++ #include statement).</td>
</tr>
<tr>
<td><strong>NOTATION</strong></td>
<td>Declares external non-XML content (for example, binary image data), and the external application that handles that content.</td>
</tr>
</tbody>
</table>

**See Also**

[DTD Developers Guide](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
**ATTLIST (Attribute List)**

The **ATTLIST** statement is used to list and declare each attribute that can belong to an element. It first specifies the name of the element (or elements) for which the attribute list will apply. It then lists each attribute by name, indicates whether it is required, and specifies what character data it is allowed to have as a value.

**Syntax**

```xml
<!ATTLIST elementName attributeName dataType default >
```

**Parameters**

- **elementName**
  The name of the element to which the attribute list applies.

- **attributeName**
  The name of an attribute. This parameter can be repeated as many times as needed to list all attributes available for use with `elementName`.

- **dataType**
  The data type for the attribute named in the `attributeName` parameter, which must be one of the following:
  - **CDATA** – The attribute will contain only character data.
  - **ID** - The value of the attribute must be unique. It cannot be repeated in other elements or attributes used in the document.
  - **IDREF** – The attribute references the value of another attribute in the document, of ID type.
  - **ENTITY** – The attribute value must correspond to the name of an external unparsed **ENTITY**, which is also declared in the same DTD.
  - **ENTITIES** - The attribute value contains multiple names of external unparsed entities declared in the DTD.
  - **NMTOKEN** – The attribute value must be a name token. Name tokens allow character data values, but are more limited than CDATA. A name token can contain letters, numbers, and some punctuation symbols such as periods, dashes, underscores and colons. Name token values, however, cannot contain any spacing characters.
  - **NMTOKENS** - The attribute value contains multiple name tokens.
See the description for NMTOKEN and Enumerated for more information.

- **Enumerated** – The attribute values are limited to those within an enumerated list. Only values that match those listed are validly parsed. All enumerated data types are enclosed in a set of parentheses with each value separated by a vertical bar ("|").

```
default
```

The default value for the attribute named in attributeName. The following table describes the possible defaults.

<table>
<thead>
<tr>
<th>Defaults</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#REQUIRED</td>
<td>The attribute must appear in the XML document or a parsing error will result. To avoid a parse error in some cases, you can optionally use the defaultValue field directly following this keyword.</td>
</tr>
<tr>
<td>#IMPLIED</td>
<td>The attribute can appear in the XML document, but if omitted, no parsing error will result. Optionally, in some cases you can also use the defaultValue field directly following this keyword.</td>
</tr>
<tr>
<td>#FIXED</td>
<td>The attribute value is fixed in the DTD and cannot be changed or overridden in the XML document. If this keyword is used, the defaultValue field directly following this keyword must also be used to declare the fixed attribute value.</td>
</tr>
</tbody>
</table>

```
defaultValue
```

A default or fixed value. The parser inserts this value into the XML document when the attribute is missing or is not used in the XML document. All values must be enclosed in a set of quotation marks (either single or double quotes).

**Note**  For each ATTLIST declaration made in the DTD, only one occurrence of the elementName needs to be used. The attributeName, dataType, and default parameters define each attribute in the list and can be repeated as many times as needed until you have listed and defined all attributes available for use with elementName.

**Example**

This example declares the following for the `<book>` element:
- An optional publisher attribute that can only contain character data.
- A fixed reseller attribute with its value set to "MyStore".
- A required ISBN attribute that must contain a unique identifying value for each <book> element in the XML document.
- A required InPrint attribute that must contain either a "yes" or "no" value. The default enforces a "yes" value when the value is not explicitly set in the XML document.

```xml
<!ATTLIST book
  publisher CDATA #IMPLIED
  reseller CDATA #FIXED "MyStore"
  ISBN ID #REQUIRED
  inPrint (yes|no) "yes"
>
```

See Also

[ELEMENT] | [ENTITY] | [NOTATION]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
ELEMENT

The ELEMENT statement is used to declare each element used within the document type defined by the DTD. It first declares the element by name, and then specifies what content is allowed for the element.

Syntax

<!ELEMENT name content >

Parameters

name
The name of the element. Exact case is required.

content
The allowable content model for the element, which must be one of the following:

- **ANY** - Any content is allowed within the element. When used in an element declaration, this keyword permits an open unrestricted content model for the elements and any of its child nodes.
- **EMPTY** - The element is not allowed to have content and must remain empty.
- **Declared content rule** – For this option, you need to write a content rule and enclose it within a set of parentheses.

The following table displays the reserved keyword or punctuation symbols that can be used along with names of other elements that are declared within the DTD to construct the element content rule.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PCDATA</td>
<td>Allows parsed character data as part of the content of the element.</td>
</tr>
<tr>
<td></td>
<td>The name of an element. This can be the element being defined, or other elements defined by name in the DTD, using additional ELEMENT declarations.</td>
</tr>
</tbody>
</table>
If there are no other punctuation markers or symbols in the content rule, one and only one occurrence of the named element is allowed and required.

Although at least one set of parentheses is required whenever declaring a content model for an element, you can nest additional sets of parentheses as well. This technique can be used to define a more complex content model for an element.

A vertical bar can be used to separate two named elements. When used, it indicates that either element (i.e. the one before the bar or the one after it) can appear as child elements.

A vertical bar can be used to separate two named elements or nested rules. When used, it indicates that the elements/rules must appear in the order specified.

The comma can be used to separate two named elements. When used, it indicates that the elements must appear in the order specified.

The question mark can be used as a suffix/operand. When used, it indicates that the element/rule that precedes it is optional, but if it is used it can occur only once at this point in the XML document structure.

The plus sign can be used as a suffix/operand. When used, it indicates that the element/rule that precedes it is required, and can occur more than once at this point in the XML document structure.

The asterisk can be used as a suffix/operand. When used, it indicates that the element/rule that precedes it is optional, and can occur more than once at this point in the XML document structure.

Examples

- Declares a <test> element that can contain any content:

```xml
<!ELEMENT test ANY >
```

- Declares an <Image> element that must be empty (i.e. cannot have content):

```xml
<!ELEMENT Image EMPTY >
```
• Declares a `<title>` element that can contain character data only (no other markup):

```xml
<!ELEMENT title (#PCDATA) >
```

• Declares a `<fruit>` element that contain either an `<apple>` element or an `<orange>` element:

```xml
<!ELEMENT fruit (apple|orange) >
```

• Declares a `<book>` element that must contain an `<author>` element followed by a `<title>` element:

```xml
<!ELEMENT book (author,title) >
```

• Declares a `<memo>` element that must contain a `<body>` element which can be optionally followed by a `<postscript>` element:

```xml
<!ELEMENT memo (body,postscript?) >
```

• Declares a `<catalog>` element that must contain one or more `<book>` elements:

```xml
<!ELEMENT catalog (book+) >
```

• Declares a `<table>` element that can either be empty or contain `<rowset>` elements:

```xml
<!ELEMENT table (rowset*) >
```

**See Also**

[ATTLIST (Attribute List)](#)  |  [ENTITY](#)  |  [NOTATION](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schemas
ENTITY

The ENTITY statement is used to define entities in the DTD, for use in both the XML document associated with the DTD and the DTD itself.

Syntax

<!ENTITY [%] name [SYSTEM|PUBLIC publicID] resource [NDATA notation]

Parameters

name
The name of the entity. Required for all entity definitions.

publicID
The public identifier for the entity. Only required if the declaration uses the PUBLIC keyword.

resource
The value for the entity. Required for all entity definitions. In the case of internal entities, this is a text string that is parsed and expanded. In the case of external entities, it is a Uniform Resource Identifier (URI) that identifies an external entity such as a file name or file type.

notation
The name of a notation declared elsewhere in the DTD using the NOTATION statement. Only required when declaring an unparsed entity through the use of the non-XML data (NDATA) keyword.

Examples

General entity (parsed internal)

The following declares an internal general entity that can be referenced as &oumlaut; in XML documents that use the DTD.

<!ENTITY oumlaut "&#38;#246;">  

The XML parser expands the raw entity value (&#38;#246;) to make it part of
the internal XML document. In the final parsed document output, the "&amp;#38;" value becomes an "&" and the value appears as "&amp;#246;" in the final parsed document.

**Parameter entity (parsed internal)**

The following declares an internal parameter entity that can be referenced as %lists; in other places in the DTD. This allows repeated references to the XHTML list elements, <ul> and <ol>.

```xml
<!ENTITY % lists "ul | ol">
```

**Unparsed external entity**

The following declares an unparsed external entity that allows you to encode links to non-XML data (in this case, an image file in GIF format) for inclusion in your XML document.

```xml
<!ENTITY image.banner SYSTEM "banner.gif" NDATA image_gif >
```

In this case, the use of the NDATA keyword requires that a NOTATION declaration for a notation called "gif" must also be used in combination with this declaration. For more information, see [NOTATION](#).

**Remarks**

Entities can be used in a variety of ways. If you are not sure how to define an entity for use, the following questions might help you better understand the different conditions for using the entities you declare in the DTD:

- Is the value of the entity simply to be used as is to abbreviate a larger text string (i.e. internal entities) that might change during the life of the document? Or will it be used as a pointer to something external to the DTD or XML document (i.e. external entities), like a file name or MIME type?
- Should the URI value of the entity be parsed (i.e. expanded as part of the DTD or an XML document that uses it), or remain unparsed (i.e. the XML parser will omit its contents from the final parsed document)?
- If parsed, should the expanded entity content become part of the XML document that uses the DTD (i.e. general entities) or part of the DTD itself (i.e. parameter entities)?
See Also

ATTLIST (Attribute List) | ELEMENT | NOTATION
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Schen
NOTATION

The NOTATION statement is used to define notations. Notations allow the XML document to pass notifying information to external applications.

Syntax

<!NOTATION name [SYSTEM|PUBLIC publicID] resource >

Parameters

name
The name of the notation. Required for all notation definitions.

publicID
The public identifier for the notation. Only required if the declaration uses the PUBLIC keyword.

resource
The value for the notation. Required for all notation definitions. Typically, if the notation is public it is a Uniform Resource Identifier (URI), readable to humans but not to machines. For system-based notations, the value might reference the file name of the application on the system that can be used to assist in handling non-XML data. For example, you might declare a notation to assist in handling an unparsed external entity, such as an image file in JPEG or GIF file format.

Examples

Notation for declaring your own data type

The following declares a notation that can be used to reference a custom catalog document type you have defined.

<!NOTATION MyCatalog SYSTEM "http://example.microsoft.com/catalog">

Notation for declaring a MIME type
The following declares a notation in the DTD to allow support for unparsed entities that reference image files in the GIF file format.

`<!NOTATION image_gif SYSTEM "image/gif">`

**See Also**

[ATTLIST (Attribute List)](#) | [ELEMENT](#) | [ENTITY](#)
This guide introduces the Microsoft COM implementation of the Schema Object Model (SOM) standard. The SOM provides a navigable set of classes that directly reflect the W3C XML Schema definition language (XSD) specification. These classes enable you to walk through the elements of an XML Schema document and obtain information about the properties and declarations, and about the relationships between them. You can then compile and validate your XML documents against this XML Schema by using the validate method of the DOM.

The Microsoft SOM implementation provides both Microsoft® Visual Basic® and Microsoft Visual C++® interfaces.

The following sections provide more information about creating applications with the SOM.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter Kit</td>
<td>This section explains the benefits of the SOM, and offers ideas for building SOM applications.</td>
</tr>
<tr>
<td>Concepts</td>
<td>This section introduces some of the fundamental concepts that relate to the SOM.</td>
</tr>
<tr>
<td>How Do I?</td>
<td>This section provides tutorials for using the SOM with other XML application programming interfaces, such as the Document Object Model (DOM) and the Simple API for XML (SAX).</td>
</tr>
</tbody>
</table>
Starter Kit

This starter kit is designed to provide tools and information for working with the Schema Object Model (SOM).

- [What is the SOM?](#)
- [Frequently Asked Questions about the SOM?](#)
- [Practical Uses of the SOM](#)

See Also

- Concepts | How Do I?
What is the SOM?

The Schema Object Model (SOM) provides a navigable set of classes that directly reflect the W3C XML Schema Definition (XSD) specification. The W3C recommendation document, "XML Schema Part 0: Primer," is available at http://www.w3.org/TR/xmlschema-0/.

The interfaces of the SOM extract information about declarations in an XML Schema document. An XML Schema is retrieved from a schema cache. This cache is accessed by the get method of the IXMLDOMSchemaCollection2 interface.

The SOM provides access to items in an XML Schema. Many of the interfaces inherit from other interfaces of the SOM. This inheritance allows a single property to provide several interfaces with individual information, using the same function.

SOM elements returned from interface properties originate at the top level of the XML Schema. The top level of an XML Schema consists of declarations that are not nested inside other declarations. The example below shows a top-level <complexType> declaration, which contains a second-level <element> declaration.

```xml
<schema xmlns = "http://www.w3.org/2001/XMLSchema">
  <complexType name = "TopLevelType">
    <all>
      <element name = "SecondLevelElement"/>
    </all>
  </complexType>
</schema>
```

To navigate through the XML Schema to the secondary and lower child items, you must use the ISchemaComplexType.contentModel property. For a full description of navigating to the child items, see Finding Child Elements of Top-Level XML Schema Elements.

See Also
Other Resources

W3C XML Schema Part 0: Primer Recommendation
Frequently Asked Questions about the SOM

The following are some frequently asked questions about the SOM.

- How do I get a schema object that I can query?
- What is the schema cache used for?
- What schema version does the SOM use?
- What does the W3C have to do with the SOM?
- How does the DOM work with the SOM?
- How does SAX work with the SOM?
- What is an item collection used for?
- Does the SOM support the multi-thread apartment model?
- How has the getDeclaration method changed in MSXML 4.0 and later?

How do I get a schema object that I can query?

You obtain a schema object from a schema cache. The schema cache is created with the IXMLDOMSchemaCollection2 interface. For more information, see Loading a Schema for Use with the ISchema Interface, and the ISchema interface reference topic.

What is the schema cache used for?

The schema cache is an object created with the IXMLDOMSchemaCollection2 interface. The schema cache stores XML Schema documents according to an associated target namespace. The target namespace must be unique for each XML Schema in the cache.

The schema cache was designed to support two types of use. First, it was designed to improve performance through schema caching. This can significantly reduce the overhead when using external schemas for validation. Secondly, it can be used to override schemas referenced or included inline.
What schema version does the SOM use?

The SOM uses the XML Schema definition language (XSD) as its schema platform. The W3C recommendation for the XML Schema is found at http://www.w3.org/2001/XMLSchema#intro. This recommendation includes a link to http://www.w3.org/2001/XMLSchema.xsd, the XML Schema that is used to validate XML Schema elements, attributes, and data types.

See Also

XML Schema Reference (XSD)

What does the W3C have to do with the SOM?

The SOM conforms to the W3C specifications, which are available at the Web site www.w3.org/TR/xmlschema-0/. SOM interfaces are created from the schema component definitions. For an example of the definitions for an element, see the specifications document, at URL http://www.w3.org/TR/xmlschema-1/#Element_Declaration_details.

The rest of the definitions can be found throughout the specifications document.

How does the DOM work with the SOM?

The DOM has two methods that return a schema object. The first is the getDeclaration method of the IXMLDOMSchemaCollection2 interface. The second is the getSchema method of the IXMLDOMSchemaCollection2 interface. For more information, see Using the DOM with the SOM.

How does SAX work with the SOM?
SAX has two functions that return a schema object. The first is the `schemaElementDecl` method of the `IMXSHEMADECLHandler` interface. The second is the `getTypeFromName` method of the `ISAXAttributes` interface. For more information, see *Using SAX with the SOM*.

**What is an item collection used for?**

An item collection is used throughout the SOM for object storage. Many of the interface functions return item collections that contain an array of specific or mixed objects. The item collection is indexed, and begins with the index of zero. The objects returned in the collection implement their specified interface when they are queried individually. Use the `itemType` property of the `ISchemaItem` interface to get the type of the items in the collection.

**See Also**

*ISchema Item Collection Interface*

**Does the SOM support the multi-thread apartment model?**

Yes, the SOM does support the multi-thread apartment model. You can marshal between rental mode and free threaded model in your application.

**Note** In MSXML, "free-threaded" means `ThreadingModel='Both'`, and cross-thread marshalling is supported.

**How has the getDeclaration method changed in MSXML 4.0 and later?**

In MSXML 4.0 Beta 2, the `getDeclaration` method was called using an `XMLSchemaCache` object. Because of some inconsistencies related to this design, the `getDeclaration` call was changed for MSXML 4.0 final release.
The following Visual Basic example shows the syntax used in MSXML 4.0 Beta 2 to make the getDeclaration method call.

```vbc
set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
nsTarget="http://www.example.microsoft.com/po"
oSchemaCache.add nsTarget, "po.xsd"
set oDoc = CreateObject("Msxml2.DOMDocument.5.0")
oDoc.async = false
oDoc.load "po.xml"
oDoc.setProperty "SelectionLanguage", "XPath"
Set oDecl = oSchemaCache.getDeclaration(oNode)

In this design, the appropriate schema was only looked for using the XMLSchemaCache object, ignoring the possibility that the information returned might not have any relationship to the actual schema information used in document validation or default attribute lookup.

For the final 4.0 release and later versions, the getDeclaration method has been changed in the following ways:

1. For the XMLSchemaCache object, the getDeclaration method will now return a result of "E_NOTIMPL", indicating it is not implemented.
2. The getDeclaration method is still part of the IXMLDOMSchemaCollection2 interface, but will return "E_NOTIMPL" when called from the IXMLDOMSchemaCollection interface.
3. The getDeclaration method is now implemented through the namespaces property of the DOMDocument2 object.
4. The getDeclaration method has also been modified to include the declarations of XML schemas used by the associated document. This provides access to XML schemas loaded using xsi:schemaLocation, as well as XML schemas loaded from a schema cache object.

The following modified example contains the syntax used to get a declaration in MSXML 5.0.

```vbc
set oDoc = CreateObject("Msxml2.DOMDocument.5.0")
oDoc.async = false
oDoc.load "po.xml"
' Retrieve the declaration for country attribute of the shipTo element
Set oDecl = oDoc.namespaces.getDeclaration(oDoc.documentElement)
WScript.Echo oDecl.namespaceURI
```

[Return to Top]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel
Practical Uses of the SOM

The SOM is primarily used for reading information, not for writing information. SAX and the DOM provide the tools for writing information to a document. The following topics provide examples of applications that use the SOM.

- Creating an Input Form from an XML Schema
- Creating a Document Map from an XML Schema

The following topics provide examples of applications that could be created with the SOM, but would be more easily created with SAX or the DOM. The purpose of these examples is to provide ideas that can be expanded on in other situations.

- Creating a DTD or XDR Schema from an XML Schema
- Creating a SQL Schema from an XML Schema Document

See Also

SOM Starter Kit
Creating an Input Form from an XML Schema

Creating a form that provided all the restrictions of an XML Schema would be time consuming. Instead, you can use the SOM to programmatically read the XML Schema, and to create HTML or Visual Basic input controls. These controls restrict input, and ensure that no information is entered into the form that will not validate against the XML Schema you used to create the form. The benefits of this form-based approach extend to working with long XML Schemas, or with many dissimilar XML Schemas. An application like this would quickly create the input form. It would also allow for uninterrupted data input when a change occurred in an XML Schema.

See Also

Practical Uses of the SOM
Creating a Document Map from an XML Schema

It is often useful for users to see a pictorial representation of XML Schema document restrictions. Creating a document map for an XML Schema can provide a quick reference to an XML Schema document. You can use the SOM to programmatically read the parts of the XML Schema. The values returned are then placed in various formats, such as a graphics or text file.

The Walk the SOM sample application presents a document map in a tab-indentated format.

See Also

Practical Uses of the SOM
Creating a DTD or XDR Schema from an XML Schema

An application that creates a DTD or XDR schema from an existing XML Schema can provide a useful tool for a business that deals exclusively with DTD or XDR schemas, but acquires and works with third party schemas that are in the XML Schema format. Such an application can read the XML Schema, and write a DOM document based on the properties returned. Manually reading the XML Schema would be less efficient than using the SOM to walk through its elements. You can also use the DOM to efficiently create the DTD or XDR document.

See Also

*Practical Uses of the SOM*
Creating a SQL Schema from an XML Schema Document

An application that develops a TSQL script, or that programmatically creates a Microsoft SQL Server database table (or series of related tables based on an XML Schema), can be a useful tool. The application would provide you with a database containing the data types and constraints of the XML Schema, without requiring you to manually create the database. For such an application, you can use the SOM to extract the properties of the XML Schema so that they can be written to a SQL server. A text document could be created with the TSQL Database Definition Language (DDL) scripting needed to create database elements based on the properties of the XML Schema. The other alternative would be to use the SQL API to create the database directly, then read in information from an XML Schema document using the SOM. Either approach provides you with a complete database representation created from any XML Schema document.

See Also

Practical Uses of the SOM
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel.
Concepts

The Schema Object Model (SOM) is a navigable set of classes that allow you to work with documents that are formed according to the XML Schema definition language (XSD) specification.

The following concepts will help you understand how to use the SOM with your XML applications.

- Loading an XML Schema for Use with the ISchema Interface
- When to Use the SOM
- Using the DOM with the SOM
- Using SAX with the SOM
- Finding Child Elements of Top-Level Schema Elements
- Things the SOM Cannot Do
Loading an XML Schema for Use with the ISchema Interface

An XML Schema must be loaded into a schema cache before it is available to the SOM interfaces. The add method of the IXMLDOMSchemaCollection interface adds the XML Schema into a schema cache. The getSchema method of the IXMLDOMSchemaCollection2 interface returns a schema object to be used with the ISchema interface.

Example

The following Visual Basic example shows an XML Schema being loaded into a schema cache, then retrieved into a schema object.

```vbnet
Dim oSchemaCache As New XMLSchemaCache50
Dim oSchema as ISchema
Dim nsTarget as String
nsTarget = "http://www.samples.microsoft.com/sampletarget"
oSchemaCache.add nsTarget, PO.xsd
Set oSchema = oSchemaCache.getSchema(nsTarget)
```

See Also

Concepts
When to Use the SOM

Use the SOM when you want to get information about the data type or logical structure of an XML document node. The SOM can also be used to build your own custom structures from an XML Schema.

The SOM is designed to retrieve information from items that are exposed after an XML Schema has been compiled. Unlike the Document Object Model (DOM), SOM does not navigate the XML Schema as an XML document. The SOM views the XML Schema in a compiled format.

See Also

Concepts
Using the DOM with the SOM

The DOM has two methods that return schema objects. The first is the `getDeclaration` method of the `IXMLDOMSchemaCollection2` interface. This method returns an XML Schema declaration. This declaration is used to validate the DOM node parameter sent to the method. The XML Schema declaration is returned as a `schemaItem` object. You can query the `schemaItem` using the SOM `ISchemaItem` interface.

The second is the `getSchema` method of the `IXMLDOMSchemaCollection2` interface. This method returns an `ISchema` object. You can obtain more information about the schema object by using the `ISchema` interface.

See Also

[Concepts](#) | [XML DOM Objects/Interfaces](#)
Using SAX with the SOM

SAX has two methods that return a schema object. The first is the `schemaElementDecl` method of the `IMXSchemDeclHandler` interface. This method returns an XML Schema element declaration. This declaration is used to validate the element being parsed.

The second is the `getTypeFromName` method of the `ISAXAttributes` interface. This method returns an attribute type for an unhandled type declaration. You can use the `ISchemaType` and `ISchemaComplexType` interfaces to explore the other properties of the returned type.

See Also

[Concepts](#) | [SAX2 Reference](#)
Finding Child Elements of Top-Level XML Schema Elements

SOM elements that are returned from interface properties originate at the top level of the XML Schema. Top-level elements of an XML Schema are declarations that are not nested inside other declarations.

Example

The following example shows a top-level <complexType> declaration that contains a second-level <element> declaration.

XML Schema: childsample.xsd

```xml
<schema xmlns = "http://www.w3.org/2001/XMLSchema">
  <complexType name = "TopLevelType">
    <all>
      <element name = "SecondLevelElement"/>
    </all>
  </complexType>
</schema>
```

To navigate through the schema to the secondary and lower child items, you must use the contentModel property of the ISchemaComplexType interface. This property returns a SchemaModelGroup object. The particles property of the ISchemaModelGroup interface returns a collection of elements, modelGroup objects, and any declarations that are in the <complexType> declaration of the parent element.

Example

In the following example, the schema above, childsample.xsd, is queried from a Visual Basic program.

```vbnet
Dim oSchemaCache as New XMLSchemaCache50
Dim oSchema as ISchema
Dim nsTarget as String
```
nsTarget = "http://www.samples.microsoft.com/sampletarget"
oSchematic.add nsTarget, "childsample.xsd"
Set oSchematic = oSchematicCache.getSchema(nsTarget)
Set oComplexTypes = oSchematic.types
Set oContentModelGroup = oComplexTypes(0).contentModel
Set oChildItems = oContentModelGroup.particles
strElementName = oChildItems(0).Name

**Note**  When you use the DOM to find child items, you can go directly to the node you want. With the SOM, however, you must start with a top-level item and drill down through the child elements until you reach the node you want.

**See Also**

[Concepts](#) | [Complex Type Definitions](#) | [Defining Complex Types](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel.
Things the SOM Cannot Do

The following tasks require the use of additional methods, outside the SOM.

1. The SOM cannot validate an XML instance document against a given XML Schema. You must use the `validate` method of the `IXMLDOMDocument2` interface to perform this task. The SOM can, however, validate data types against an XML Schema item, such as an element or an attribute. The SOM does this via the `isValid` method of the `ISchemaType` interface.

2. The SOM cannot modify a loaded XML Schema, or create a new XML Schema. You must use the DOM to perform these tasks. After you have made your modifications, the XML Schema can be loaded into a `SchemaCache` object to validate the XML. The SOM provides read-only access to the XML schema.

3. Unlike the DOM, the SOM does not provide a search method for child nodes. For information about navigating to a child element, see Finding Child Elements of Top-Level Schema Elements.

See Also

Concepts
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Development Guide
How Do I?

The following tutorials demonstrate common SOM tasks.

- Walk the SOM
- Locate Declarations
- Write Annotations
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel...
Walk the SOM

This sample application walks through an XML Schema and prints selected parts in a tab-structured format. The DOM Schema Cache is used to store the XML Schema. The schema object is obtained by the getSchema method of the SchemaCollection2 interface. The interfaces of the SOM are then used to walk through the items in the XML Schema. Formatting functions and methods are present throughout the application, but they are used solely for text output. They can therefore be ignored for functionality of the SOM interfaces.

This topic is divided into the following sections.

- Instructions for Running the Walk the SOM Application
- The Walk the SOM Application
- Uncommented Code for the Walk the SOM Application
Instructions for Running the Walk the SOM Application

Note  The following sample assumes that you are using a computer with Windows Scripting Host (WSH) installed, such as a computer running Windows 2000 Professional or Windows 98.

To run the application

1. Open Notepad.
3. From the File menu, click Save As. Save the file as po.xml to a folder on your computer.
4. From the File menu, click New.
6. From the File menu, click Save As. Save the file as po.xsd to the same folder in which you saved po.xml.
7. From the File menu, click New.
8. Copy the Uncommented Code for the Walk the SOM Application. Paste it into the Notepad window.
9. From the File menu, click Save As. Save the file as po.vbs to the same folder in which you saved po.xml and po.xsd.
10. Run po.vbs from either the Command Prompt or Windows Explorer.

The results will be displayed in a message box window.
# The Walk the SOM Application

This section contains the following:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>po.xml</td>
<td>The XML document.</td>
</tr>
<tr>
<td>po.xsd</td>
<td>The XML Schema document.</td>
</tr>
<tr>
<td>po.vbs Step-by-Step</td>
<td>The VBScript application, explained step-by-step with textual comments. This topic walks you through the application code.</td>
</tr>
</tbody>
</table>
The Walk the SOM application uses the following XML document.

```xml
<?xml version="1.0" ?>
<purchaseOrder xmlns="http://www.example.microsoft.com/po"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.microsoft.com/po po.xsd"
    orderDate="1999-10-20">
    <shipTo country="US">
        <name>Kim Yoshida</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Neil Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm that this is electric.</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel...
The Walk the SOM application uses the following XML Schema document.

```xml
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="www.example.microsoft.com/po"
    elementFormDefault="qualified"
    finalDefault="restriction">

<xsd:annotation>
    <xsd:documentation xml:lang="en">
        Copyright 2000 Example.Microsoft.com. All rights reserved.
    </xsd:documentation>
</xsd:annotation>

<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

<xsd:element name="comment" type="xsd:string"/>

<xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
        <xsd:element name="shipTo" type="USAddress"/>
        <xsd:element name="billTo" type="USAddress"/>
        <xsd:element ref="comment" minOccurs="0"/>
        <xsd:element name="items" type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="USAddress">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="state" type="xsd:string"/>
        <xsd:element name="zip" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN"
        fixed="US"/>
</xsd:complexType>

<xsd:complexType name="Items">
    <xsd:sequence>
    </xsd:sequence>
</xsd:complexType>
```
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity">
        <xsd:simpleType>
          <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxExclusive value="100"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice" type="xsd:decimal"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="partNum" type="SKU" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<!-- Stock Keeping Unit, a code for identifying products. -->
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <!--xsd:pattern value="\d{3}-[A-Z]{2}"/>-->
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
po.vbs Step-by-Step

This topic walks you through the Walk the SOM application. The code is interspersed with textual comments that describe each step of the application.

Click here for the [Uncommented Code for the Walk the SOM Application](#).
The Application

The code begins with some constant declarations.

' Item types:
SOMITEM_SCHEMA = 4*1024
SOMITEM_ATTRIBUTE = SOMITEM_SCHEMA + 1
SOMITEM_ATTRIBUTEGROUP = SOMITEM_SCHEMA + 2
SOMITEM_NOTATION = SOMITEM_SCHEMA + 3
SOMITEM_ANYTYPE = 8*1024
SOMITEM_DATATYPE = SOMITEM_ANYTYPE+256
SOMITEM_SIMPLETYPE = SOMITEM_DATATYPE+256
SOMITEM_COMPLEXTYPE = 9*1024
SOMITEM_PARTICLE = 16*1024
SOMITEM_ANY = SOMITEM_PARTICLE+1
SOMITEM_ANYATTRIBUTE = SOMITEM_PARTICLE+2
SOMITEM_ELEMENT = SOMITEM_PARTICLE+3
SOMITEM_GROUP = SOMITEM_PARTICLE+256
SOMITEM_ALL = SOMITEM_GROUP+1
SOMITEM_CHOICE = SOMITEM_GROUP+2
SOMITEM_SEQUENCE = SOMITEM_GROUP+3
SOMITEM_EMPTYPARTICLE = SOMITEM_GROUP+4

' Attribute uses
SCHEMAUSE_OPTIONAL = 0
SCHEMAUSE_PROHIBITED = 1
SCHEMAUSE_REQUIRED = 2
SCHEMACONTENTTYPE_EMPTY = 0
SCHEMACONTENTTYPE_TEXTONLY = 1
SCHEMACONTENTTYPE_ELEMENTONLY = 2
SCHEMACONTENTTYPE_MIXED = 3

remarks = 0

Create a schema cache object. This object will be used later to contain the XML Schema document, po.xsd.

Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")
nsTarget="http://www.example.microsoft.com/po"
Add the XML Schema document to the schema cache, using its add method. A SOM schema object is returned. The SOM interfaces will now be used to explore the schema object.

For more information about the schema cache, see the IXMLDOMSchemaCollection add and get methods.

```
oSchemaCache.add nsTarget, "po.xsd"
Set oSchema = oSchemaCache.getSchema(nsTarget)
```

Use the itemByName method of the ISchemaItemCollection interface to locate the <purchaseOrder> element.

```
result = "<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>"
```

Use the elements collection, from the schema object, to explore the information in the individual elements.

```
For Each oE in oSchema.elements
    result = result + printElement(oE, 0)
Next
For Each oA in oSchema.attributes
    result = result + printAttr(oA, t)
Next
```

```
result = result + vbNewLine
```

Use the collection of type objects (ISchemaType interface) to explore each type declaration from the schema object.

```
For Each oT in oSchema.types
    result = result + processType(oT, 0)
Next
```

```
result = result + "</xsd:schema>"
```

```
WScript.Echo result
' ---------------------------------------------------------------
```

Create a function to examine the itemType property of the type object passed to it. This function will send the type object to the appropriate function for examining the properties of the passed object.

```
Function processType(oType, t)
```
If oType.itemType = SOMITEM_ANYTYPE Then
    res = res + printTab(t+1) + "<!-- " + oType.name +" -->"
End If
If oType.itemType = SOMITEM_COMPLEXTYPE then
    res = res + processComplexType(oType, t+1)
End If
If oType.itemType = SOMITEM_SIMPLETYPE then
    res = res + processSimpleType(oType, t+1)
End If
processType = res + vbNewLine
End Function

' ---------------------------------------------------------------------

Create a function to walk through the properties of a complexType object that is passed to it.

Function processComplexType(oComplex, t)
    res = printTab(t) + "<xsd:complexType"
    Check to see if a name attribute was declared in the type declaration.
        If oComplex.name <> "" Then
            res = res + " name='" + oComplex.name +"'"
        End If
    res = res + ">
    Check the contentType property to decide how to process the four possible choices of content type.
        If oComplex.contentType = SCHEMACONTENTTYPE_EMPTY Then
            res = res + printRemark("empty")
        End If
        If oComplex.contentType = SCHEMACONTENTTYPE_TEXTONLY Then
            res = res + printRemark("textonly")
        End If
    Because the type might contain other elements, send the content model to the processGroup function that will walk through the contentModel properties.
        If oComplex.contentType = SCHEMACONTENTTYPE_ELEMENTONLY Then
            res = res + printRemark("elementonly ")
            res = res + processGroup(oComplex.contentModel, t+1)
        End If
        If oComplex.contentType = SCHEMACONTENTTYPE_MIXED Then
            res = res + "<!-- mixed -->"+vbNewLine
            res = res + processGroup(oComplex.contentModel, t+1)
        End If
res = res + vbNewline
res = res + printRestrictions(oComplex, t+1)

On Error Resume Next
Set any = oComplex.anyAttribute.name
If Err.number = 0 Then
    res = res + oComplex.anyAttribute.name
End If

Walk through each attribute declaration in the complex type. The attribute objects that are used are returned from the attributes property of the ISchemaType object.

    For Each oAttr in oComplex.attributes
        res = res + printAttr(oAttr, t+1)
    Next

    processComplexType = res + printTab(t) + "</xsd:complexType>"+vbNewline
End Function

' ---------------------------------------------------------------------

Create a function to walk through a simpleType object (ISchemaType) interface and get its properties.

Function processSimpleType(oSimple, t)
    res = printTab(t) + "<xsd:simpleType"
    If oSimple.name <> "" Then
        res = res + " name='" + oSimple.name +"'"
    End If
    res = res + "">"+vbNewline

Call a function to walk through the restrictions of the simple type that is passed to it.

    If oSimple.baseTypes.length = 1 Then

There is only one base type. Therefore, send the type object to the function.

        res = res + printRestrictions(oSimple, t+1)
    Else

There are multiple base types. Therefore, send each type in the baseTypes collection to a function that will extract its name property.

        For Each oType in oSimple.baseTypes
Create a function to examine the \texttt{itemType} property of the \texttt{modelGroup} object. The function will then send each item from the \texttt{particles} collection to the appropriate function for examining the properties of the item.

Function processChoiceOrSequence(poGroup, t)
res = ""
For Each item in poGroup.particles
    If item.itemType = SOMITEM_ELEMENT then
        res = res + printElement(item, t+1)
    End If
    If (item.itemType and SOMITEM_GROUP) = SOMITEM_GROUP then
        res = res + processGroup(item, t+1)+vbNewline
    End If
    If item.itemType = SOMITEM_ANY then
        res = res + "any: " + item.name+vbNewline
End Function
End If
Next
processChoiceOrSequence = res
End Function

'---------------------------------------------------------------------

Create a function to walk through the properties of an element object.

Function printElement(oElement, t)
    res = printTab(t) + "<xsd:element "
    Check the isReference property of the element object to see if the element is a reference to a top-level element declaration.
    If oElement.isReference Then
        res = res + "ref='" + oElement.name + '"' + printParticles(o
        res = res + "<!-- "
    Check the isAbstract property of the element object to see if the element has had its abstract attribute set to true or to false.
    res = res + " abstract='" & oElement.isAbstract & """
    res = res + " -->"
    Else
    Get the type information for the element in a type object returned from the type property of the element object.
    Set oType=oElement.type
    Send the element object to a function that will extract its particle information from the inherited minOccurs and maxOccurs properties of the element object.
    res = res + "name='" + oElement.name + '"' + printParticles( res = res + " abstract='" & oElement.isAbstract & ""
    res = res + " id='" & oElement.id & '"'
    If oType.name = "" Then
        res = res + "">" + vbNewLine
    Check the itemType enumerated value of the type object to see what interface is needed to extract type information for the element's type.
    If oType.itemType = SOMITEM_COMPLEXTYPE Then
The type has been declared as a separate type declaration. The name property of the type object is not equal to "". 

Else
    If printName(oType) <> "xsd:anyType" Then
        res = res + " type='" + printName(oType) + "'
    End If

    If oType.itemType <> SOMITEM_COMPLEXTYPE Then
        If printRestrictions(oType, 0) = "" Then
            res = res + "/>
        Else
            res = res + "">
            vbNewLine + processSimpleType(oType, t)
            res = res + printTab(t) + "</xsd:element>
        End If
    Else
        res = res + "/>
    End If
End If
End If
End If
End Function

Use the scope property of the element object to find out the name of the scope used in the element declaration.

    rem = "scope:" & printName(oElement.scope)
    res = res + printRemark( "rem" )
    printElement = res
End Function

Create a function to get the particle information from the object that is passed in.

Function printParticles(oParticle)
    If oParticle.minOccurs <> 1 Then
        res = res + " minOccurs='" & oParticle.minOccurs & "'
    End If
    If oParticle.maxOccurs <> 1 Then
        If oParticle.maxOccurs = -1 Then
            res = res + " maxOccurs='unbounded'
        Else
            res = res + " maxOccurs='" + oParticle.maxOccurs & "'
        End If
    End If
Create a function to walk through the properties of an attribute object.

Function printAttr(oAttr, t)

Check the isReference property of the attribute object to see if the attribute is a reference to a top-level element declaration.

    If oAttr.isReference Then
        printAttr = printAttr + printTab(t) + "<xsd:attribute ref='" + oAttr.name + "'
    Else
        printAttr = printAttr + printTab(t) + "<xsd:attribute name='" + oAttr.name + "'
    End If

Check to see whether the type used for the attribute is declared in the attribute declaration, or declared separately. The code performs this check by looking at the name property of the type property for the attribute object.

    If oAttr.type.name <> "" Then
        printAttr = printAttr + " type='" + printName(oAttr.type) + "'
    End If

Check the defaultValue property of the attribute object to see if the defaultValue attribute was defined in the declaration of the attribute.

    If oAttr.defaultValue <> "" Then
        printAttr = printAttr + " default='" + oAttr.defaultValue + "'
    End If

Check the fixedValue property of the attribute object to see if the fixedValue attribute was defined in the declaration of the attribute.

    If oAttr.fixedValue <> "" Then
        printAttr = printAttr + " fixed='" + oAttr.fixedValue + "'
    End If

Check the use property of the attribute object to see the restrictions placed on the entry of the attribute into an XML Schema instance document.
If oAttr.use = SCHEMAUSE_OPTIONAL Then printAttr = printAttr + "use='optional'"
If oAttr.use = SCHEMAUSE_PROHIBITED Then printAttr = printAttr + "use='prohibited'"
If oAttr.use = SCHEMAUSE_REQUIRED Then printAttr = printAttr + "use='required'"
printAttr = printAttr + "/>
rem = "scope:" & printName(oElement.scope)
printAttr = printAttr + printRemark("rem")
End Function

'---------------------------------------------------------------------

This function is used to format the output information in a tab structure. No
SOM functionality is included in this function.

Function printTab(t)
    tab=""
    For x=0 to t
        tab=tab+"    "
    Next
    printTab=tab
End Function

'---------------------------------------------------------------------

Create a function to check the type property of an item and get the name of the
item. The function uses the name property to return the name. The name property
is inherited from the ISchemaItem interface.

Function printName(item)
    printName ="
    If (item.itemType and SOMITEM_DATATYPE) = SOMITEM_DATATYPE Then
        printName = "xsd:"
    End If
    If item.itemType = SOMITEM_ANYTYPE Then
        printName = "xsd:"
    End If
    printName= printName + item.name
End Function

'---------------------------------------------------------------------

Create a function to get the restriction information from the type object or the
complexType object that is passed to it. Each restriction that returns a string is
checked for a NULL value. If the value is not NULL, the restriction value is
retrieved from the appropriate property. Each restriction that returns an integer is
checked for a value of "–1". This value indicates that the restriction is not used.
If the value is greater than "–1", the restriction value is retrieved for the
appropriate property. All other restrictions used in this function have comments listed below.

Function printRestrictions(oType, t)
res = ""
If oType.minExclusive <> "" Then
    res = res + printTab(t+1) + "<xsd:minExclusive value='" + oType.minExclusive + '"/>
End If
If oType.minInclusive <> "" Then
    res = res + printTab(t+1) + "<xsd:minInclusive value='" + oType.minInclusive + '"/>
End If
If oType.maxExclusive <> "" Then
    res = res + printTab(t+1) + "<xsd:maxExclusive value='" + oType.maxExclusive + '"/>
End If
If oType.maxInclusive <> "" Then
    res = res + printTab(t+1) + "<xsd:maxInclusive value='" + oType.maxInclusive + '"/>
End If
If oType.totalDigits > -1 Then
    res = res + printTab(t+1) + "<xsd:totalDigits value='" + oType.totalDigits + '"/>
End If
If oType.fractionDigits > -1 Then
    res = res + printTab(t+1) + "<xsd:fractionDigits value='" + oType.fractionDigits + '"/>
End If
If oType.length > -1 Then
    res = res + printTab(t+1) + "<xsd:length value='" + oType.length + '"/>
End If
If oType.minLength > -1 Then
    res = res + printTab(t+1) + "<xsd:minLength value='" + oType.minLength + '"/>
End If
If oType.maxLength > -1 Then
    res = res + printTab(t+1) + "<xsd:maxLength value='" + oType.maxLength + '"/>
End If

Check the length of the enumeration property. If the length is greater than zero, get the value of the enumeration from the collection of enumerations.

    If oType.enumeration.length > 0 Then
        For Each item in oType.enumeration
            res = res + printTab(t+1) + "<xsd:enumeration value='" + item + '"/>
        Next
    End If

Check the value of the whitespace property. If the value is greater than zero, get the whitespace value of the type object.

    If oType.whitespace > 0 Then
        res = res + printTab(t+1) + "<xsd:whitespace value='" + oType.whitespace + '"" & oTy
End If
If oType.patterns.length <> 0 Then
    For Each oPattern in oType.patterns
        res = res + printTab(t+1) + "<xsd:pattern value='" + oPattern + vbNewLine
    Next
End If

printRestrictions = ""
If res <> "" and oType.baseTypes.length > 0 Then
    printRestrictions = printRestrictions + printTab(t) + "<xsd:restriction base='"
    printRestrictions = printRestrictions + printName(oType.baseTypes(0))
    printRestrictions = printRestrictions + res
    printRestrictions = printRestrictions + printTab(t) + "</xsd:restriction>"
End If

End Function
'
----------------------------------------------------------

This function wraps the value that is passed in the parameter in remark tags.

No SOM functionality is included in this function.

Function printRemark(r)
    If remarks = 1 Then
        printRemark = "<!-- " + r + " -->"
    End If
    printRemark = printRemark + vbNewLine
End Function

See Also

ISchema Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel.
Uncommented Code for the Walk the SOM Application

SOMITEM_SCHEMA = 4*1024
SOMITEM_ATTRIBUTE = SOMITEM_SCHEMA + 1
SOMITEM_ATTRIBUTEGROUP = SOMITEM_SCHEMA + 2
SOMITEM_NOTATION = SOMITEM_SCHEMA + 3

SOMITEM_ANYTYPE = 8*1024
SOMITEM_DATATYPE = SOMITEM_ANYTYPE+256
SOMITEM_SIMPLETYPE = SOMITEM_DATATYPE+256
SOMITEM_COMPLEXTYPE = 9*1024
SOMITEM_PARTICLE = 16*1024
SOMITEM_ANY = SOMITEM_PARTICLE+1
SOMITEM_ANYATTRIBUTE = SOMITEM_PARTICLE+2
SOMITEM_ELEMENT = SOMITEM_PARTICLE+3
SOMITEM_GROUP = SOMITEM_PARTICLE+256

SOMITEM_ALL = SOMITEM_GROUP+1
SOMITEM_CHOICE = SOMITEM_GROUP+2
SOMITEM_SEQUENCE = SOMITEM_GROUP+3
SOMITEM_EMPTYPARTICLE = SOMITEM_GROUP+4

' Attribute uses
SCHEMAUSE_OPTIONAL = 0
SCHEMAUSE_PROHIBITED = 1
SCHEMAUSE_REQUIRED = 2

SCHEMACONTENTTYPE_EMPTY = 0
SCHEMACONTENTTYPE_TEXTONLY = 1
SCHEMACONTENTTYPE_ELEMENTONLY = 2
SCHEMACONTENTTYPE_MIXED = 3

remarks = 0

Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")

' Load the schema.
nsTarget="http://www.example.microsoft.com/po"

oSchemaCache.add nsTarget, "po.xsd"
Set oSchema = oSchemaCache.getSchema(nsTarget)
result = "<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>"
For Each oE in oSchema.elements
    result = result + printElement(oE, 0)
Next
For Each oA in oSchema.attributes
    result = result + printAttr(oA, t)
Next
result = result + vbCrLf
For Each oT in oSchema.types
    result = result + processType(oT, 0)
Next
result = result + "</xsd:schema>"
WScript.Echo result
'
Function processType(oType, t)
    ' res = printTab(t) + printRemark(oType.name) + vbCrLf
    If oType.itemType = SOMITEM_ANYTYPE Then
        res = res + printTab(t+1) + "<!-- " + oType.name + " -->"
    End If
    If oType.itemType = SOMITEM_COMPLEXTYPE Then
        res = res + processComplexType(oType, t+1)
    End If
    If oType.itemType = SOMITEM_SIMPLETYPE Then
        res = res + processSimpleType(oType, t+1)
    End If
    processType = res + vbCrLf
End Function
Function processComplexType(oComplex, t)
    res = printTab(t) + "<xsd:complexType"
    If oComplex.name <> "" Then
        res = res + " name='" + oComplex.name + "'
    End If
    res = res + ">
    If oComplex.contentType = SCHEMACONTENTTYPE_EMPTY Then
        res = res + printRemark("empty")
    End If
    If oComplex.contentType = SCHEMACONTENTTYPE_TEXTONLY Then
        res = res + printRemark("textonly")
    End If
    If oComplex.contentType = SCHEMACONTENTTYPE_ELEMENTONLY Then
        res = res + printRemark("elementonly")
    End If
End Function
res = res + processGroup(oComplex.contentModel, t+1)
End If
If oComplex.contentType = SCHEMACONTENTTYPE_MIXED Then
  res = res + printRemark("mixed")
  res = res + processGroup(oComplex.contentModel, t+1)
End If
res = res + vbCrLf
res = res + printRestrictions(oComplex, t+1)

On Error Resume Next
Set any = oComplex.anyAttribute.name
If Err.number = 0 Then
  res = res + oComplex.anyAttribute.name
End If
For Each oAttr in oComplex.attributes
  res = res + printAttr(oAttr, t+1)
Next

processComplexType = res + printTab(t) + "</xsd:complexType>"+ vbCrLf
End Function

Function processSimpleType(oSimple, t)
res = printTab(t) + "<xsd:simpleType"
If oSimple.name <> "" Then
  res = res + " name='" + oSimple.name +"'
End If
res = res + "">"+ vbCrLf
If oSimple.baseTypes.length = 1 Then
  res = res + printRestrictions(oSimple, t+1)
Else
  For Each oType in oSimple.baseTypes
    res = res + "<baseType name='" + printName(oType) +"'>"+
  Next
End If

processSimpleType = res + printTab(t) + "</xsd:simpleType>"+ vbCrLf
End Function

Function processGroup(poGroup, t)
res = ""
' List elements in the sequence.
If poGroup.itemType = SOMITEM_ALL Then
  res = res + printTab(t+1) + "<xsd:all>"+ vbCrLf
  res = res + processChoiceOrSequence(poGroup, t+1)
  res = res + printTab(t+1) + "</xsd:all>"
End If
If poGroup.itemType = SOMITEM_CHOICE Then
    res = res + printTab(t+1) + "<xsd:choice>"+vbNewline
    res = res + processChoiceOrSequence(poGroup, t+1)
    res = res + printTab(t+1) + "</xsd:choice>
End If

If poGroup.itemType = SOMITEM_SEQUENCE Then
    res = res + printTab(t+1) + "<xsd:sequence>"+vbNewline
    res = res + processChoiceOrSequence(poGroup, t+1)
    res = res + printTab(t+1) + "</xsd:sequence>"
End If

processGroup = res
End Function

Function processChoiceOrSequence(poGroup, t)
    res = ""
    For Each item in poGroup.particles
        If item.itemType = SOMITEM_ELEMENT Then
            res = res + printElement(item, t+1)
        End If
        If (item.itemType and SOMITEM_GROUP) = SOMITEM_GROUP Then
            res = res + processGroup(item, t+1)+vbNewline
        End If
        If item.itemType = SOMITEM_ANY Then
            res = res + "any: " + item.name+vbNewline
        End If
    Next
    processChoiceOrSequence = res
End Function

Function printElement(oElement, t)
    res = printTab(t) + "<xsd:element 
    If oElement.isReference Then
        res = res + "ref='" + oElement.name + "'" + printParticles(o
        res = res + "<!-" 
        res = res + " abstract='" & oElement.isAbstract & "'
        res = res + " -->"
    Else
        Set oType=oElement.type
        res = res + "name='" + oElement.name + "'" + printParticles( res = res + " abstract='" & oElement.isAbstract & "'
        res = res + " id='" & oElement.id & "" 
        If oType.name = "" Then
            res = res + "" + vbNewLine
            If oType.itemType = SOMITEM_COMPLEXTYPE Then
                res = res + printElement + processComplexType(oType, 
            Else
                res = res + processSimpleType(oType, t)
        End If
    End If
res = res + printTab(t) + "</xsd:element>"
Else
    If printName(oType) <> "xsd:anyType" Then
        res = res + " type=" + printName(oType) + ""
    End If
End If

If oType.itemType <> SOMITEM_COMPLEXTYPE Then
    If printRestrictions(oType, 0) = "" Then
        res = res + "/"
        Else
        res = res + vbCrLf + vbCrLf + processSimpleType(oType)
        res = res + printTab(t) + "</xsd:element>"
    End If
Else
    res = res + vbCrLf
End If
End If
End If
rem = "scope:" & printName(oElement.scope)
res = res + printRemark("rem")
printElement = res
End Function

Function printParticles(oParticle)
    If oParticle.minOccurs <> 1 Then
        res = res + " minOccurs=" & oParticle.minOccurs & ""
    End If
    If oParticle.maxOccurs <> 1 Then
        If oParticle.maxOccurs = -1 Then
            res = res + " maxOccurs='unbounded'"
        Else
            res = res + " maxOccurs='' & oParticle.maxOccurs & ""
        End If
    End If
End Function
printParticles = res
End Function

Function printAttr(oAttr, t)
    If oAttr.isReference Then
        printAttr = printAttr + printTab(t) + "<xsd:attribute re
    Else
        printAttr = printAttr + printTab(t) + "<xsd:attribute na
    End If
    If oAttr.type.name <> "" Then
        printAttr = printAttr + " type='" + printName(oAttr.type
    End If
    If oAttr.defaultValue <> "" Then
        printAttr = printAttr + " default='" + oAttr.defaultValue
    End If

If oAttr.fixedValue <> "" Then
    printAttr = printAttr + " fixed='' + oAttr.fixedValue + "" + vbCrLf
End If
If oAttr.use = SCHEMAUSE_OPTIONAL Then printAttr = printAttr + " use='optional'" + vbCrLf
If oAttr.use = SCHEMAUSE_PROHIBITED Then printAttr = printAttr + vbCrLf
    rem = "scope:" & itemName(oElement.scope)
End If
printAttr = printAttr + vbCrLf
End Function

Function printTab(t)
    tab = ""
    for x=0 to t
        tab = tab + " " + vbCrLf
    next
    printTab = tab
End Function

Function itemName(item)
    itemName = ""
    If (item.itemType and SOMITEM_DATATYPE) = SOMITEM_DATATYPE Then
        itemName = "xsd:" + vbCrLf
    End If
    If item.itemType = SOMITEM_ANYTYPE Then
        itemName = "xsd:" + vbCrLf
    End If
    itemName = itemName + item.name + vbCrLf
End Function

Function printRestrictions(oType, t)
    res = ""
    If oType.minExclusive <> "" Then
        res = res + printTab(t+1) + vbCrLf + "<xsd:minExclusive value='" + vbCrLf + oType.minExclusive + vbCrLf + vbCrLf + vbCrLf + vbCrLf + vbCrLf
    End If
    If oType.minInclusive <> "" Then
        res = res + printTab(t+1) + vbCrLf + "<xsd:minInclusive value='" + vbCrLf + oType.minInclusive + vbCrLf + vbCrLf + vbCrLf + vbCrLf + vbCrLf
    End If
    If oType.maxExclusive <> "" Then
        res = res + printTab(t+1) + vbCrLf + "<xsd:maxExclusive value='" + vbCrLf + oType.maxExclusive + vbCrLf + vbCrLf + vbCrLf + vbCrLf + vbCrLf
    End If
    If oType.maxInclusive <> "" Then
        res = res + printTab(t+1) + vbCrLf + "<xsd:maxInclusive value='" + vbCrLf + oType.maxInclusive + vbCrLf + vbCrLf + vbCrLf + vbCrLf + vbCrLf
    End If
    If oType.totalDigits > -1 Then
        res = res + printTab(t+1) + vbCrLf + "<xsd:totalDigits value='" + vbCrLf + oType.totalDigits + vbCrLf + vbCrLf + vbCrLf + vbCrLf + vbCrLf
    End If
    If oType.fractionDigits > -1 Then
        res = res + printTab(t+1) + vbCrLf + "<xsd:fractionDigits value='" + vbCrLf + oType.fractionDigits + vbCrLf + vbCrLf + vbCrLf + vbCrLf + vbCrLf
    End If
If oType.length > -1 Then
    res = res + printTab(t+1) + "<xsd:length value='" & oType.length & '"/>"
    vbNewLine
End If
If oType.minLength > -1 Then
    res = res + printTab(t+1) + "<xsd:minLength value='" & oType.minLength & '"/>"
    vbNewLine
End If
If oType.maxLength > -1 Then
    res = res + printTab(t+1) + "<xsd:maxLength value='" & oType.maxLength & '"/>"
    vbNewLine
End If
If oType.enumeration.length > 0 Then
    For Each item in oType.enumeration
        res = res + printTab(t+1) + "<xsd:enumeration value='" + item + '"/>"
        vbNewLine
    Next
End If
If oType.whitespace > 0 Then
    res = res + printTab(t+1) + "<xsd:whitespace value='" & oType.whitespace & '"/>"
    vbNewLine
End If
If oType.patterns.length <> 0 Then
    For Each oPattern in oType.patterns
        res = res + printTab(t+1) + "<xsd:pattern value='" + oPattern + '"/>"
        vbNewLine
    Next
End If
printRestrictions = ""
If res <> "" and oType.baseTypes.length > 0 Then
    printRestrictions = printRestrictions + printTab(t) + "<xsd:restriction base='"
    printRestrictions = printRestrictions + res
    printRestrictions = printRestrictions + printTab(t) + "</xsd:restriction>"
End If

End Function

Function printRemark(r)
    If remarks = 1 Then
        printRemark = "<!-- " + r + " -->
    End If
    printRemark = printRemark + vbNewLine
End Function
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel
Locate Declarations

This sample application goes through an XML Schema and prints selected declarations in a tab-structured format. The DOM schema cache is used to store the XML Schema. The schema object is obtained by the `getSchema` method of the `IXMLDOMSchemaCollection2` interface. The interfaces of the SOM are used to retrieve information about the declarations used in this sample. Formatting functions and methods are present throughout the application, but are used solely for text output and can be ignored for functionality of the SOM interfaces.

This topic is divided into the following sections.

- Instructions for Running the Locate Declarations Application
- The Locate Declarations Application
- Uncommented Code for the Locate Declarations Application

See Also

`IXMLDOMSchemaCollection/XMLSchemaCache`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel
Instructions for Running the Locate Declarations Application

Note The following sample assumes that you are using a computer with Windows Scripting Host (WSH) installed, such as a computer running Windows 2000 Professional or Windows 98.

To run the application

1. Open Notepad.
2. Copy the XML document, po2.xml. Paste it into the Notepad window.
3. From the File menu, click Save As. Save the file as po2.xml to a folder on your computer.
4. From the File menu, click New.
6. From the File menu, click Save As. Save the file as po2.xsd to the same folder in which you saved po2.xml.
7. From the File menu, click New.
9. From the File menu, click Save As. Save the file as po2.vbs to the same folder in which you saved po2.xml and po2.xsd.
10. Run po2.vbs from either the Command Prompt or Windows Explorer.

The results will be displayed in a message box window.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel...
# The Locate Declarations Application

This section contains the following:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>po2.xml</td>
<td>The XML document.</td>
</tr>
<tr>
<td>po2.xsd</td>
<td>The XML Schema document.</td>
</tr>
<tr>
<td>po2.vbs Step-by-Step</td>
<td>The VBScript application, explained step-by-step with textual comments.</td>
</tr>
<tr>
<td></td>
<td>This topic walks you through the application code.</td>
</tr>
</tbody>
</table>
The Locate Declarations application uses the following XML document.

```xml
<po:purchaseOrder orderDate="1999-10-20" xmlns:po="http://www.example.microsoft.com/po">
  <shipTo country="US">
    <name>Kim Yoshida</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Neil Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <po:comment>Hurry, my lawn is going wild!</po:comment>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <po:comment>Confirm that this is electric.</po:comment>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</po:purchaseOrder>
```
The Locate Declarations application uses the following XML Schema document.

```xml
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    xmlns:po='http://www.example.microsoft.com/po'
    targetNamespace='http://www.example.microsoft.com/po'>

    <xsd:annotation>
        <xsd:documentation>
            Copyright 2000 Example.Microsoft.com. All rights reserved.
        </xsd:documentation>
    </xsd:annotation>

    <xsd:element name='comment' type='xsd:string'/>

    <xsd:element name='purchaseOrder' type='po:PurchaseOrderType'/>

    <xsd:complexType name='USAddress'>
        <xsd:sequence>
            <xsd:element name='name' type='xsd:string'/>
            <xsd:element name='street' type='xsd:string'/>
            <xsd:element name='city' type='xsd:string'/>
            <xsd:element name='state' type='xsd:string'/>
            <xsd:element name='zip' type='xsd:decimal'/>
        </xsd:sequence>
        <xsd:attribute name='country' type='xsd:NMTOKEN' fixed='US'/>
    </xsd:complexType>

    <xsd:simpleType name='SKU'>
        <xsd:restriction base='xsd:string'>
            <xsd:pattern value='\d{3}-[A-Z]{2}'/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name='Items'>
        <xsd:sequence>
            <xsd:element name='item' minOccurs='0' maxOccurs='unbounded'>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name='productName' type='xsd:string'/>
                        <xsd:element name='quantity'>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```
<xsd:simpleType>
  <xsd:restriction base='xsd:positiveInteger'>
    <xsd:minInclusive value='1'/>
    <xsd:maxExclusive value='100'/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name='USPrice' type='xsd:decimal'/>
<xsd:element ref='po:comment'/>
<xsd:element name='shipDate' type='xsd:date' minOccurs='0'/>
</xsd:sequence>
<xsd:attribute name='partNum' type='po:SKU'/>
</xsd:complexType>

<xsd:complexType name='PurchaseOrderType'>
  <xsd:sequence>
    <xsd:element name='shipTo' type='po:USAddress'/>
    <xsd:element name='billTo' type='po:USAddress'/>
    <xsd:element ref='po:comment' minOccurs='0'/>
    <xsd:element name='items' type='po:Items'/>
  </xsd:sequence>
  <xsd:attribute name='orderDate' type='xsd:date'/>
  <xsd:attribute name='confirmDate' type='xsd:date' use='required'/>
</xsd:complexType>
</xsd:schema>
**po2.vbs Step-by-Step**

This topic walks you through the Locate Declarations application. The code is interspersed with textual comments that describe each step of the application.

Click here for the [Uncommented Code for the Locate Declarations Application](#).
The Application

The code begins with some constant declarations.

' Item types:
SOMITEM_SCHEMA = 4*1024
SOMITEM_ATTRIBUTE = SOMITEM_SCHEMA + 1
SOMITEM_ATTRIBUGROUP = SOMITEM_SCHEMA + 2
SOMITEM_NOTATION = SOMITEM_SCHEMA + 3
SOMITEM_ANYTYPE = 8*1024
SOMITEM_DATATYPE = SOMITEM_ANYTYPE+256
SOMITEM_SIMPLETYPE = SOMITEM_DATATYPE+256
SOMITEM_COMPLEXTYPE = 9*1024
SOMITEM_PARTICLE = 16*1024
SOMITEM_ANY = SOMITEM_PARTICLE+1
SOMITEM_ANYATTRIBUTE = SOMITEM_PARTICLE+2
SOMITEM_ELEMENT = SOMITEM_PARTICLE+3
SOMITEM_GROUP = SOMITEM_PARTICLE+256
SOMITEM_ALL = SOMITEM_GROUP+1
SOMITEM_CHOICE = SOMITEM_GROUP+2
SOMITEM_SEQUENCE = SOMITEM_GROUP+3
SOMITEM_EMPTYPARTICLE = SOMITEM_GROUP+4
SCHEMAUSE_OPTIONAL = 0
SCHEMAUSE_PROHIBITED = 1
SCHEMAUSE_REQUIRED = 2
SCHEMACONTENTTYPE_EMPTY = 0
SCHEMACONTENTTYPE_TEXTONLY = 1
SCHEMACONTENTTYPE_ELEMENTONLY = 2
SCHEMACONTENTTYPE_MIXED = 3
remarks = 0

Create a schema cache object. This object will be used later to contain the XML Schema document, po2.xsd.

set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")

Add the XML Schema document to the schema cache, using its add method. A SOM schema object is returned. The SOM interfaces will now be used to explore the schema object.
The following block of code sets properties on a DOMDocument. For more information about DOMDocuments, see the IXMLDOMDocument/DOMDocument interface, and the setProperty method of DOMDocument2.

Set oDoc = CreateObject("Msxml2.DOMDocument.5.0")
oDoc.async = false
oDoc.validateOnParse = false

The schema cache is stored in the schemas property of the DomDocument.

set oDoc.schemas = oSchemaCache
oDoc.load "po2.xml"
oDoc.setProperty "SelectionLanguage", "XPath"
oDoc.setProperty "SelectionNamespaces", "xmlns:po='http://www.example.microsoft.com/po'"

For the next three groups of code, set the node object oNo to the DOM node for the specified section of the XML document.

Set oNo = oDoc.selectSingleNode("//po:purchaseOrder/shipTo/@country"
result = result + printDecl(oNo) + vbNewLine

Set oNo = oDoc.selectSingleNode("//po:purchaseOrder/items/item/quantity"
result = result + printDecl(oNo) + vbNewLine

Set oNo = oDoc.selectSingleNode("//po:purchaseOrder/items/item/@part"
result = result + printDecl(oNo) + vbNewLine

WScript.Echo result
'

Create a function to create a declaration object. This function will examine the itemType to determine whether the declaration is an element or an attribute.

Function printDecl(oNode)

Use the getDeclaration method that is implemented by the namespaces method of the DOMDocument2 object to retrieve a declaration object for the node object that is passed to this function.

Set oDecl = oDoc.namespaces.getDeclaration(oNode)
Check the **itemType** of the declaration object. If the type is an element, pass the declaration object to a function that will examine the properties of the object.

    If oDecl.itemType = SOMITEM_ELEMENT Then
        printDecl = printElement(oDecl, 1)
    End If

If the type is an attribute, send the declaration object to a function that will examine the properties of the **attribute** object.

    If oDecl.itemType = SOMITEM_ATTRIBUTE Then
        printDecl = printAttr(oDecl, 1)
    End If

Send a **type** object to a function that will extract its properties. The type object is the type specified in the type attribute of the current attribute.

    printDecl = printDecl + processType(oDecl.Type, 1)
End If
End Function
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel
Uncommented Code for the Locate Declarations Application

' Item types:
SOMITEM_SCHEMA = 4*1024
SOMITEM_ATTRIBUTE = SOMITEM_SCHEMA + 1
SOMITEM_ATTRIBUTEGROUP = SOMITEM_SCHEMA + 2
SOMITEM_NOTATION = SOMITEM_SCHEMA + 3

SOMITEM_ANYTYPE = 8*1024
SOMITEM_DATATYPE = SOMITEM_ANYTYPE+256
SOMITEM_SIMPLETYPE = SOMITEM_DATATYPE+256
SOMITEM_COMPLEXTYPE = 9*1024

SOMITEM_PARTICLE = 16*1024
SOMITEM_ANY = SOMITEM_PARTICLE+1
SOMITEM_ANYATTRIBUTE = SOMITEM_PARTICLE+2
SOMITEM_ELEMENT = SOMITEM_PARTICLE+3
SOMITEM_GROUP = SOMITEM_PARTICLE+256

SOMITEM_ALL = SOMITEM_GROUP+1
SOMITEM_CHOICE = SOMITEM_GROUP+2
SOMITEM_SEQUENCE = SOMITEM_GROUP+3
SOMITEM_EMPTYPARTICLE = SOMITEM_GROUP+4

SCHEMAUSE_OPTIONAL = 0
SCHEMAUSE_PROHIBITED = 1
SCHEMAUSE_REQUIRED = 2

SCHEMACONTENTTYPE_EMPTY = 0
SCHEMACONTENTTYPE_TEXTONLY = 1
SCHEMACONTENTTYPE_ELEMENTONLY = 2
SCHEMACONTENTTYPE_MIXED = 3

remarks = 0

Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
nsTarget="http://www.example.microsoft.com/po"
oSchemaCache.add nsTarget, "po2.xsd"

Set oDoc = CreateObject("Msxml2.DOMDocument.5.0")
oDoc.async = false
oDoc.validateOnParse = false
set oDoc.schemas = oSchemaCache
oDoc.load "po2.xml"
oDoc.setProperty "SelectionLanguage", "XPath"
oDoc.setProperty "SelectionNamespaces", "xmlns:po='http://www.example.microsoft.com/po'"
oDoc.setProperty "SelectionNamespaces", "xmlns:po='http://www.example.microsoft.com/po'"

result = ""
Set oNo = oDoc.selectSingleNode("//po:purchaseOrder/shipTo/@country"
result = result + printDecl(oNo) + vbNewLine

Set oNo = oDoc.selectSingleNode("//po:purchaseOrder/items/item/quantity"
result = result + printDecl(oNo) + vbNewLine

Set oNo = oDoc.selectSingleNode("//po:purchaseOrder/items/item/@part"
result = result + printDecl(oNo) + vbNewLine

WScript.Echo result

Function printDecl(oNode)
    Set oDecl = oDoc.namespaces.getDeclaration(oNode)
    If oDecl.itemType = SOMITEM_ELEMENT Then
        printDecl = printElement(oDecl, 1)
    End If
    If oDecl.itemType = SOMITEM_ATTRIBUTE Then
        printDecl = printAttr(oDecl, 1)
        printDecl = printDecl + processType(oDecl.Type, 1)
    End If
End Function

Function processType(oType, t)
    If oType.itemType = SOMITEM_ANYTYPE Then
        res = res + printTab(t+1) + "<!-- " + oType.name + " -->"
    End If
    If oType.itemType = SOMITEM_COMPLEXTYPE Then
        res = res + processComplexType(oType, t+1)
    End If
    If oType.itemType = SOMITEM_SIMPLETYPE Then
        res = res + processSimpleType(oType, t+1)
    End If
    processType = res + vbNewLine
End Function

Function processComplexType(oComplex, t)
    res = printTab(t) + '<xsd:complexType'
    If oComplex.name <> "" Then
        res = res + ' name=""' + oComplex.name + '"
    End If
    res = res + '>'
    If oComplex.contentType = SCHEMACONTENTTYPE_EMPTY Then
        res = res + printRemark("empty")
End If
If oComplex.contentType = SCHEMACONTENTTYPE_TEXTONLY Then
    res = res + printRemark("textonly")
End If
If oComplex.contentType = SCHEMACONTENTTYPE_ELEMENTONLY Then
    res = res + printRemark("elementonly")
    res = res + processGroup(oComplex.contentModel, t+1)
End If
If oComplex.contentType = SCHEMACONTENTTYPE_MIXED Then
    res = res + printRemark("mixed")
    res = res + processGroup(oComplex.contentModel, t+1)
End If
res = res + vbNewline
res = res + processRestrictions(oComplex, t+1)

On Error Resume Next
Set any = oComplex.anyAttribute.name
If Err.number = 0 Then
    res = res + oComplex.anyAttribute.name
End If

For Each oAttr in oComplex.attributes
    res = res + printAttr(oAttr, t+1)
Next

processComplexType = res + printTab(t) + "</xsd:complexType>"+vbNewline
End Function

Function processSimpleType(oSimple, t)
res = printTab(t) + "<xsd:simpleType"
If oSimple.name <> "" Then
    res = res + " name='" + oSimple.name +"'"
End If
res = res + "">"+vbNewline

If oSimple.baseTypes.length = 1 Then
    res = res + printRestrictions(oSimple, t+1)
Else
    For Each oType in oSimple.baseTypes
        res = res + "<baseType name='" + printName(oType) +"'>"+
    Next
End If

processSimpleType = res + printTab(t) + "</xsd:simpleType>"+vbNewline
End Function

Function processGroup(poGroup, t)
    res = ""
If poGroup.itemType = SOMITEM_ALL Then  
    res = res + printTab(t+1) + "<xsd:all>"+vbNewline  
    res = res + processChoiceOrSequence(poGroup, t+1)  
    res = res + printTab(t+1) + "</xsd:all>"
End If

If poGroup.itemType = SOMITEM_CHOICE Then  
    res = res + printTab(t+1) + "<xsd:choice>"+vbNewline  
    res = res + processChoiceOrSequence(poGroup, t+1)  
    res = res + printTab(t+1) + "</xsd:choice>"
End If

If poGroup.itemType = SOMITEM_SEQUENCE Then  
    res = res + printTab(t+1) + "<xsd:sequence>"+vbNewline  
    res = res + processChoiceOrSequence(poGroup, t+1)  
    res = res + printTab(t+1) + "</xsd:sequence>"
End If
processGroup = res
End Function

Function processChoiceOrSequence(poGroup, t)  
    res = ""
    For Each item in poGroup.particles  
        If item.itemType = SOMITEM_ELEMENT Then  
            res = res + printElement(item, t+1)  
        End If  
        If (item.itemType and SOMITEM_GROUP) = SOMITEM_GROUP Then  
            res = res + processGroup(item, t+1)+vbNewline  
        End If  
        If item.itemType = SOMITEM_ANY Then  
            res = res + "any: " + item.name+vbNewline  
        End If
    Next
    processChoiceOrSequence = res
End Function

Function printElement(oElement, t)  
    res = printTab(t) + "<xsd:element  
    If oElement.isReference Then  
        res = res + "ref='" + oElement.name + "'" + printParticles(o 
        res = res + "<!-:- "  
        res = res + " abstract='' & oElement.isAbstract & '''"  
        res = res + " - ->"
    Else  
        Set oType=oElement.type  
        res = res + "name='" + oElement.name + "'" + printParticles( 
        res = res + " abstract='' & oElement.isAbstract & '''"  
        res = res + " id='" & oElement.id & "'"  
        If oType.name = "" Then  
            res = res + "">
    End Function
If oType.itemType = SOMITEM_COMPLEXTYPE Then
    res = res + printElement + processComplexType(oType,
Else
    res = res + processSimpleType(oType, t)
End If
res = res + printTab(t) + "</xsd:element>"
Else
    If printName(oType) <> "xsd:anyType" Then
        res = res + " type='" + printName(oType) + "'"
    End If
    If oType.itemType <> SOMITEM_COMPLEXTYPE Then
        If printRestrictions(oType, 0) = "" Then
            res = res + "/"
        Else
            res = res + "">" + vbNewLine + processSimpleType(
                res = res + printTab(t) + "</xsd:element>"
        End If
    Else
        res = res + "/""
    End If
End If
End If
rem = "scope:" & printName(oElement.scope)
res = res + printRemark( "rem" )
printElement = res
End Function

Function printParticles(oParticle)
    If oParticle.minOccurs <> 1 Then
        res = res + " minOccurs='" & oParticle.minOccurs & "'
    End If
    If oParticle.maxOccurs <> 1 Then
        If oParticle.maxOccurs = -1 Then
            res = res + " maxOccurs='unbounded'
        Else
            res = res + " maxOccurs='" & oParticle.maxOccurs & ""
        End If
    End If
End If
printParticles = res
End Function

Function printAttr(oAttr, t)
    If oAttr.isReference Then
        printAttr = printAttr + printTab(t) + "<xsd:attribute re
    Else
        printAttr = printAttr + printTab(t) + "<xsd:attribute na
End If
End Function
End If
If oAttr.type.name <> "" Then
printAttr = printAttr + " type='' + printName(oAttr.type)
End If
If oAttr.defaultValue <> "" Then
    printAttr = printAttr + " default='" + oAttr.defaultValue + 
End If
If oAttr.fixedValue <> "" Then
    printAttr = printAttr + " fixed='" + oAttr.fixedValue + 
End If
If oAttr.use = SCHEMAUSE_OPTIONAL Then printAttr = printAttr + 
If oAttr.use = SCHEMAUSE_PROHIBITED Then printAttr = printAttr + 
If oAttr.use = SCHEMAUSE_REQUIRED Then printAttr = printAttr + 
printAttr = printAttr + ">
rem = "scope:" & printName(oElement.scope)
printAttr = printAttr + printRemark("rem")
End Function

Function printTab(t)
    tab=""
    For x=0 to t
        tab=tab+" "
    Next
    printTab=tab
End Function

Function printName(item)
    printName ="
    If (item.itemType and SOMITEM_DATATYPE) = SOMITEM_DATATYPE Then
        printName= "xsd:"
    End If
    If item.itemType = SOMITEM_ANYTYPE Then
        printName= "xsd:"
    End If
    printName= printName + item.name
End Function

Function printRestrictions(oType, t)
    res = 
    If oType.minExclusive <> "" Then
        res = res + printTab(t+1) + "<xsd:minExclusive value=''"+ oTy
    End If
    If oType.minInclusive <> "" Then
        res = res + printTab(t+1) + "<xsd:minInclusive value=''"+ oTy
    End If
    If oType.maxExclusive <> "" Then
        res = res + printTab(t+1) + "<xsd:maxExclusive value=''"+ oTy
    End If
    If oType.maxInclusive <> "" Then
        res = res + printTab(t+1) + "<xsd:maxInclusive value=''"+ oTy
    End If
    If oType.totalDigits > -1 Then
        res = res + printTab(t+1) + "<xsd:totalDigits value=''"+ oTy
    End If

res = res + printTab(t+1) + "<xsd:totalDigits value='" & oType.totalDigits & 
End If
If oType.fractionDigits > -1 Then
    res = res + printTab(t+1) + "<xsd:fractionDigits value='" & oType.fractionDigits & 
End If
If oType.length > -1 Then
    res = res + printTab(t+1) + "<xsd:length value='" & oType.length & 
End If
If oType.minLength > -1 Then
    res = res + printTab(t+1) + "<xsd:minLength value='" & oType.minLength & 
End If
If oType.maxLength > -1 Then
    res = res + printTab(t+1) + "<xsd:maxLength value='" & oType.maxLength & 
End If
If oType.enumeration.length > 0 Then
    For each item in oType.enumeration
        res = res + printTab(t+1) + "<xsd:enumeration value='" + item + 
    Next
End If
If oType.whitespace > 0 Then
    res = res + printTab(t+1) + "<xsd:whitespace value='" & oType.whitespace & 
End If
If oType.patterns.length > 0 Then
    For Each pattern in oType.patterns
        res = res + printTab(t+1) + "<xsd:pattern value='" + pattern + 
    Next
End If

printRestrictions = ""
If res <> "" Then
    printRestrictions = printRestrictions + printTab(t) + "<xsd: 
    printRestrictions = printRestrictions + res
    printRestrictions = printRestrictions + printTab(t) + 
End If

End Function

Function printRemark(r)
    If remarks = 1 Then
        printRemark = "<!-- " + r + " -->
    End If
    printRemark = printRemark + vbNewLine
End Function
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel.
Write Annotations

This sample application writes annotations to all types of objects: IXMLDOMDocument, IXMLDOMNode, IVBSAXContentHandler, and ISAXContentHandler. First, the code instantiates a schema cache object, a DOM Document object, and a Sax writer object. The DOM Schema Cache is used to store the XML Schema. The schema object is obtained by the getSchema method of the IXMLDOMSchemaCollection2 interface. The writeAnnotation method of the ISchemaItem interface creates the annotations in the separate objects. The.documentElement and output properties echo the declaration text of the annotations that are created.

This topic is divided into the following sections.

- Instructions for Running the Write Annotations Application
- The Write Annotations Application
- Uncommented Code for the Write Annotations Application
Instructions for Running the Write Annotations Application

Note The following sample assumes you are using a computer with Windows Scripting Host (WSH) installed, such as a computer running Windows 2000 Professional or Windows 98.

1. Open Notepad.
3. From the File menu, click Save As. Save the file as po3.xsd to a folder on your computer.
4. From the File menu, click New.
5. Copy the Uncommented Code for the Write Annotations Application. Paste it into the Notepad window.
6. From the File menu, click Save As. Save the file as po3.vbs to the same folder in which you saved po3.xsd.

   The results will be displayed in a message box window.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel.
# The Write Annotations Application

This section contains the following:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>po3xsd</td>
<td>The XML Schema document. The VBScript application, explained step-by-step with textual comments. This topic walks you through the application code.</td>
</tr>
<tr>
<td>po3.vbs Step-by-Step</td>
<td></td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel
The Write Annotations application uses the following XML Schema document.

```xml
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
            xmlns:po = "http://www.example.microsoft.com/po"
            targetNamespace = "http://www.example.microsoft.com/po">

  <xsd:annotation>
    <xsd:documentation>
      Copyright 2001 Example.Microsoft.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:annotatio
<xsd:element name='state' type='xsd:string'/>
<xsd:element name='zip' type='xsd:decimal'/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
po3.vbs Step-by-Step

This topic walks you through the Locate Declarations application. The code is interspersed with textual comments that describe each step of the application.

Click here for the Uncommented Code for the Write Annotations Application.
The Application

The code begins by creating objects to write the annotations to. For more information about these objects, see **IXMLDOMSchemaCollection/XMLSchemaCache**, **IXMLDOMDocument/DOMDocument**, and **MXXMLWriter CoClass**.

```vba
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set writer = CreateObject("Msxml2.MXXMLWriter.5.0")
writer.output=""
```

Add the XML Schema document to the schema cache, using its add method. A SOM schema object is returned. The code now uses SOM interfaces to explore the schema object.

For more information about the schema cache, see the sections about the **IXMLDOMSchemaCollection** add and get methods.

```vba
nsTarget="http://www.example.microsoft.com/po"
oSchemaCache.add nsTarget, "po3.xsd"
Set oSchema = oSchemaCache.getSchema(nsTarget)
```

Write top-level XML Schema annotations into a DOM document.

```vba
oSchema.writeAnnotation(oAnnotationDoc)
WScript.Echo oAnnotationDoc.documentElement.xml
```

Write top-level XML Schema annotations into a DOM document node.

```vba
Set oAnnotationDoc.documentElement = oAnnotationDoc.createElement("root")
oSchema.writeAnnotation(oAnnotationDoc.documentElement)
WScript.Echo oAnnotationDoc.documentElement.xml
```

Write top-level XML Schema annotations into a SAX writer node.

```vba
oSchema.writeAnnotation(writer)
WScript.Echo writer.output
```

Write annotations for an XML Schema element into a DOM Document.

```vba
Set ua = oSchema.types.itemByName("USAddress")
```
Set oAnnotationDoc.documentElement = oAnnotationDoc.createElement("r")
ua.writeAnnotation(oAnnotationDoc)
WScript.Echo oAnnotationDoc.documentElement.xml

Write annotations for an XML Schema element into a DOM Document element.

ua.writeAnnotation(oAnnotationDoc.documentElement)
WScript.Echo oAnnotationDoc.documentElement.xml
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Devel.
Uncommented Code for the Write Annotations Application

Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")
Set writer = CreateObject("Msxml2.MXXMLWriter.5.0")
writer.output=""

nsTarget="http://www.example.microsoft.com/po"
oSchemaCache.add nsTarget, "po3.xsd"
Set oSchema = oSchemaCache.getSchema(nsTarget)

oSchema.writeAnnotation(oAnnotationDoc)
WScript.Echo oAnnotationDoc.documentElement.xml

Set oAnnotationDoc.documentElement = oAnnotationDoc.createElement("root")
oSchema.writeAnnotation(oAnnotationDoc.documentElement)
WScript.Echo oAnnotationDoc.documentElement.xml

oSchema.writeAnnotation(writer)
WScript.Echo writer.output

Set ua = oSchema.types.itemByName("USAddress")
Set oAnnotationDoc.documentElement = oAnnotationDoc.createElement("r")
ua.writeAnnotation(oAnnotationDoc)
WScript.Echo oAnnotationDoc.documentElement.xml

ua.writeAnnotation(oAnnotationDoc.documentElement)
WScript.Echo oAnnotationDoc.documentElement.xml
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
SOM Reference

The Schema Object Model (SOM) is based on the W3C recommendation document, "XML Schema Part 0: Primer." This document is located at http://www.w3.org/TR/xmlschema-0/.

The interfaces of the SOM provide information about schemas that are retrieved from a schema cache. This cache is accessed through the getSchema method of the IXMLDOMSchemaCollection2 interface.

The SOM provides access to items in a schema. Many of the interfaces inherit from other interfaces. The following graphic shows interfaces to the right of the interfaces they inherit from. For example, ISchemaElement inherits from ISchemaParticle, which inherits from ISchemaItem.

You can use the graphic to view the reference material for an interface by clicking on the interface name.

Inheritance Map

SOM Reference contains the following topics.

- XML SOM Enumerated Constants
- ISchema Interface
- ISchemaAny Interface
- ISchemaAttribute Interface
- ISchemaAttributeGroup Interface
- ISchemaComplexType Interface
- ISchemaElement Interface
- ISchemaIdentityConstraint Interface
- ISchemaItem Interface
- ISchemaItemCollection Interface
- ISchemaModelGroup Interface
- ISchemaNotation Interface
- ISchemaParticle Interface
- ISchemaStringCollection Interface
- ISchemaType Interface

See Also

SOM Developer's Guide
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
XML SOM Enumerated Constants

All enumerated values are found in msxml2.idl or msxml5.dll type lib. There are six enumerations, provided in the following tables:

- **SOMITEMTYPE Enumerations**
- **SCHEMASURE Enumerations**
- **SCHEMACONSTRUCTIONMETHOD Enumerations**
- **SCHEMACONTENTTYPE Enumeration**
- **SCHEMAPROCESSCONTENTS Enumerations**
- **SCHEMADWHTSPACE Enumerations**
- **SCHEMATYPEVARIETY Enumerations**

**SOMITEMTYPE Enumerations**

Used by the `itemType` property of the `ISchemaItem` interface.

<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
<th>HEX Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM_SCHEMA</td>
<td>4096</td>
<td>0x1000</td>
</tr>
<tr>
<td>SOMITEM_ATTRIBUTE</td>
<td>4097</td>
<td>0x1001</td>
</tr>
<tr>
<td>SOMITEM_ATTRIBUTEGROUP</td>
<td>4098</td>
<td>0x1002</td>
</tr>
<tr>
<td>SOMITEM_NOTATION</td>
<td>4099</td>
<td>0x1003</td>
</tr>
<tr>
<td>SOMITEM_IDENTITYCONSTRAINT</td>
<td>4352</td>
<td>0x1100</td>
</tr>
<tr>
<td>SOMITEM_KEY</td>
<td>4353</td>
<td>0x1101</td>
</tr>
<tr>
<td>SOMITEM_KEYREF</td>
<td>4354</td>
<td>0x1102</td>
</tr>
<tr>
<td>SOMITEM_UNIQUE</td>
<td>4355</td>
<td>0x1103</td>
</tr>
<tr>
<td>SOMITEM_ANYTYPE</td>
<td>8192</td>
<td>0x2000// also type mask</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE</td>
<td>8448</td>
<td>0x2100// built-in type mask</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ANYTYPE</td>
<td>8449</td>
<td>0x2101</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ANYURI</td>
<td>8450</td>
<td>0x2102</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_BASE64BINARY</td>
<td>8451</td>
<td>0x2103</td>
</tr>
</tbody>
</table>
SCHEMASE USE Enumerations

Used by the `use` property of the `ISchemaAttribute` interface.

<table>
<thead>
<tr>
<th>Enumerated Type</th>
<th>Value</th>
<th>DEC Value</th>
<th>HEX Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM_DATATYPE_STRING</td>
<td>8485</td>
<td>0x2125</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_TIME</td>
<td>8486</td>
<td>0x2126</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_TOKEN</td>
<td>8487</td>
<td>0x2127</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_UNSIGNEDBYTE</td>
<td>8488</td>
<td>0x2128</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_UNSIGNEDDINT</td>
<td>8489</td>
<td>0x2129</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_UNSIGNEDLONG</td>
<td>8490</td>
<td>0x212A</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_UNSIGNEDSHORT</td>
<td>8491</td>
<td>0x212B</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_YEAR</td>
<td>8492</td>
<td>0x212C</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_YEARMONTH</td>
<td>8493</td>
<td>0x212D</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_SIMPLETYPE</td>
<td>8704</td>
<td>0x2200</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_COMPLEXTYPE</td>
<td>9216</td>
<td>0x2400</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_PARTICLE</td>
<td>16384</td>
<td>0x4000</td>
<td>// particle mask</td>
</tr>
<tr>
<td>SOMITEM_ANY</td>
<td>16385</td>
<td>0x4001</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_ANYATTRIBUTE</td>
<td>16386</td>
<td>0x4002</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_ELEMENT</td>
<td>16387</td>
<td>0x4003</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_GROUP</td>
<td>16640</td>
<td>0x4100</td>
<td>// group mask</td>
</tr>
<tr>
<td>SOMITEM_ALL</td>
<td>16641</td>
<td>0x4101</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_CHOICE</td>
<td>16642</td>
<td>0x4102</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_SEQUENCE</td>
<td>16643</td>
<td>0x4103</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_EMPTYPARTICLE</td>
<td>16644</td>
<td>0x4104</td>
<td></td>
</tr>
<tr>
<td>SOMITEM_NULL</td>
<td>2048</td>
<td>0x0800</td>
<td></td>
</tr>
<tr>
<td>SOMITEM singapore NULL_TYPE</td>
<td>10240</td>
<td>0x2800</td>
<td></td>
</tr>
<tr>
<td>SOMITEM singapore NULL_ANY</td>
<td>18433</td>
<td>0x4801</td>
<td></td>
</tr>
<tr>
<td>SOMITEM singapore NULL_ANYATTRIBUTE</td>
<td>18434</td>
<td>0x4802</td>
<td></td>
</tr>
<tr>
<td>SOMITEM singapore NULL_ELEMENT</td>
<td>18435</td>
<td>0x4803</td>
<td></td>
</tr>
</tbody>
</table>
SCHEMADERIVATIONMETHOD Enumerations

Used by the substitutionGroupExclusions property of the ISchemaElement interface, the disallowedSubstitutions property of the ISchemaElement interface, the final property of the ISchemaType interface, and the prohibitedSubstitution property of the ISchemaComplexType interface.

<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
<th>HEX Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMADERIVATIONMETHOD_EMPTY</td>
<td>0</td>
<td>0x0000</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_SUBSTITUTION</td>
<td>1</td>
<td>0x0001</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_EXTENSION</td>
<td>2</td>
<td>0x0002</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_RESTRICTION</td>
<td>4</td>
<td>0x0004</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_LIST</td>
<td>8</td>
<td>0x0008</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_UNION</td>
<td>16</td>
<td>0x0010</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_ALL</td>
<td>255</td>
<td>0x00FF</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_NONE</td>
<td>256</td>
<td>0x0100</td>
</tr>
</tbody>
</table>

SCHEMACONTENTTYPE Enumeration

Used by the contentType property of the ISchemaComplexType interface.

<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMACONTENTTYPE_EMPTY</td>
<td>0</td>
</tr>
<tr>
<td>SCHEMACONTENTTYPE_TEXTONLY</td>
<td>1</td>
</tr>
<tr>
<td>SCHEMACONTENTTYPE_ELEMENTONLY</td>
<td>2</td>
</tr>
<tr>
<td>SCHEMACONTENTTYPE_MIXED</td>
<td>3</td>
</tr>
</tbody>
</table>

SCHEMAPROCESSCONTENTS Enumerations

Used by the processContents property of the ISchemaAny interface.
<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAPROCESSCONTENTS_NONE</td>
<td>0</td>
</tr>
<tr>
<td>SCHEMAPROCESSCONTENTS_SKIP</td>
<td>1</td>
</tr>
<tr>
<td>SCHEMAPROCESSCONTENTS_LAX</td>
<td>2</td>
</tr>
<tr>
<td>SCHEMAPROCESSCONTENTS_STRICT</td>
<td>3</td>
</tr>
</tbody>
</table>

**SCHEMAWHITESPACE Enumerations**

Used by the `whitespace` property of the `ISchemaType` interface.

<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAWHITESPACE_NONE</td>
<td>-1</td>
</tr>
<tr>
<td>SCHEMAWHITESPACE_PRESERVE</td>
<td>0</td>
</tr>
<tr>
<td>SCHEMAWHITESPACE_REPLACE</td>
<td>1</td>
</tr>
<tr>
<td>SCHEMAWHITESPACE_COLLAPSE</td>
<td>2</td>
</tr>
</tbody>
</table>

**SCHEMATYPEVARIETY Enumerations**

Used by the `variety` property of the `ISchemaType` interface.

<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMATYPEVARIETY_NONE</td>
<td>-1</td>
</tr>
<tr>
<td>SCHEMATYPEVARIETY_ATOMIC</td>
<td>0</td>
</tr>
<tr>
<td>SCHEMATYPEVARIETY_LIST</td>
<td>1</td>
</tr>
<tr>
<td>SCHEMATYPEVARIETY_UNION</td>
<td>2</td>
</tr>
</tbody>
</table>
ISchema Interface

Provides information about individual XML Schemas in the schema cache. Each XML Schema in the schema cache must have a unique namespace to differentiate it from other XML Schemas. The IISchema object is returned by using IXMLDOMSchemaCollection2.getSchema or ISchemaItem.schema. The getSchema method requires the namespace argument to return an XML Schema, and the schema property returns the XML Schema for the current XML Schema item.

Properties

- **attributeGroups**
  Retrieves the collection of `<attributeGroup>` declarations. This collection contains objects that implement the ISchemaAttributeGroup interface.

- **attributes**
  Retrieves the collection of top-level `<attribute>` declarations. This collection contains objects that implement the ISchemaAttribute interface.

- **elements**
  Retrieves the collection of top-level `<element>` declarations. This collection contains objects that implement the ISchemaElement interface.

- **modelGroups**
  Retrieves the collection of `<modelGroup>` declarations. This collection contains objects that implement the ISchemaModelGroup interface.

- **notations**
  Retrieves the collection of notations. This collection contains objects that implement the ISchemaNotation interface.

- **schemaLocations**
  Retrieves the string collection of XML Schema URIs that are imported to or included in an XML Schema.

- **targetNamespace**
  Retrieves the value of the targetNamespace attribute for the XML Schema.

- **types**
  Retrieves a collection of named simple-type and complex-type objects defined in the XML Schema.

  Retrieves the value of the version attribute of the XML
version Schema.

[Script]

**Implementation Syntax**

```
var oSchema = oSchemaItem.schema;
```

[Visual Basic]

**Implementation Syntax**

```
Set oSchema = oSchemaItem.schema
```

[C/C++]

**Implementation Syntax**

```
HRESULT schema(ISchema* schema);
```

**Versioning**

MSXML 4.0 and later

**Requirements**

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib, msxml2.idl

**Inheritance:** Inherits from the **ISchemaItem** interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

IXMLDOMSchemaCollection2 | ISchemaItem Interface | Using Namespaces in Schemas | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
attributeGroups Property

Returns an ISchemaItemCollection object. This collection contains ISchemaAttributeGroup objects that are top-level items. To find secondary and lower child items, you must use the ISchemaComplexType.contentModel property.

To obtain information about the attributes that are returned in the collection, use the ISchemaAttributeGroup interface. For more information about using attributes in an XML Schema, see the group element in the XML Schema Reference (XSD) and Attribute Groups in the XML Schema Developer's Guide.

Example

The following VBScript example shows how to list the top-level <attributeGroup> declarations.

```vbnet
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
oSchemaCache.add "", "po.xsd"
Set oSchema = oSchemaCache.getSchema(""

For Each oAttributeGroup in oSchema.attributeGroups
    WScript.Echo oAttributeGroup.name
Next

[Script]

Implementation Syntax

var oAttributeGroups = oISchema.attributeGroups;

Parameters

None.

Return Values
**oAttributeGroups**

An object. The collection of `<attributeGroup>` declarations. This collection contains objects that implement the ISchemaAttributeGroup interface.

[Visual Basic]

**Implementation Syntax**

Set oAttributeGroups = oISchema.attributeGroups

**Parameters**

None.

**Return Values**

**oAttributeGroups**

An object. The collection of `<attributeGroup>` declarations. This collection contains objects that implement the ISchemaAttributeGroup interface.

[C/C++]

**Implementation Syntax**

HRESULT get_attributeGroups (ISchemaItemCollection** attributeGroups

**Parameters**

attributeGroups [out,retval]

An object. The collection of `<attributeGroup>` declarations.

**Return Values**

S_OK

The value returned if successful.
E_P Bücher
The value returned if the attributeGroups collection is NULL.

E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaItemCollection Interface | ISchemaAttributesGroup Interface

Applies to: ISchema Interface
attributes Property

Returns an ISchemaItemCollection object. This collection contains the top-level ISchemaAttribute objects for the XML Schema. The ISchemaAttribute interface is used to further examine the collection of attribute objects.

Example

This example uses attributes from the ISchema.attributes property.

The VBScript example below uses the following XML Schema document.

```xml
<?xml version='1.0'?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <xsd:attribute name='attr1' type='xsd:string' />
  <xsd:attribute name='attr2' type='xsd:date' />
</xsd:schema>
```

The following is the VBScript example.

```vbs
Dim oSchemaCache as New XMLSchemaCache
Dim oSchema as ISchema
Dim oAttributes as ISchemaAttribute
Dim nsTarget as String

nsTarget = "http://www.w3.org/2000/09/xmldsig#"
oSchemaCache.add nsTarget, "po.xsd"
Set oSchema = oSchemaCache.getSchema(nsTarget)

For Each oAttribute in oSchema.attributes
  msgbox oAttribute.name
Next
```

The following VBScript example shows how to list the top-level <attribute> declarations.

```vbs
Set oSchemaCache = CreateObject("Msxml2/XMLSchemaCache.5.0")
oSchemaCache.add "", "po.xsd"
Set oSchema = oSchemaCache.getSchema(""

For Each oAttribute in oSchema.attributes
  WScript.Echo oAttribute.name
```

Implementation Syntax

var oAttributes = oISchema.attributes;

Parameters

None.

Return Values

oAttributes

An object. The collection of top-level <attribute> declarations. This collection contains objects that implement the ISchemaAttribute interface.

[Visual Basic]

Implementation Syntax

Set oAttributes = oISchema.attributes

Parameters

None.

Return Values

oAttributes

An object. The collection of top-level <attribute> declarations. This collection contains objects that implement the ISchemaAttribute interface.

[C/C++]

Implementation Syntax
HRESULT get_attributes (ISchemaItemCollection** attributes);

Parameters

attributes [out,retval]
   An object. The collection of top-level <attribute> declarations.

Return Values

S_OK
   The value returned if successful.
E_Pointer
   The value returned if the attributes collection is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaItemCollection Interface | ISchemaAttribute Interface

Applies to: ISchema Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
elements Property

Returns an ISchemaItemCollection object. This collection contains ISchemaElement objects that are top-level items. To find secondary and lower child items, you must use the ISchemaComplexType.contentModel property.

To obtain information about the elements that are returned in the collection, use the ISchemaElement interface. For more information about using elements in an XML Schema, see Declaring Elements in the XML Schema Developer's Guide.

Example

The following is an example of an element declared in an XML Schema.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Software">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="title" />  
        <xsd:element name="company" />  
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The following VBScript example shows how to list the top-level <element> declarations.

```vbs
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
oSchemaCache.add ", " , "po.xsd"
Set oSchema = oSchemaCache.getSchema("")

For Each oElement in oSchema.elements 
    WScript.Echo oElement.name
Next
```

[Script]

Implementation Syntax
var oElements = oISchema.elements;

Parameters

None.

Return Values

oElements
An object. The collection of top-level <element> declarations. This collection contains objects that implement the ISchemaElement interface.

[Visual Basic]

Implementation Syntax

Set oElements = oISchema.elements

Parameters

None.

Return Values

oElements
An object. The collection of top-level <element> declarations. This collection contains objects that implement the ISchemaElement interface.

[C/C++]

Implementation Syntax

HRESULT get_elements (ISchemaItemCollection** elements);

Parameters

elements [out,retval]
An object. The collection of top-level <element> declarations.

**Return Values**

**S_OK**
The value returned if successful.

**E_POINTER**
The value returned if the elements collection is NULL.

**E_FAIL**
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[ISchemaItemCollection Interface] | [ISchemaElement Interface]

Applies to: [ISchema Interface]
modelGroups Property

Returns an ISchemaItemCollection that contains ISchemaModelGroup objects. The model group will contain any combination of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny.

To obtain information about the objects returned in the collection, use the itemType property to determine the correct interface to use. For more information about the use of model groups in an XML schema, see Content Models and Named Model Groups, in the XML Schema Developer's Guide.

[Script]

Implementation Syntax

var oModelGroups = oISchema.modelGroups;

Parameters

None.

Return Values

oModelGroups

An object. The collection of <modelGroup> declarations. This collection contains objects that implement the ISchemaModelGroup interface.

[Visual Basic]

Implementation Syntax

Set oModelGroups = oISchema.modelGroups

Parameters
None.

**Return Values**

{
\textit{oModelGroups}
\begin{itemize}
  \item An object. The collection of \texttt{<modelGroup>} declarations. This collection contains objects that implement the \texttt{ISchemaModelGroup} interface.
\end{itemize}

[C/C++]

**Implementation Syntax**

\texttt{HRESULT get_modelGroups (ISchemaItemCollection** modelGroups);}

**Parameters**

\textit{modelGroups} [out,retval]
\begin{itemize}
  \item An object. The collection of \texttt{<modelGroup>} declarations.
\end{itemize}

**Return Values**

- S\_OK
  - The value returned if successful.
- E\_POINTER
  - The value returned if the \texttt{modelGroups} collection is NULL.
- E\_FAIL
  - The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- \texttt{ISchemaItemCollection Interface} | \texttt{ISchemaModelGroup Interface}

Applies to: \texttt{ISchema Interface}
notations Property

Returns a collection of notation objects.

Example

The following example shows how to list the <notation> declarations from the ISchemaItemCollection of notations.

```
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
oSchemaCache.add ",", "po.xsd"
Set oSchema = oSchemaCache.getSchema(""

For Each oNotation in oSchema.notation
    WScript.Echo oNotation.name + " " + oNotation.sourceforge + oNotation.publicIdentifier +
Next
```

[Script]

Implementation Syntax

```
var oNotations = oISchema.notation;
```

Parameters

None.

Return Values

oNotations

An object. The collection of notations. This collection contains objects that implement the ISchemaNotation interface.

[Visual Basic]

Implementation Syntax
Set oNotations = oISchema.notations

**Parameters**

None.

**Return Values**

*oNotations*

An object. The collection of notations. This collection contains objects that implement the ISchemaNotation interface.

**[C/C++]**

**Implementation Syntax**

HRESULT get_notations (ISchemaItemCollection** notations);

**Parameters**

notations [out,retval]

An object. The collection of notations.

**Return Values**

S_OK

The value returned if successful.

E_Pointer

The value returned if the notations collection is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**
ISchemaItemCollection Interface | ISchemaNotation Interface

Applies to: ISchema Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
schemaLocations Property

Returns a string collection. This collection contains all of the URIs that are imported to or included in the XML Schema object being used. Each URI in the XML Schema is represented by a string item in the collection. The URI text is retrieved by using the item property of the ISchemaStringCollection.

Example

The following VBScript example shows how to lists the URIs of the imported or included XML Schemas.

Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
oSchemaCache.add "", "po.xsd"
Set oSchema = oSchemaCache.getSchema("")
For Each oURI in oSchema.schemaLocations
    WScript.Echo oURI
Next

[Script]

Implementation Syntax

var oSchemaLocations = oISchema.schemaLocations;

Parameters

None.

Return Values

OISchemaLocations
An object. The collection of XML Schema URIs imported to or included in this XML Schema. This collection contains strings that contain the schema URIs.
[Visual Basic]

**Implementation Syntax**

```
Set oSchemaLocations = oISchema.schemaLocations
```

**Parameters**

None.

**Return Values**

- `oSchemaLocations`
  An object. The collection of XML Schema URIs imported to or included in this XML Schema. This collection contains strings that contain the schema URIs.

[C/C++]

**Implementation Syntax**

```
HRESULT get_schemaLocations (ISchemaStringCollection** schemaLocations);
```

**Parameters**

- `schemaLocations` [out,retval]
  An object. The collection of schema URIs.

**Return Values**

- `S_OK`
  The value returned if successful.
- `E_Pointer`
  The value returned if the `schemaLocations` collection is NULL.
- `E_FAIL`
  The value returned if something else is wrong.
To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaStringCollection Interface | Using Other Namespaces

Applies to: ISchema Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
targetNamespace Property

Returns a string that indicates the value of the targetNamespace attribute of the XML Schema. The targetNamespace is an attribute of the <schema> declaration.

Example

The following is an example of the targetNamespace attribute in an XML Schema.

```xml
<schema targetNamespace="http://www.samplenamespace.microsoft.com">
    <element>
    </element>
</schema>
```

[Script]

**Implementation Syntax**

```javascript
var strTargetNamespace = oISchema.targetNamespace;
```

**Parameters**

None.

**Return Values**

`strTargetNamespace`


[Visual Basic]

**Implementation Syntax**

```vbnet
var strTargetNamespace = oISchema.targetNamespace
```
Parameters

None.

Return Values

\textit{strTargetNamespace}


[C/C++]

Implementation Syntax

\begin{verbatim}
HRESULT get_targetNamespace(BSTR* targetNamespace);
\end{verbatim}

Parameters

\textit{targetNamespace} [out,retval]


Return Values

\begin{enumerate}
\item \textbf{S\_OK}
  \begin{description}
  \item[The value returned if successful.]
  \end{description}
\item \textbf{E\_Pointer}
  \begin{description}
  \item[The value returned if the targetNamespace is NULL.]
  \end{description}
\item \textbf{E\_FAIL}
  \begin{description}
  \item[The value returned if something else is wrong.]
  \end{description}
\end{enumerate}

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

\begin{itemize}
\item [Using Namespaces in Schemas](Using\ Namespaces\ in\ Schemas) \ | \ [Using Other Namespaces](Using\ Other\ Namespaces)
\end{itemize}

Applies to: [ISchema Interface](ISchema\ Interface)
**types Property**

Returns an ISchemaItemCollection object. This collection contains top-level <simpleType> and <complexType> declarations of the XML Schema. The ISchemaType.itemType property determines whether the object is of the type complex or simple.

**Example**

This VBScript example shows type information being extracted from an XML Schema document.

The following is the XML Schema used in the example.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" finalDefault>
  <xsd:element name="item">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="productName" type="xsd:string"/>
        <xsd:element name="quantity">
          <xsd:simpleType>
            <xsd:restriction base="xsd:positiveInteger">
              <xsd:maxExclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="USPrice" type="xsd:decimal"/>
        <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="partNum" type="xsd:int" use="required"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The following is the VBScript example.

```vbnet
Dim oSchemaCache as New XMLSchemaCache50
Dim oSchema as ISchema
Dim oType as ISchemaType
Dim nsTarget as String
Dim strText as String
```
nsTarget = "http://www.w3.org/2000/09/xmldsig#"
oSchemaCache.add nsTarget, "po.xsd"
Set oSchema = oSchemaCache.getSchema(nsTarget)
For Each oType in oSchema.types
    strText = oType.name & " is a ">
    If oType.itemType = SOMITEM_ANYTYPE Then
        strText = strText & "built-in complex any type"
    End If
    If oType.itemType = SOMITEM_SIMPLETYPE Then
        strText = strText & "simple type"
    End If
    If oType.itemType = SOMITEM_COMPLEXTYPE Then
        strText = strText & "complex type"
    End If
    If (oType.itemType and SOMITEM_DATATYPE) = SOMITEM_DATATYPE Then
        strText = strText & "built-in type"
    End If
Next
msgbox strText

[Script]

Implementation Syntax

var oTypes = oISchema.types;

Parameters

None.

Return Values

oTypes
    An object. The collection of named simple-type and complex-type objects
defined in the XML Schema. This collection contains objects that
implement the ISchemaType interface and the ISchemaComplexType
interface.

[Visual Basic]

Implementation Syntax
Set oTypes = oISchema.types

**Parameters**

None.

**Return Values**

**oTypes**

An object. The collection of named simple-type and complex-type objects defined in the XML Schema. This collection contains objects that implement the ISchemaType interface and the ISchemaComplexType interface.

[C/C++]

**Implementation Syntax**

HRESULT get_types (ISchemaItemCollection** types);

**Parameters**

**types [out,retval]**

An object. The collection of named simple-type and complex-type objects defined in the XML Schema. This collection contains objects that implement the ISchemaType interface or the ISchemaComplexType interface. Use ISchemaType::isComplexType to query for the correct interface.

**Return Values**

**S_OK**

The value returned if successful.

**E_POINTER**

The value returned if the types object is Null.

**E_FAIL**

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the
Language Filter button in the upper-left corner of the page.

See Also

ISchemaItemCollection Interface | ISchemaType Interface | ISchemaComplexType Interface | ISchemaItem itemType Property

Applies to: ISchema Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
version Property

Returns a string that indicates the value of the version attribute for the `<schema>` declaration.

Example

The following is an example of the version attribute in an XML Schema.

```xml
<xsd:schema version="4.57.2" xmlns:xsd="www.w3.org/2001/XMLSchema"
    <xsd:element>
    </xsd:element>
</xsd:schema>
```

[Script]

**Implementation Syntax**

```javascript
var strVersion = oISchema.version;
```

**Parameters**

None.

**Return Values**

`strVersion`

[Visual Basic]

**Implementation Syntax**

```vbnet
strVersion = oISchema.version
```

**Parameters**

None.
None.

**Return Values**

$strVersion$


[C/C++]

**Implementation Syntax**

```c
HRESULT get_version(BSTR* version);
```

**Parameters**

`version [out,retval]`


**Return Values**

S_OK

The value returned if successful.

E_Pointer

The value returned if the `version` parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [ISchema Interface](#)
ISchemaAny Interface

Provides information about an <anyAttribute> declaration. The <anyAttribute> contains well-formed XML in the text portion of its element container. The two properties of the ISchemaAny interface provide namespace URI and process handling information for the <anyAttribute> declaration. The ISchemaAny object is returned from the anyAttribute property of the ISchemaComplexType interface and the ISchemaAttributeGroup interface.

Properties

namespaces

Retrieves the collection of namespaces for the <anyAttribute> declaration.

processContents

Retrieves the processContents instructions for the <anyAttribute> declaration.

[Script]
var oSchemaAny = oSchemaComplexType.anyAttribute;

[Visual Basic]
Set oSchemaAny = oSchemaComplexType.anyAttribute

[C/C++]
HRESULT anyAttribute(ISchemaAny** anyAttribute);

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib
Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaParticle interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaComplexType anyAttribute Property | anyAttribute Element | any Element | SOM Reference
namespaces Property

Returns a string collection of namespaces. The namespaces are used by the <anyAttribute> declaration to validate the well-formed XML.

[Script]

Implementation Syntax

var oNamespaces = oISchemaAny.namespaces;

Parameters

None.

Return Values

oNamespaces

An object. The collection of namespaces for the any particle. The namespaces returned can be: ((#any|#other)|List of (anyURI| (#targetNamespace|#local))).

[Visual Basic]

Implementation Syntax

Set oNamespaces = oISchemaAny.namespaces

Parameters

None.

Return Values

oNamespaces
An object. The collection of namespaces for the any particle. The namespaces returned can be: ((##any|##other)|List of (anyURI|(#targetNamespace|##local))).

[C/C++]

**Implementation Syntax**

```c
HRESULT get_namespaces(ISchemaStringCollection* namespaces);
```

**Parameters**

`namespaces` [out,retval]

An object. The collection of namespaces for the any particle.

**Return Values**

- S_OK
  - The value returned if successful.
- E_POINTER
  - The value returned if the `namespaces` object is NULL.
- E_FAIL
  - The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- [ISchemaStringCollection Interface](#) | [Using Namespaces in Schemas](#)

Applies to: [ISchemaAny Interface](#)
processContents Property

Returns an integer that corresponds to a declared constant of the SCHEMAPROCESSCONTENTS type. The following table shows each value and its description.

<table>
<thead>
<tr>
<th>Process content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAPROCESSCONTENTS_SKIP</td>
<td>Validation is skipped for the XML content.</td>
</tr>
<tr>
<td>SCHEMAPROCESSCONTENTS_LAX</td>
<td>Validation is performed if a schema can be located for the &lt;anyAttribute&gt; declaration.</td>
</tr>
<tr>
<td>SCHEMAPROCESSCONTENTS_STRICT</td>
<td>Validation is performed and an error is returned if the schema cannot be located.</td>
</tr>
</tbody>
</table>

[Script]

Implementation Syntax

```javascript
var oProcessContents = oISchemaAttributeGroup.processContents;
```

Parameters

None.

Return Values

`oProcessContents`

An object. The processContents for the any particle.

[Visual Basic]

Implementation Syntax
Set oProcessContents = oISchemaAttributeGroup.processContents

**Parameters**

None.

**Return Values**

\( oProcessContents \)

An object. The processContents for the any particle.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_processContents(SCHEMAPROCESSCONTENTS* processContents);
```

**Parameters**

\( processContents \) [out,retval]

An object. The processContents for the any particle.

**Return Values**

S_OK
The value returned if successful.

E_POINTER
The value returned if the processContents object is NULL.

E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

any Element | anyAttribute Element
Applies to: ISchemaAny Interface
ISchemaAttribute Interface

Provides information about attribute items. Use the ISchema.attributes property to obtain ISchemaAttribute items.

Example

The following example shows ISchemaAttribute objects being created.

```
Dim oSchema as MSXML5.ISchema
Dim oSchemaAttr as MSXML5.ISchemaAttribute
Set oSchemaAttr = oSchema.attributes
```

Properties

- **defaultValue**
  Retrieves the default value of the attribute.

- **fixedValue**
  Retrieves the fixed value of the attribute.

- **isReference**
  Retrieves whether the attribute object is a reference to a top-level <element> declaration.

- **scope**
  Retrieves the element definition type, either global or complex.

- **type**
  Retrieves the type of the attribute.

- **use**
  Retrieves the use style for the attribute

[Script]

Implementation Syntax

```
var oAttributes = oSchema.attributes;
var oAttribute = oAttributes(index);
```

Parameters

- **index**
A long integer. An index of the item within the collection. The first item is zero.

[Visual Basic]

**Implementation Syntax**

Set oAttributes = oSchema.attributes
Set oAttribute = oAttributes(index)

**Parameters**

`index`
A long integer. An index of the item within the collection. The first item is zero.

[C/C++]

**Implementation Syntax**

HRESULT get_attributes (ISchemaItemCollection** attributes);
HRESULT attributes->get_item(long index, ISchemaAttribute** attribute)

**Parameters**

`index`
A long integer. An index of the item within the collection. The first item is zero.

**Versioning**

MSXML 4.0 and later

**Requirements**

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib
Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaItem interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Declaring Attributes | SOM Reference
defaultValue Property

Returns a string. This string is the default value of the attribute being queried. The <attribute> declaration includes the default attribute.

Example

The following is an example of a default attribute in an XML Schema.

```xml
<schema>
  <element name="someelement">
    <attribute name="someattribute" default="attr text" type="string"/>
  </element>
</schema>
```

[Script]

**Implementation Syntax**

```javascript
var strDefaultValue = oISchemaAttribute.defaultValue;
```

**Parameters**

None.

**Return Values**

`strDefaultValue`

A string. The value of the default attribute.

[Visual Basic]

**Implementation Syntax**

```vbnet
strDefaultValue = oISchemaAttribute.defaultValue
```

**Parameters**
Return Values

strDefaultValue
A string. The value of the default attribute.

[C/C++]

Implementation Syntax

HRESULT get_defaultValue(BSTR* defaultValue);

Parameters

defaultValue [out,retval]
A string. The value of the default attribute.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the defaultValue parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

element Element

Applies to: ISchemaAttribute Interface
**fixedValue Property**

Returns a string. This string is the fixed value of the attribute being queried. The `<attribute>` declaration includes the `fixed` attribute.

**Example**

The following is an example of the `fixed` attribute in an XML Schema.

```xml
<schema>
  <element name="someelement">
    <attribute name="someattribute" fixed="attr text" type="string"/>
  </element>
</schema>
```

**Implementation Syntax**

```javascript
var strFixedValue = oISchemaAttribute.fixedValue;
```

**Parameters**

None.

**Return Values**

`strFixedValue`

A string. The value of the `fixed` attribute.

**[Visual Basic]**

**Implementation Syntax**

```javascript
strFixedValue = oISchemaAttribute.fixedValue
```

**Parameters**
None.

**Return Values**

`strFixedValue`

A string. The value of the `fixed` attribute.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_fixedValue(BSTR* fixedValue);
```

**Parameters**

`fixedValue` [out, retval]

A string. The value of the `fixed` attribute.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the `fixedValue` parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[element Element](#)

Applies to: [ISchemaAttribute Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
isReference Property

Returns a Boolean variant. A value of true indicates that the XML Schema attribute object is a reference to a top-level <element> declaration.

[Script]

Implementation Syntax

```javascript
var bolIsReference = oISchemaAttribute.isReference;
```

Parameters

None.

Return Values

`bolIsReference`
A Boolean variant. Indicates whether the XML Schema attribute object is a reference to a top-level <element> declaration.

[Visual Basic]

Implementation Syntax

```vbnet
bolIsReference = oISchemaAttribute.isReference
```

Parameters

None.

Return Values

`bolIsReference`
A Boolean variant. Indicates whether the XML Schema attribute object is
a reference to a top-level <element> declaration

[C/C++]

Implementation Syntax

HRESULT get_isReference (VARIANT_BOOL* isReference);

Parameters

isReference [out,retval]
   A Boolean variant. Indicates whether the XML Schema attribute object is
   a reference to a top-level <element> declaration

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the isReference parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the
Language Filter button in the upper-left corner of the page.

See Also

attribute Element | Declaring Attributes

Applies to: ISchemaAttribute Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**scope Property**

Returns an ISchemaComplexType. The complex type is a definition of the locally declared complex type. Globally declared types will return NULL for the scope property. This property is absent in the case of declarations within named model groups.

**[Script]**

**Implementation Syntax**

```javascript
var oScope = oISchemaAttribute.scope;
```

**Parameters**

None.

**Return Values**

`oScope`

An object. A `<complexType>` declaration of the attribute being queried.

**[Visual Basic]**

**Implementation Syntax**

```vbnet
Set oScope = oISchemaAttribute.scope
```

**Parameters**

None.

**Return Values**

`oScope`
An object. A `<complexType>` declaration of the attribute being queried.

[C/C++]

**Implementation Syntax**

```
HRESULT get_scope(ISchemaComplexType** scope);
```

**Parameters**

`scope [out,retval]`

An object. A `<complexType>` declaration of the attribute being queried.

**Return Values**

- **S_OK**
  The value returned if successful.
- **E_POINTER**
  The value returned if the `scope` parameter is NULL.
- **E_FAIL**
  The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note** Globally declared elements will return a NULL value for this property.

**See Also**

- [ISchemaItem Interface](#)

Applies to: [ISchemaAttribute Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Referen...
type Property

Returns an ISchemaType object. This object contains type information about the attribute item being queried. Use the ISchemaType interface to obtain further information about the attribute's type object. When the value returned for the type property is SCHEMAUSE_PROHIBITED, the value of the ISchemaItem.itemType property of the attribute is SOMITEM_DATATYPE_ANYTYPE.

Example

The following is an example of the type definition in a schema.

```xml
<schema>
   <attribute name="someAttribute" type = "xsd:string"/>
   <complexType name="newitemtype">
   </complexType>
</schema>
```

[Script]

**Implementation Syntax**

```javascript
var oType = oISchemaAttribute.type;
```

**Parameters**

None.

**Return Values**

`oType`

An object. The type of the attribute.

[Visual Basic]

**Implementation Syntax**
Set oType = oISchemaAttribute.type

**Parameters**

None.

**Return Values**

*oType*

An object. The type of the attribute.

[C/C++]

**Implementation Syntax**

HRESULT get_type(ISchemaType** type);

**Parameters**

*type [out,retval]*

An object. A simple type definition.

**Return Values**

*S_OK*

The value returned if successful.

*E_POINTER*

The value returned if the type parameter is NULL.

*E_FAIL*

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[ISchemaType Interface] | [Simple Type Definitions]
Applies to: ISchemaAttribute Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
use Property

Returns an integer that corresponds to a value from the declared constant SCHEMAUSE values. The use property indicates the type of validation restrictions that are on an attribute. The <attribute> declaration includes the use attribute.

When the value returned for the use property is SCHEMAUSE_PROHIBITED, the value of the ISchemaItem.itemType property of the attribute is SOMITEM_DATATYPE_ANYTYPE. The following table shows the values returned and their descriptions.

<table>
<thead>
<tr>
<th>SCHEMAUSE value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAUSE_OPTIONAL</td>
<td>Attribute is not required to be in the document.</td>
</tr>
<tr>
<td>SCHEMAUSE_PROHIBITED</td>
<td>Attribute cannot be in the document.</td>
</tr>
<tr>
<td>SCHEMAUSE_REQUIRED</td>
<td>Attribute is required to be in the document.</td>
</tr>
</tbody>
</table>

Example

The following is an example of the use attribute in an XML Schema.

```xml
<schema>
   <element name="someelement">
      <attribute name="someattribute" use="prohibited"/>
      <attribute name="newattribute" use="required"/>
   </element>
</schema>
```

[Script]

Implementation Syntax

```javascript
var conUse = oISchemaAttribute.use;
```

Parameters
None.

**Return Values**

*conUse*

A constant. The use style for the attribute.

[Visual Basic]

**Implementation Syntax**

```vbnet
conUse = oISchemaAttribute.use
```

**Parameters**

None.

**Return Values**

*conUse*

A constant. The use style for the attribute.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_use(SCHEMAUSE* use);
```

**Parameters**

*use* [out,retval]

An object. The use style for the attribute. The following values are possible.

- SCHEMAUSE_OPTIONAL
- SCHEMAUSE_PROHIBITED
- SCHEMAUSE_REQUIRED
**Return Values**

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the use parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

attribute Element

Applies to: ISchemaAttribute Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
ISchemaAttributeGroup Interface

Provides information about attribute groups. This interface provides the attributes and the <anyAttribute> element that are included in the <attributeGroup> declaration. The ISchemaAttributeGroup objects are returned from the attributeGroups property of the ISchema interface.

Properties

anyAttribute
Retrieves an ISchemaAny object with information about the <anyAttribute> element.

attributes
Retrieves the collection of attributes declared. The collection contains objects that implement the ISchemaAttribute interface.

[Script]

Implementation Syntax

var oAttributeGroups = oSchema.attributeGroups;
var oAttributeGroup = oAttributeGroups(index);

Parameters

index
A long integer. An index of the item within the collection. The first item is zero.

[Visual Basic]

Implementation Syntax

Set oAttributeGroups = oSchema.attributeGroups
Set oAttributeGroup = oAttributeGroups(index)

Parameters
index
A long integer. An index of the item within the collection. The first item is zero.

[C/C++]

Implementation Syntax

HRESULT get_attributeGroups(ISchemaItemCollection** attributeGroups);
HRESULT attributeGroups->get_item(
    long index,
    ISchemaAttributeGroup** attributeGroup);

Parameters

index
A long integer. An index of the item within the collection. The first item is zero.

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaItem interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
anyAttribute Property

Returns an ISchemaAny object. This object provides information about the <anyAttribute> element in the <attributeGroup> declaration.

[Script]

Implementation Syntax

var oAnyAttribute = oISchemaAttributeGroup.anyAttribute;

Parameters

None.

Return Values

oAnyAttribute
  An object. The anyAttribute object, if one is declared.

[Visual Basic]

Implementation Syntax

Set oAnyAttribute = oISchemaAttributeGroup.anyAttribute

Parameters

None.

Return Values

oAnyAttribute
  An object. The anyAttribute object, if one is declared.
Implementation Syntax

HRESULT get_anyAttribute(ISchemaAny* anyAttribute);

Parameters

anyAttribute [out,retval]
An object. Pointer to the anyAttribute object.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the anyAttribute object is NULL
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaAny Interface | anyAttribute Element
Applies to: ISchemaAttributeGroup Interface
attributes Property

Returns a collection of attributes that are declared in the attribute group declaration. Use the ISchemaAttribute interface to get information from individual attributes in the collection.

[Script]

**Implementation Syntax**

```javascript
var oAttributes = oISchemaAttributeGroup.attributes;
```

**Parameters**

None.

**Return Values**

`oAttributes`

An object. The collection of declared attributes. This collection contains objects that implement the ISchemaAttribute interface.

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oAttributes = oISchemaAttributeGroup.attributes
```

**Parameters**

None.

**Return Values**

`oAttributes`
An object. The collection of declared attributes. This collection contains objects that implement the ISchemaAttribute interface.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_attributes(ISchemaItemCollection** attributes);
```

**Parameters**

`attributes [out,retval]`

An object. The collection of attributes.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the attributes object is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaAttribute Interface | Attribute Groups | attributeGroup Element

Applies to: ISchemaAttributeGroup Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
ISchemaComplexType Interface

Provides access to basic type information and a DOM representation of the type object. You obtain an ISchemaComplexType object by using the scope property of the ISchemaAttribute interface, the scope property of the ISchemaElement interface or the types property of ISchema interface.

Properties

- **anyAttribute**
  Retrives the anyAttribute object, if one is declared.

- **attributes**
  Retrieves the collection of attributes declared. This collection contains objects that implement the ISchemaAttribute interface.

- **contentModel**
  Retrieves the content model of the complexType object being queried.

- **contentType**
  Retrieves the content-type of the complexType object being queried.

- **isAbstract**
  Returns true if the complexType object is abstract.

- **prohibitedSubstitution**
  Retrieves the prohibited substitutions for complexType object being queried.

[Script]

**Implementation Syntax**

```script
var oSchemaComplexType = oSchemaElement.scope;
```

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oSchemaComplexType = oSchemaElement.scope
```

[C/C++]

**Implementation Syntax**

```c++
```

Implementation Syntax

HERESULT types(ISchemaItemCollection** types);

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaType interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaAttribute scope Property | ISchemaElement scope Property | ISchema types Property | Defining Complex Types | Complex Type Definitions | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer"
anyAttribute Property

Returns an ISchemaAny object. This object provides information about the <anyAttribute> element in the <complexType> declaration.

[Script]

Implementation Syntax

var oAnyAttribute = oISchemaComplexType.anyAttribute;

Parameters

None.

Return Values

oAnyAttribute
   An object. The anyAttribute object, if one is declared.

[Visual Basic]

Implementation Syntax

Set oAnyAttribute = oISchemaComplexType.anyAttribute

Parameters

None.

Return Values

oAnyAttribute
   An object. The anyAttribute object, if one is declared.
[C/C++]

**Implementation Syntax**

```c
HRESULT get_anyAttribute(ISchemaAny** anyAttribute);
```

**Parameters**

*anyAttribute* [out,retval]

An object. Pointer to the anyAttribute object.

**Return Values**

- **S_OK**
  The value returned if successful.
- **E_POINTER**
  The value returned if the anyAttribute object is NULL.
- **E_FAIL**
  The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- **anyAttribute Element**

Applies to: **ISchemaComplexType Interface**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**attributes Property**

Returns an attribute collection. This collection contains the attributes that are a part of the `<complexType>` declaration. You find other attribute properties by using the `ISchemaAttribute` interface with the items in the collection.

[Script]

**Implementation Syntax**

```javascript
var oAttributes = oISchemaComplexType.attributes;
```

**Parameters**

None.

**Return Values**

`oAttributes`  
An object. The collection of attributes declared. This collection contains objects that implement the `ISchemaAttribute` interface.

[Visual Basic]

**Implementation Syntax**

```v Basic
Set oAttributes = oISchemaComplexType.attributes
```

**Parameters**

None.

**Return Values**

`oAttributes`
An object. The collection of attributes declared. This collection contains objects that implement the ISchemaAttribute interface.

[C/C++]

**Implementation Syntax**

`HERESULT get_attributes(ISchemaItemCollection** attributes);`

**Parameters**

`attributes [out,retval]`

An object. The collection of attributes.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the attributes collection is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

attribute Element

Applies to: ISchemaComplexType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
contentModel Property

Returns an ISchemaModelGroup object. This object contains information about the content model of the complexType being queried. The ISchemaModelGroup interface is used to obtain further information about the object.

[Script]

Implementation Syntax

```javascript
var oContentModel = oISchemaComplexType.contentModel;
```

Parameters

None.

Return Values

```
oContentModel
   An object. The content model of the type.
```

[Visual Basic]

Implementation Syntax

```vbscript
Set oContentModel = oISchemaComplexType.contentModel
```

Parameters

None.

Return Values

```
oContentModel
   An object. The content model of the type.
```
Implementation Syntax

HRESULT get_contentModel (ISchemaModelGroup** contentModel);

Parameters

contentModel [out, retval]
   An object. The content model of the type.

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the contentModel object is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaModelGroup Interface

Applies to: ISchemaComplexType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**contentType Property**

Returns an integer that corresponds to a declared constant value of the SCHEMACONTENTTYPE enumeration. See [XML SOM Enumerated Constants](#) for the integer values.

The following table lists the four possible content types and their descriptions.

<table>
<thead>
<tr>
<th>SCHEMACONTENTTYPE value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMACONTENTTYPE_EMPTY</td>
<td>Type does not contain elements, text, or attributes.</td>
</tr>
<tr>
<td>SCHEMACONTENTTYPE_TEXTONLY</td>
<td>Type contains only text. No elements or attributes.</td>
</tr>
<tr>
<td>SCHEMACONTENTTYPE_ELEMENTONLY</td>
<td>Type contains only element declarations.</td>
</tr>
<tr>
<td>SCHEMACONTENTTYPE_MIXED</td>
<td>Type contains elements, attributes, or text.</td>
</tr>
</tbody>
</table>

[Script]

**Implementation Syntax**

```javascript
var conContentType = oISchemaComplexType.contentType;
```

**Parameters**

None.

**Return Values**

`conContentType`

An integer. The contentType of the complexType.

[Visual Basic]
**Implementation Syntax**

```c
conContentType = oISchemaComplexType.contentTypeDef
```

**Parameters**

None.

**Return Values**

`conContentType`

An integer. The `contentType` of the `complexType`.

**[C/C++]**

**Implementation Syntax**

```c
HRESULT get_contentType (SCHEMACONTENTTYPE* contentType);
```

**Parameters**

`contentType [out,retval]`

An integer. The `contentType` of the `complexType`.

**Return Values**

S_OK

The value returned if successful.

E_FAIL

The value returned if something is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [ISchemaComplexType Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
isAbstract Property

Returns a Boolean true or false. If the value is true, the type can appear as the type definition in an element declaration, and can be referenced as an `xsi:type` attribute in a document. If the value is false, the type cannot appear as the type definition or be referenced as an `xsi:type` attribute.

[Script]

**Implementation Syntax**

```javascript
var bolIsAbstract = oISchemaComplexType.isAbstract;
```

**Parameters**

None.

**Return Values**

`bolIsAbstract`

A Boolean value. Returns true if the `complexType` object is abstract.

[Visual Basic]

**Implementation Syntax**

```vbscript
bolIsAbstract = oISchemaComplexType.isAbstract
```

**Parameters**

None.

**Return Values**

`bolIsAbstract`
A Boolean value. Returns true if the complexType object is abstract.

[C/C++]

**Implementation Syntax**

HERESULT get_isAbstract(VARIANT_BOOL* isAbstract);

**Parameters**

`isAbstract` [out,retval]
A Boolean value. VARIANT_TRUE if the complex-type is abstract; VARIANT_FALSE if the complex-type is not abstract.

**Return Values**

S_OK
The value returned if successful.
E_POINTER
The value returned if the `isAbstract` parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

complexType Element

Applies to: ISchemaComplexType Interface
prohibitedSubstitution Property

Returns a value from the SCHEMADERIVATIONMETHOD enumeration. The possible values are SCHEMADERIVATIONMETHOD_EXTENSION and SCHEMADERIVATIONMETHOD_RESTRICTION. The returned value specifies whether an <element> declaration that appears in a content model will be prevented from additionally validating elements.

See XML SOM Enumerated Constants for the integer values.

Validation might be prevented for one or both of the following two reasons.

- The additional element contains an xsi:type attribute that identifies an extension or restriction for the type of the declared element.
- The element is in a substitution group. This group has had a substitution performed against it using the element that has a declaration with the prohibitedSubstitution attribute.

[Script]

Implementation Syntax

```javascript
var conProhibitedSubstitution = oISchemaComplexType.prohibitedSubstitution;
```

Parameters

None.

Return Values

`conProhibitedSubstitution`

An integer. The prohibited substitution for this complexType. The prohibited substitution can be one of the following:

```javascript
SCHEMADERIVATIONMETHOD_EXTENSION
```
Implementation Syntax

**[Visual Basic]**

```visual-basic
conProhibitedSubstitution = oISchemaComplexType.prohibitedSubstitution
```

**Parameters**

None.

**Return Values**

*conProhibitedSubstitution*

An integer. The prohibited substitution for this `complexType`. The prohibited substitution can be one of the following:

- SCHEMADERIVATIONMETHOD_EXTENSION
- SCHEMADERIVATIONMETHOD_RESTRICTION
- SCHEMADERIVATIONMETHOD_ALL

**[C/C++]**

Implementation Syntax

```c
HERESULT get_prohibitedSubstitution (SCHEMAFILTER* prohibitedSubstitution);
```

**Parameters**

*prohibitedSubstitution* [out,retval]

An integer. The prohibited substitutions for this `complexType`. The prohibited substitution can be one of the following:
SCHEMADERIVATIONMETHOD_EXTENSION
SCHEMADERIVATIONMETHOD_RESTRICTION
SCHEMADERIVATIONMETHOD_ALL

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the prohibitedSubstitution parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: ISchemaComplexType Interface
ISchemaElement Interface

Provides information about element items. You obtain ISchemaElement items by using the ISchema.elements property, the ISchemaElement.substitutionGroup property, or the ISchemaModelGroup.particles property.

Properties

- **defaultValue**
  - Retrieves the default value of the element.

- **disallowedSubstitutions**
  - Retrieves the disallowed substitutions.

- **fixedValue**
  - Retrieves the fixed value of the element.

- **identityConstraints**
  - Retrieves the collection of ISchemaIdentityConstraint objects.

- **isAbstract**
  - Returns a true value if the element object being queried has an abstract attribute value of true.

- **isNillable**
  - Retrieves the value of the nullable attribute of the element.

- **isReference**
  - Retrieves whether the element object is a reference to a top-level <element> declaration.

- **scope**
  - Retrieves the element definition type, either global or complex.

- **substitutionGroup**
  - Retrieves the ISchemaElement object.

- **substitutionGroupExclusions**
  - Retrieves the substitution group exclusions.

- **type**
  - Retrieves the type of the element.

[Script]

```
var oElements = oSchema.elements;
var oSchemaElement = oSchemaElement.item(index);
```

Parameters
index
A long integer. An index of the item within the collection. The first item is zero.

[Visual Basic]

Implementation Syntax

Set oElements = oSchema.elements
Set oElement = oElements.item(index)

Parameters

index
A long integer. An index of the item within the collection. The first item is zero.

[C/C++]

Implementation Syntax

HRESULT get_elements (ISchemaItemCollection** elements);
HRESULT elements->get_item(long index, ISchemaElement** element);

Parameters

index
A long integer. An index of the item within the collection. The first item is zero.

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]
Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaParticle interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Declaring Elements | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
defaultValue Property

Returns a string. This string contains the default value of the element being queried. The declaration of this element has a default attribute.

[Script]

**Implementation Syntax**

```javascript
var strDefaultValue = oISchemaElement.defaultValue;
```

**Parameters**

None.

**Return Values**

- **strDefaultValue**
  
  A string. The value of the default attribute in the `<element>` declaration.

[Visual Basic]

**Implementation Syntax**

```vbscript
strDefaultValue = oISchemaElement.defaultValue
```

**Parameters**

None.

**Return Values**

- **strDefaultValue**
  
  A string. The value of the default attribute in the `<element>` declaration.
**Implementation Syntax**

HRESULT get_defaultValue(BSTR* defaultValue);

**Parameters**

*defaultValue [out,retval]*

A string. The value of the default attribute in the `<element>` declaration.

**Return Values**

- **S_OK**
  
  The value returned if successful.

- **E_POINTER**

  The value returned if the defaultValue parameter is NULL.

- **E_FAIL**

  The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

- [element Element](#)

Applies to: [ISchemaElement Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
disallowedSubstitutions Property

Returns an enumerated value from the SCHEMADERIVATIONMETHOD declarations. The possible values are SCHEMADERIVATIONMETHOD_SUBSTITUTION, SCHEMADERIVATIONMETHOD_EXTENSION, and SCHEMADERIVATIONMETHOD_RESTRICTION. The returned valued specifies whether an element declaration that appears in a content model will be prevented from additionally validating elements.

Validation might be prevented for one or both of the following two reasons.

- The additional element contains an xsi:type attribute that identifies an extension or restriction for the type of the declared element.
- The element is in the disallowed substitution group. This group has had a substitution performed against it using the element that has a declaration with the disallowedSubstitution attribute.

[Script]

Implementation Syntax

```javascript
var oDisallowedSubstitutions = oISchemaAttribute.disallowedSubstitut
```

Parameters

None.

Return Values

`oDisallowedSubstitutions`

An enumerated value. The result is a combination of the following values.

- SCHEMADERIVATIONMETHOD_SUBSTITUTION
- SCHEMADERIVATIONMETHOD_EXTENSION
SCHEMADERIVATIONMETHOD_RESTRICTION
SCHEMADERIVATIONMETHOD_ALL

[Visual Basic]

**Implementation Syntax**

```vbnet
oDisallowedSubstitutions = oISchemaAttribute.DisallowedSubstitutions
```

**Parameters**

None.

**Return Values**

`oDisallowedSubstitutions`
An enumerated value. The result is a combination of the following values.

- SCHEMADERIVATIONMETHOD_SUBSTITUTION
- SCHEMADERIVATIONMETHOD_EXTENSION
- SCHEMADERIVATIONMETHOD_RESTRICTION
- SCHEMADERIVATIONMETHOD_ALL

[C/C++]

**Implementation Syntax**

```c
HRESULT get_disallowedSubstitutions (SCHEMADERIVATIONMETHOD* disallo
```

**Parameters**

`disallowedSubstitutions [out,retval]`
An enumerated value. The result is a combination of the following values.

- SCHEMADERIVATIONMETHOD_SUBSTITUTION
SCHEMADERIVATIONMETHOD_EXTENSION
SCHEMADERIVATIONMETHOD_RESTRICTION
SCHEMADERIVATIONMETHOD_ALL

**Return Values**

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the disallowedSubstitution parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaItem Interface

Applies to: ISchemaElement Interface
fixedValue Property

Returns a string. This string contains the fixed value of the element being queried. The <element> declaration includes the fixed attribute.

[Script]

Implementation Syntax

var strFixedValue = oISchemaElement.fixedValue;

Parameters

None.

Return Values

strFixedValue

A string. The value of the fixed attribute in the <element> declaration.

[Visual Basic]

Implementation Syntax

strFixedValue = oISchemaElement.fixedValue

Parameters

None.

Return Values

strFixedValue

A string. The value of the fixed attribute in the <element> declaration.
[C/C++]

**Implementation Syntax**

HRESULT get_fixedValue(BSTR* fixedValue);

**Parameters**

*fixedValue [out, retval]*

A string. The value of the fixed attribute in the <element> declaration.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the fixedValue parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[element Element](#)

Applies to: [ISchemaElement Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Referen...
identityConstraints Property

Returns an ISchemaItemCollection of ISchemaIdentityConstraint objects. These objects implement the ISchemaIdentityConstraint interface.

[Script]

Implementation Syntax

```javascript
var oIdentityConstraints = oISchemaAttribute.identityConstraints;
```

Parameters

None.

Return Values

**oIdentityConstraints**
An object. The collection of ISchemaIdentityConstraint objects.

[Visual Basic]

Implementation Syntax

```vbscript
Set oIdentityConstraints = oISchemaAttribute.identityConstraints
```

Parameters

None.

Return Values

**oIdentityConstraints**
An object. The collection of ISchemaIdentityConstraint objects.
[C/C++]

**Implementation Syntax**

HRESULT get_identityConstraints(ISchemaItemCollection** identityConstraints);

**Parameters**

*identityConstraints* [out,retval]  
An object. The collection of ISchemaIdentityConstraint objects.

**Return Values**

S_OK  
The value returned if successful.
E_POINTER  
The value returned if the identityConstraints collection is NULL.
E_FAIL  
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[Schema Element]  |  [ISchemaIdentityConstraint Interface]

Applies to: [ISchemaElement Interface]
isAbstract Property

Returns a Boolean. If the value is true, the element can appear as the type definition in an element declaration, and can be referenced as an xsi:type attribute in a document. If the value is false, the element cannot appear as the type definition or be referenced as an xsi:type attribute.

[Script]

Implementation Syntax

var bolIsAbstract = oISchemaAttribute.isAbstract;

Parameters

None.

Return Values

bolIsAbstract

A Boolean. Indicates whether the schema element object is abstract.

[Visual Basic]

Implementation Syntax

bolIsAbstract = oISchemaAttribute.isAbstract

Parameters

None.

Return Values

bolIsAbstract
A Boolean. Indicates whether the schema element object is abstract.

[C/C++]

**Implementation Syntax**

HRESULT get_isAbstract(VARIANT_BOOL* isAbstract);

**Parameters**

isAbstract [out,retval]
   A Boolean. The value VARIANT_TRUE is returned if the element is abstract.

**Return Values**

S_OK
   The value returned if successful.

E_POINTER
   The value returned if the isAbstract parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaItem Interface

Applies to: ISchemaElement Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**isNillable Property**

This function retrieves the `nillable` attribute of the element. A nillable element is valid with no content, even if its content type requires content. For an element to be nillable, its declaration must have the attribute `xsd:nillable` with a value of `true`, and its instance element must have the attribute `xsi:nil` with a value of `true`. The element must be empty, but it can have attributes declared in its `<complexType>` declaration.

**Implementation Syntax**

```javascript
var bolIsNillable = oISchemaElement.isNillable;
```

**Parameters**

None.

**Return Values**

`bolIsNillable`

A Boolean. The `nillable` attribute of the element. Returns `true` if the element is nillable.

**Implementation Syntax**

```javascript
bolIsNillable = oISchemaElement.isNillable
```

**Parameters**

None.
Return Values

**bolIsNillable**
A Boolean. The nillable attribute of the element. Returns true if the element is nillable.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_isNillable(VARIANT_BOOL* isNillable);
```

**Parameters**

_isNillable_ [out,retval]
A Boolean. Returns VARIANT_TRUE if the element is nillable.

**Return Values**

S_OK
The value returned if successful.

E_POINTER
The value returned if the _isNillable_ parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

element Element

Applies to: ISchemaElement Interface
isReference Property

Returns a Boolean. If true, the schema element object is a reference to a top-level <element> declaration.

Example

The following example of an XML Schema includes some reference declarations. The VBScript code below the XML Schema retrieves isReference values from the XML Schema.

The following is an XML Schema with reference declarations.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:myschema:all" xmlns:y="urn:myschema:all">
    <xsd:complexType name="CustomerType">
        <xsd:sequence>
            <xsd:element ref="y:Customer"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Customer"/>
</xsd:schema>
```

The following is the VBScript example.

```vbnet
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
nstarget = "urn:myschema:all"
oSchemaCache.add nstarget, "cx.xml"
Set oSchema = oSchemaCache.getSchema(nstarget)
' Retrieve the reference to the <element> declaration.
Set oType = oSchema.types.itemByName("CustomerType")
Set oSequence = oType.contentModel
Set oElemRef = oSequence.particles.item(0)
res = oElemRef.name + " isRef:" & oElemRef.isReference & vbCrLf
' Retrieve the original <element> declaration.
Set oElemOrg = oSchema.elements.itemByName(oElemRef.name)
res = res + oElemOrg.name + " isRef:" & oElemOrg.isReference & vbCrLf
WScript.Echo res
```
The Echo command will show the following results.

Customer isRef: True
Customer isRef: False

[Script]

**Implementation Syntax**

```javascript
var bolIsReference = oISchemaElement.isReference;
```

**Parameters**

None.

**Return Values**

*bolIsReference*

A Boolean. Indicates whether the schema element object is a reference to a top-level `<element>` declaration.

[Visual Basic]

**Implementation Syntax**

```vbnet
bolIsReference = oISchemaElement.isReference
```

**Parameters**

None.

**Return Values**

*bolIsReference*

A Boolean. Indicates whether the schema element object is a reference to a top-level `<element>` declaration.

[C/C++]
Implementation Syntax

HRESULT get_isReference (VARIANT_BOOL* isReference);

Parameters

isReference [out,retval]
   A Boolean. Indicates whether the schema element object is a reference to a top-level <element> declaration.

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the isReference parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

element Element | Declaring Elements

Applies to: ISchemaElement Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**scope Property**

Returns an ISchemaComplexType. The complex type is a definition of the locally declared complex type. Globally declared types will return a value of NULL for the `scope` property. This property is absent in the case of declarations within named model groups.

[Script]

**Implementation Syntax**

```javascript
var oScope = oISchemaElement.scope;
```

**Parameters**

None.

**Return Values**

`oScope`

An object. A `<complexType>` declaration of the element being queried.

[Visual Basic]

**Implementation Syntax**

```vbscript
Set oScope = oISchemaAttribute.scope
```

**Parameters**

None.

**Return Values**

`oScope`
An object. A `<complexType>` declaration of the element being queried.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_scope(ISchemaComplexType** scopeType);
```

**Parameters**

`scopeType [out,retval]`

An object. A `<complexType>` declaration of the element being queried.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the `scope` parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaComplexType Interface

Applies to: ISchemaElement Interface
**substitutionGroup Property**

Returns a top-level ISchemaElement object. This object is the `<element>` declaration that the substitutionGroup attribute of the current object refers to. The substitution group provides an option for the declaration of an element. For example, if you have the following two elements, `<A name='eleA'/>` and `<B name='eleB' substitutionGroup='eleA'/>`, you could use the `<B>` element anywhere in the instance document where the `<A>` element was expected. This might work well in a `<sequence>` declaration.

**Example**

The following example shows an XML Schema document and the instance documents that would be valid for its declaration.

```xml
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <xsd:element name='eleA' type='xsd:string'/>
  <xsd:element name='eleB' substitutionGroup='eleA'/>
  <xsd:element name='eleSeq'>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref='eleA'/>
        <xsd:element ref='eleB'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

This instance document would validate:

```xml
<eleSeq>
  <eleA>a string</eleA>
  <eleB>another string</eleB>
</eleSeq>
```

This instance documents would also validate:

```xml
<eleSeq>
  <eleB>a string</eleB>
  <eleB>another string</eleB>
</eleSeq>
```
Implementation Syntax

```javascript
var oSubstitutionGroup = oISchemaAttribute.substitutionGroup;
```

Parameters

None.

Return Values

`oSubstitutionGroup`

An object. The top-level ISchemaElement object.

[Visual Basic]

Implementation Syntax

```vbnet
Set oSubstitutionGroup = oISchemaAttribute.substitutionGroup
```

Parameters

None.

Return Values

`oSubstitutionGroup`

An object. The top-level ISchemaElement object.

[C/C++]

Implementation Syntax

```c
HRESULT get_substitutionGroup(ISchemaElement** substitutionGroup);
```

Parameters
*substitutionGroup* [out,retval]
   An object. The ISchemaElement object.

**Return Values**

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the substitutionGroup parameter is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**

Applies to: ISchemaElement Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
substitutionGroupExclusions Property

Returns an enumerated value from the SCHEMADERIVATIONMETHOD declarations. The possible values are SCHEMADERIVATIONMETHOD_EXTENSION and SCHEMADERIVATIONMETHOD_RESTRICTION. The returned value is the derived-by type that will cause a substitution group affiliation to fail.

You create a substitution group affiliation by adding the substitutionGroup attribute to an <element> declaration. When the substitutionGroupExclusion attribute is set to "extension" and the affiliated element has been derived by extension, substitution will not work. When the substitutionGroupExclusion attribute is set to "restriction" and the affiliated element has been derived by restriction, substitution will not work. When both "extension" and "restriction" are set as values for the substitutionGroupExclusion attribute, no substitution is allowed for the elements in the affiliation group.

Example

The following example shows an XML Schema document that is not valid because 'eleB' cannot nominate 'eleA' as the head of its substitution group affiliation.

```xml
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <xsd:element name='eleA' type='xsd:string' final='restriction'/>
  <xsd:element name='eleB' substitutionGroup='eleA'>
    <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
        <xsd:maxLength value='20' />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name='eleSeq'>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref='eleA' />
        <xsd:element ref='eleB' />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Implementation Syntax

var enuSubstitutionGroupExclusions = oISchemaAttribute. substitution

Parameters

None.

Return Values

enuSubstitutionGroupExclusions
An enumerated value. The substitution group exclusions. The following values are possible.

SCHEMADERIVATIONMETHOD_EXTENSION
SCHEMADERIVATIONMETHOD_RESTRICTION
SCHEMADERIVATIONMETHOD_ALL

[Visual Basic]

Implementation Syntax

enuSubstitutionGroupExclusions = oISchemaAttribute. substitutionGroup

Parameters

None.

Return Values

enuSubstitutionGroupExclusions
An enumerated value. The substitution group exclusions. The following values are possible.

SCHEMADERIVATIONMETHOD_EXTENSION
SCHEMADERIVATIONMETHOD_RESTRICTION
SCHEMADERIVATIONMETHOD_ALL

[C/C++]

Implementation Syntax

HRESULT get_substitutionGroupExclusions(SCHEMADERIVATIONMETHOD* substitutionsGroupExclusions);

Parameters

substitutionGroupExclusions [out,retval]
    An enumerated value. The following values are possible.
    SCHEMADERIVATIONMETHOD_EXTENSION
    SCHEMADERIVATIONMETHOD_RESTRICTION
    SCHEMADERIVATIONMETHOD_ALL

Return Values

S_OK
    The value returned if successful.
E_POINTER
    The value returned if the substitutionsGroupExclusions parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: ISchemaElement Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**type Property**

Returns an ISchemaType object. This object contains type information about the element item being queried. You obtain further information about the type object of the element by using the ISchemaType interface.

[Script]

**Implementation Syntax**

```
var oType = oISchemaElement.type;
```

**Parameters**

None.

**Return Values**

- **oType**
  - An object. The type of the element.

[Visual Basic]

**Implementation Syntax**

```
set oType = oISchemaElement.Type
```

**Parameters**

None.

**Return Values**

- **oType**
  - An object. The type of the element.
**Implementation Syntax**

HRESULT get_type(ISchemaType** type);

**Parameters**

*type [out,retval]*
An object. A simple type definition or a complex type definition of the element.

**Return Values**

S_OK
The value returned if successful.
E_POINTER
The value returned if the *type* parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaType Interface
Applies to: ISchemaElement Interface
ISchemaIdentityConstraint Interface

Returns the ISchemaIdentityConstraint object from the identityConstraints property of the ISchemaElement interface.

Properties

- **fields**
  Retrieves a collection of restricted XPath expressions.
- **referencedKey**
  Retrieves the referenced key property.
- **selector**
  Retrieves the restricted XPath expression.

[Script]

**Implementation Syntax**

```javascript
var oSchemaIdentityConstraint = oSchemaElement.identityConstraints;
```

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oSchemaIdentityConstraint = oSchemaElement.identityConstraints
```

[C/C++]

**Implementation Syntax**

```c
HRESULT identityConstraint(ISchemaItemCollection** constraints);
```

Versioning

MSXML 4.0 and later

Requirements
[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib, msxml2.idl

**Inheritance:** Inherits from the ISchemaItem interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[ISchemaElement identityConstraints Property] | [SOM Reference]
fields Property

Returns a string collection that contains restricted XPath expressions.

[Script]

Implementation Syntax

```javascript
var oFields = oISchemaIdentityConstraint.fields;
```

Parameters

None.

Return Values

`oFields`

An object. The collection of restricted XPath expressions.

[Visual Basic]

Implementation Syntax

```visualbasic
Set oFields = oISchemaIdentityConstraint.fields
```

Parameters

None.

Return Values

`oFields`

An object. The collection of restricted XPath expressions.

[C/C++]
**Implementation Syntax**

HRESULT get_fields(ISchemaStringCollection** fields);

**Parameters**

*fields [out retval]*
   An object. The collection of restricted XPath expression strings.

**Return Values**

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the *fields* object is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

ISchemaStringCollection Interface | XPath Reference

Applies to: ISchemaIdentityConstraint Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
referencedKey Property

Returns a SchemaIdentityConstraint object. This object is the referenced key property for the SchemaIdentityConstraint being queried.

[Script]

**Implementation syntax**

```javascript
var objReferencedKey = objISchemaIdentityConstraint.referencedKey;
```

**Parameters**

None.

**Return Values**

`objReferencedKey`

An object. The referenced key property.

[Visual Basic]

**Implementation syntax**

```vbnet
Set objReferencedKey = objISchemaIdentityConstraint.referencedKey
```

**Parameters**

None.

**Return Values**

`objReferencedKey`

An object. The referenced key property.
Implementation syntax

HRESULT get_referencedKey(ISchemaIdentityConstraint** key);

Parameters

key [out,retval]
The referenced key property.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the key parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

XML Schema Elements | field Element | selector Element

Applies to: ISchemaIdentityConstraint Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
selector Property

Returns a string with the value of a restricted XPath expression.

[Script]

Implementation Syntax

var strSelector = oISchemaIdentityConstraint.selector;

Parameters

None.

Return Values

strSelector

A string. A restricted XPath expression.

[Visual Basic]

Implementation Syntax

strSelector = oISchemaIdentityConstraint.selector

Parameters

None.

Return Values

strSelector

A string. A restricted XPath expression.

[C/C++]
Implementation Syntax

HRESULT get_selector(BSTR* selector);

Parameters

selector [out,retval]
A string. An XPath expression.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the selector parameter returns NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

XPath Reference

Applies to: ISchemaIdentityConstraint Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
ISchemaItem Interface

Returns information about all the individual items in an XML Schema. The itemByName and itemByQName methods of the ISchemaItemCollection interface return the ISchemaItem objects. The information provided by the ISchemaItem interface is top-level. To obtain more information about the item, use the other interfaces in the SOM. For a list of these interfaces, see the SOM Reference overview.

Example

The following is an example of an XML Schema document with some schema items and the properties obtained from ISchemaItem.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:pub="http://www.lucernepublishing.com/fake/schema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:attribute name="booktitle"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Properties

- **id**: Retrieves the id attribute of the schema item.
- **itemType**: Retrieves the type of the schema item.
- **name**: Retrieves the name of the item. The name is an NCName as defined by XML-Namespaces.
- **namespaceURI**: Retrieves the namespace URI of the item.
- **schema**: Retrieves the parent schema of the item.
- **unhandledAttributes**: Retrieves the collection of attributes that are not defined in the schema namespace.

Methods
**writeAnnotation**

Writes an annotation to the XML Schema and to items in the XML Schema.

[Script]

**Implementation Syntax**

```javascript
var oSchemaItem = oSchemaItemCollection.item(1);
```

[Visual Basic]

**Implementation Syntax**

```vbscript
Set oSchemaItem = oSchemaItemCollection.item(1)
```

[C/C++]

**Implementation Syntax**

```
HRESULT item(long index, ISchemaItem** item);
```

**Versioning**

MSXML 4.0 and later

**Requirements**

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib, msxml2.idl

**Inheritance:** None.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

ISchemaItemCollection itemByName Method | ISchemaItemCollection itemByQName Method | Declaring Attributes | Declaring Elements | SOM Reference
**id Property**

Returns an identifier for the schema item in the form of a string. This identifier is the id attribute in the declaration of the item.

**Example**

The following is an example of the id attribute, used in an XML Schema.

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="book" id="bookID"/>
</schema>
```

[Script]

**Implementation Syntax**

```javascript
var strID = oISchemaItem.id;
```

**Parameters**

None.

**Return Values**

`strID`

A string. The id attribute of the schema item.

[Visual Basic]

**Implementation Syntax**

```vbnet
strID = oISchemaItem.id
```

**Parameters**
None.

**Return Values**

`strID`
A string. The `id` attribute of the schema item.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_id(BSTR* id);
```

**Parameters**

`id [out,retval]`
A string. The `id` attribute of the schema item.

**Return Values**

S_OK
The value returned if successful.
E_POINTER
The value returned if the `id` parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[Declaring Attributes] [Declaring Elements]

Applies to: [ISchemaItem Interface]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
itemType Property

Returns a schema type from the SOMITEMTYPE constants list. The itemType property is used to determine which SOM interface is used to get more information about the item being queried. The type will be one of the following values.

<table>
<thead>
<tr>
<th>Enumerated Type Value</th>
<th>DEC Value</th>
<th>HEX Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM_SCHEMA</td>
<td>4096</td>
<td>0x1000</td>
</tr>
<tr>
<td>SOMITEM_ATTRIBUTE</td>
<td>4097</td>
<td>0x1001</td>
</tr>
<tr>
<td>SOMITEM_ATTRIBUTEGROUP</td>
<td>4098</td>
<td>0x1002</td>
</tr>
<tr>
<td>SOMITEM_NOTATION</td>
<td>4099</td>
<td>0x1003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identity Constraints</th>
<th>DEC Value</th>
<th>HEX Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM.IDENTITYCONSTRAINT</td>
<td>4352</td>
<td>0x1100</td>
</tr>
<tr>
<td>SOMITEM_KEY</td>
<td>4353</td>
<td>0x1101</td>
</tr>
<tr>
<td>SOMITEM_KEYREF</td>
<td>4354</td>
<td>0x1102</td>
</tr>
<tr>
<td>SOMITEM_UNIQUE</td>
<td>4355</td>
<td>0x1103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types</th>
<th>DEC Value</th>
<th>HEX Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM_ANYTYPE</td>
<td>8192</td>
<td>0x2000</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE</td>
<td>8448</td>
<td>0x2100</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ANYTYPE</td>
<td>8449</td>
<td>0x2101</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ANYURI</td>
<td>8450</td>
<td>0x2102</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_BASE64BINARY</td>
<td>8451</td>
<td>0x2103</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_BOOLEAN</td>
<td>8452</td>
<td>0x2104</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_BYTE</td>
<td>8453</td>
<td>0x2105</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_DATE</td>
<td>8454</td>
<td>0x2106</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_DATETIME</td>
<td>8455</td>
<td>0x2107</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_DAY</td>
<td>8456</td>
<td>0x2108</td>
</tr>
<tr>
<td>XML DTD Type</td>
<td>Number</td>
<td>Hex Code</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_DECIMAL</td>
<td>8457</td>
<td>0x2109</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_DOUBLE</td>
<td>8458</td>
<td>0x210A</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_DURATION</td>
<td>8459</td>
<td>0x210B</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ENTITIES</td>
<td>8460</td>
<td>0x210C</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ENTITY</td>
<td>8461</td>
<td>0x210D</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_FLOAT</td>
<td>8462</td>
<td>0x210E</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_HEXBINARY</td>
<td>8463</td>
<td>0x210F</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_ID</td>
<td>8464</td>
<td>0x2110</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_IDREF</td>
<td>8465</td>
<td>0x2111</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_IDREFS</td>
<td>8466</td>
<td>0x2112</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_INT</td>
<td>8467</td>
<td>0x2113</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_INTEGER</td>
<td>8468</td>
<td>0x2114</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_LANGUAGE</td>
<td>8469</td>
<td>0x2115</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_LONG</td>
<td>8470</td>
<td>0x2116</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_MONTH</td>
<td>8471</td>
<td>0x2117</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_MONTHDAY</td>
<td>8472</td>
<td>0x2118</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NAME</td>
<td>8473</td>
<td>0x2119</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NCNAME</td>
<td>8474</td>
<td>0x211A</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NEGATIVEINTEGER</td>
<td>8475</td>
<td>0x211B</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NMTOKEN</td>
<td>8476</td>
<td>0x211C</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NMTOKENS</td>
<td>8477</td>
<td>0x211D</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NONNEGATIVEINTEGER</td>
<td>8478</td>
<td>0x211E</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NONPOSITIVEINTEGER</td>
<td>8479</td>
<td>0x211F</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NORMALIZEDSTRING</td>
<td>8480</td>
<td>0x2120</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_NOTATION</td>
<td>8481</td>
<td>0x2121</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_POSITIVEINTEGER</td>
<td>8482</td>
<td>0x2122</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_QNAME</td>
<td>8483</td>
<td>0x2123</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_SHORT</td>
<td>8484</td>
<td>0x2124</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_STRING</td>
<td>8485</td>
<td>0x2125</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_TIME</td>
<td>8486</td>
<td>0x2126</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_TOKEN</td>
<td>8487</td>
<td>0x2127</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_UNSIGNEDBYTE</td>
<td>8488</td>
<td>0x2128</td>
</tr>
<tr>
<td>SOMITEM_DATATYPE_UNSIGNEDDINT</td>
<td>8489</td>
<td>0x2129</td>
</tr>
</tbody>
</table>
SOMITEM_DATATYPE_UNSIGNEDLONG 8490 0x212A
SOMITEM_DATATYPE_UNSIGNEDSHORT 8491 0x212B
SOMITEM_DATATYPE_YEAR 8492 0x212C
SOMITEM_DATATYPE_YEARMONTH 8493 0x212D
SOMITEM_SIMPLETYPE 8704 0x2200
SOMITEM_COMPLEXTYPE 9216 0x2400
SOMITEM_PARTICLE 16384 0x4000 // particle mask
SOMITEM_ANY 16385 0x4001
SOMITEM_ANYATTRIBUTE 16386 0x4002
SOMITEM_ELEMENT 16387 0x4003
SOMITEM_GROUP 16640 0x4100 // group mask
SOMITEM_ALL 16641 0x4101
SOMITEM_CHOICE 16642 0x4102
SOMITEM_SEQUENCE 16643 0x4103
SOMITEM_EMPTYPARTICLE 16644 0x4104
SOMITEM_NULL 2048 0x0800 // null items
SOMITEMM_NULL_TYPE 10240 0x2800
SOMITEM_NULL_ANY 18433 0x4801
SOMITEM_NULL_ANYATTRIBUTE 18434 0x4802
SOMITEM_NULL_ELEMENT 18435 0x4803

[Script]

**Implementation Syntax**

```javascript
var conType = oSchemaItem.itemType;
```

**Parameters**

None.
Return Values

`conType`

An Enum of type SOMITEMTYPE. The type of the ISchemaItem object.

[Visual Basic]

Implementation Syntax

`conType = oSchemaItem.itemType`

Parameters

None.

Return Values

`conType`

An Enum of type SOMITEMTYPE. The type of the ISchemaItem object.

[C/C++]

Implementation Syntax

`HRESULT get_itemType(SOMITEMTYPE* itemType);`

Parameters

`itemType [out,retval]`

An Enum of type SOMITEMTYPE. The type of the ISchemaItem object.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the `itemType` parameter is NULL.
E_FAIL

The value returned something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: ISchemaItem Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
name Property

Returns a name for the schema item in the form of a string. The name is the schema attribute name in the declaration of the item.

Example

The following is an example of the name attribute, used in an XML Schema.

```xml
<schema>
  <element name="title"/>
</schema>
```

[Script]

Implementation Syntax

```javascript
var strName = oISchemaItem.name;
```

Parameters

None.

Return Values

`strName`
A string. The name attribute of the schema item.

[Visual Basic]

Implementation Syntax

```vbnet
strName = oISchemaItem.name
```

Parameters
Return Values

strName
A string. The name attribute of the schema item.

[C/C++]

Implementation Syntax

HRESULT get_name(BSTR* name);

Parameters

name [out,retval]
A string. The name attribute of the schema item.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the name parameter returns NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Declaring Attributes | Declaring Elements

Applies to: ISchemaItem Interface
namespaceURI Property

Returns a string that contains the value of the namespace URI for the item being queried.

Example

The following is an example of the namespace URI for an element and an attribute, used in an XML schema.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:oth="http://www.example.microsoft.com/other-schema">
    <xsd:element name="bookcategory">
        <xsd:complexType>
            <xsd:attribute name="book"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

[Script]

**Implementation Syntax**

```javascript
var strNamespaceURI = oISchemaItem.namespaceURI;
```

**Parameters**

None.

**Return Values**

`strNamespaceURI`  
A string. The namespace URI of the schema item.

[Visual Basic]

**Implementation Syntax**
strNamespaceURI = oISchemaItem.namespaceURI

Parameters

None.

Return Values

strNamespaceURI
A string. The namespace URI of the schema item.

[C/C++]

Implementation Syntax

HRESULT get_namespaceURI(BSTR* namespaceURI);

Parameters

namespaceURI [out,retval]
A string. The namespace URI of the schema item.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the URI returned is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Using Namespaces in Schemas | Using Other Namespaces
Applies to: ISchemaItem Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
schema Property

Returns an ISchema object for this property of the ISchemaItem object. The object returned is the XML Schema to which the schema item validates. The ISchema interface is used to examine further information about the object. This property will return NULL for built-in data types if the schema item has an itemType of SOMITEM_DATATYPE_XXX and a built-in complex anyType, or if the schema item is itself an ISchema object. The NULL value is returned to prevent circular referencing.

[Script]

Implementation Syntax

var oSchema = oISchemaItem.schema;

Parameters

None.

Return Values

oSchema

An object. The parent XML Schema of the item.

[Visual Basic]

Implementation Syntax

Set oSchema = oISchemaItem.schema

Parameters

None.
Return Values

\textit{oSchema}

An object. The parent XML Schema of the item.

[C/C++]

Implementation Syntax

\texttt{HRESULT get\_schema(ISchema* schema);} 

Parameters

\textit{schema} [\texttt{out}, \texttt{retval}]

An object. The parent XML Schema of the item.

Return Values

S\_OK

The value returned if successful.

E\_POINTER

The value returned if the \textit{schema} object is NULL.

E\_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

\texttt{ISchema Interface} | \texttt{Using Other Namespaces}

Applies to: \texttt{ISchemaltem Interface}
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
unhandledAttributes Property

Returns an ISchemaItemCollection object. This collection contains all the attributes that are not defined in the schema namespace.

Example

The following is an example of an unhandled attribute in a schema.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:catalog="http://www.example.microsoft.com/catalog">
    <xsd:element name="book" id="bookID" catalog:attr="isbn"/>
</xsd:schema>
```

[Script]

Implementation Syntax

```javascript
var oAttributes = oISchemaItem.unhandledAttributes;
```

Parameters

None.

Return Values

`oAttributes`

An object. The collection of attributes that are not defined in the schema namespace.

[Visual Basic]

Implementation Syntax

```vbnet
Set oAttributes = oISchemaItem.unhandledAttributes
```
Parameters

None.

Return Values

$oAttributes$
An object. The collection of attributes that are not defined in the schema namespace.

[C/C++]

Implementation Syntax

HRESULT get_unhandledAttributes(ISAXAttributes** attributes);

Parameters

$attributes$ [out,retval]
An object. The collection of attributes that are not defined in the schema namespace.

Return Values

S_OK
The value returned if no errors are reported.
E_FAIL
The value returned if something is wrong.

Remarks

The return value of this property is a collection of unhandled attributes of the ISAXAttributes type.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

Using Other Namespaces | ISAXAttributes Interface

Applies to: ISchemaItem Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
writeAnnotation Method

Writes a top-level annotation or an annotation for an XML Schema `<element>` declaration. The following objects can have annotations written to them: `IXMLDOMDocument`, `IXMLDOMNode`, `IVBSAXContentHandler`, and `ISAXContentHandler`. Any one of these four object types can be passed in the `annotationSink` parameter.

If the object passed in the `annotationSink` parameter is an `IXMLDOMDocument` (schema) object, the annotation for the item is written to the top-level `<annotations>` for the XML Schema. If a top-level annotation exists, an `<annotations>` root element is created, and all `<annotation>` declarations are located inside it.

For the other three object types, the `<annotation>` node of the object that is passed in is considered the root node. The DOMDocument will have the annotation node appended to the IXMLDOMNode.

Example

The `po.xsd` XML Schema that is referred to in these examples can be found in XML Schema Examples.

The following VBScript example shows how to write a top-level schema annotation into a DOM document.

```vbnet
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")
oSchemaCache.add ",", "po.xsd"
Set oSchema = oSchemaCache.getSchema(""
oSchemas.writeAnnotation(oAnnotationDoc)
WScript.Echo oAnnotationDoc.documentElement.xml
```

The following VBScript example shows how to write an annotation for a schema item into a DOM document.

```vbnet
Set oSchemaCache = CreateObject("Msxml2.XMLSchemaCache.5.0")
Set oAnnotationDoc = CreateObject("Msxml2.DOMDocument.5.0")
oSchemaCache.add ",", "po.xsd"
```
Set oSchema = oSchemaCache.getSchema("")
Set ua = oSchema.types.itemByName("USAddress")
ua.writeAnnotation(oAnnotationDoc)
WScript.Echo oAnnotationDoc.documentElement.xml

[Script]

**Implementation Syntax**

```javascript
var bolIsWritten = oISchemaItem.annotation(oAnnotationSink);
```

**Parameters**

`oAnnotationSink`  
An object. A pointer to the annotation target object. The following interfaces can be sent: IXMLDOMDocument, IXMLDOMNode, IVBSAXContentHandler, and ISAXContentHandler.

**Return Values**

`bolIsWritten`  
A Boolean value. True if there is any annotation information for this schema item. False if there are no annotations for this schema item.

[Visual Basic]

**Implementation Syntax**

```vbnet
bolIsWritten = oISchemaItem.annotation(oAnnotationSink);
```

**Parameters**

`oAnnotationSink`  
An object. A pointer to the annotation target object. The following interfaces can be sent: IXMLDOMDocument, IXMLDOMNode, IVBSAXContentHandler, and ISAXContentHandler.

**Return Values**
**bolIsWritten**

A Boolean value. True if there is any annotation information for this schema item. False if there are no annotations for this schema item.

[C/C++]

**Implementation Syntax**

```c
HRESULT writeAnnotation(IUnknown* annotationSink,
VARIANT_BOOL* isWritten);
```

**Parameters**

- **annotationSink [in]**
  A pointer to the annotation target object. The following interfaces can be sent: IXMLDOMDocument, IXMLDOMNode, IVBSAXContentHandler, and ISAXContentHandler.

- **isWritten [out, retval]**
  A Boolean value. VARIANT_TRUE if there is any annotation information for this schema item. VARIANT_FALSE if there are no annotations for this schema item.

**Return Values**

- **S_OK**
  The value returned if successful.

- **E_INVALIDARG**
  The value returned if the annotationSink is not one of the following types: XMLDOMDocument, IXMLDOMNode, IVBSAXContentHandler, or ISAXContentHandler.

- **E_POINTER**
  The value returned if the isWritten parameter is NULL.

- **E_FAIL**
  The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

annotation Element

Applies to: ISchemaItem Interface
ISchemaItemCollection Interface

Stores collections of objects returned from various properties of the SOM interfaces. You can use the methods of the ISchemaItemCollection interface to search and browse these collections.

Different types of objects are returned for different interfaces. For example, elements return objects that invoke the ISchemaElement interface, attributes return objects that invoke the ISchemaAttribute interface, and so on.

The following is a list of the interface property for each property that returns an item collection.

<table>
<thead>
<tr>
<th>Property</th>
<th>Collection contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISchema.types</td>
<td>types</td>
</tr>
<tr>
<td>ISchema.elements</td>
<td>elements</td>
</tr>
<tr>
<td>ISchema.attributes</td>
<td>attributes</td>
</tr>
<tr>
<td>ISchema.attributeGroups</td>
<td>attribute groups</td>
</tr>
<tr>
<td>ISchema.modelGroups</td>
<td>model groups</td>
</tr>
<tr>
<td>ISchema.notations</td>
<td>notations</td>
</tr>
<tr>
<td>ISchemaAttributeGroup.attributes</td>
<td>attributes</td>
</tr>
<tr>
<td>ISchemaElement.identityConstraints</td>
<td>constraints</td>
</tr>
<tr>
<td>ISchemaItem.baseTypes</td>
<td>base types</td>
</tr>
<tr>
<td>ISchemaComplexType.attributes</td>
<td>attributes</td>
</tr>
<tr>
<td>ISchemaModelGroup.particles</td>
<td>particles</td>
</tr>
</tbody>
</table>

Properties

- **item**: Retrieves ISchemaItem objects by the index.
- **length**: Retrieves the number of schema items in the collection.

Methods
itemByName

Retrieves ISchemaItem objects by name.

itemByQName

Retrieves ISchemaItem objects by name, along with the specified QName.

[Script]

**Implementation Syntax**

```javascript
var oISchemaElements = ISchema.elements; or
var oISchemaIdentityConstraint = ISchemaElement.identityConstraints;
```

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oISchemaElements = ISchema.elements or
Set oISchemaIdentityConstraint = ISchemaElement.identityConstraints
```

[C/C++]

**Implementation Syntax**

```c
HRESULT get_elements (ISchemaItemCollection** elements); or
HRESULT identityConstraints(ISchemaItemCollection** identityConstraints);
```

**Versioning**

MSXML 4.0 and later

**Requirements**

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib, msxml2.idl
Inheritance: None

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaItem Interface | SOM Reference
item Property

Returns an ISchemaItem object, based on the index that is passed to the function. You can use this method to walk through a collection and examine properties for each item. To walk through the collection, increment the index from 0 to one less than the number returned from the length property.

The ISchemaItem.itemType property is used to obtain the object type for the items in the collection.

The appropriate interface is used to obtain further information about the items in the collection. See the table in the ISchemaItemCollection Interface topic for a list of the interfaces used.

[Script]

Implementation Syntax

```javascript
var oISchemaItem = oISchemaItemCollection.item(lngIndex)
```

Parameters

`lngIndex`
A long integer. The index of the item to be retrieved.

Return Values

`oISchemaItem`
An ISchemaItem object. The schema item at the specified index.

[Visual Basic]

Implementation Syntax

```vbnet
Set oISchemaItem = oISchemaItemCollection.item(lngIndex)
```
Parameters

\textit{lngIndex}
A long integer. The index of the item to be retrieved.

Return Values

\textit{oISchemaItem}
An ISchemaItem object. The schema item at the specified index.

[C/C++]

Implementation Syntax

\texttt{HRESULT get\_item(}
\texttt{long index,}
\texttt{ISchemaItem** item);}

Parameters

\textit{index} [in]
A long integer. The index of the item to be retrieved.
\textit{item} [out,retval]
An object. The schema item at the specified index.

Return Values

\texttt{S\_OK}
The value returned if successful.
\texttt{E\_POINTER}
The value returned if the \textit{item} parameter is NULL.
\texttt{E\_FAIL}
The value returned if the index is out of bounds.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button \[\text{ ]}\ in the upper-left corner of the page.

See Also
ISchemaItem Interface | ISchemaItem itemType Property

Applies to: ISchemaItemCollection Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
length Property

Returns a long integer. The number of items in the ISchemaItemCollection object. The index values for the collection start at "0" and go to the number that is the value of the length property minus 1.

[Script]

Implementation Syntax

var lngCount = oISchemaItemCollection.length

Parameters

None.

Return Values

lngCount

A long integer. The number of items in the schema item collection.

[Visual Basic]

Implementation Syntax

lngCount = oISchemaItemCollection.length

Parameters

None.

Return Values

lngCount

A long integer. The number of items in the schema item collection.
[C/C++]

Implementation Syntax

HRESULT get_length(long* length);

Parameters

length [out,retval]
A long integer. The number of items in the collection.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the length parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaItemCollection Item Method

Applies to: ISchemaItemCollection Interface
itemByKeyName Method

Returns an ISchemaItem object. The collection is searched to find an ISchemaItem that has a name property in the targetNamespace of the schema that matches the name parameter sent to the method. The returned object contains information about the item name that is sent to the function in the name parameter. The ISchemaItem interface is used to derive further information from the returned item. For more information about ISchemaItem objects, see ISchemaItem Interface.

[Script]

Implementation Syntax

```javascript
var oISchemaItem = oISchemaItemCollection.itemByName(strName)
```

Parameters

strName

A string. The name of the schema item to be retrieved.

Return Values

oISchemaItem

An ISchemaItem object. The schema item at the specified index.

[Visual Basic]

Implementation Syntax

```vbnet
Set oISchemaItem = oISchemaItemCollection.itemByName(strName)
```

Parameters

strName
A string. The name of the schema item to be retrieved.

Return Values

$oISchemaItem$

An ISchemaItem object. The schema item at the specified index.

[C/C++]

Implementation Syntax

HRESULT get_itemByName(
    BSTR name,
    ISchemaItem** item);

Parameters

$name$ [in]

A string. The name of the schema item to be retrieved.

$item$ [out,retval]

An object. The schema item at the specified index.

Return Values

S_OK

The value returned if successful.

E_INVALIDARG

The value returned if name is NULL.

E_POINTER

The value returned if item is NULL.

E_FAIL

The value returned if the named item is not found.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
ISchemaItem Interface | ISchemaItem itemType Property | ISchemaItem Name Property

Applies to: ISchemaItemCollection Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
itemByQName Method

Returns an ISchemaItem object. The returned object contains information about the item name and the corresponding namespace that are sent to the function in the name and namespaceURI parameters. The ISchemaItem interface obtains further information from the returned object.

[Script]

Implementation Syntax

var oISchemaItem = oISchemaItemCollection.itemByQName(strName, strNamespaceURI)

Parameters

strName
A string. The name of the schema item to be retrieved.

strNamespaceURI
A string. The name of the namespace URI that the item to be retrieved belongs to.

Return Values

oISchemaItem
An ISchemaItem object. The schema item that has the same name and namespace URI that are passed in through the parameters.

[Visual Basic]

Implementation Syntax

Set oISchemaItem = oISchemaItemCollection.itemByQName(strName, strNamespaceURI)

Parameters
strName
   A string. The name of the schema item to be retrieved.

strNamespaceURI
   A string. The name of the namespace URI that the item to be retrieved belongs to.

**Return Values**

oISchemaItem
   An ISchemaItem object. The schema item that has the same name and namespace URI that are passed in through the parameters.

[C/C++]

**Implementation Syntax**

HRESULT get_itemByQName(
   BSTR name,
   BSTR namespaceURI,
   ISchemaItem** item);

**Parameters**

name [in]
   A string. The name of the schema item to be retrieved.

namespaceURI [in]
   A string. The name of the namespace URI that the item to be retrieved belongs to.

item [out,retval]
   An ISchemaItem object. The schema item that has the same name and namespace URI that are passed in through the parameters.

**Return Values**

S_OK
   The value returned if successful.

E_INVALIDARG
   The value returned if the name parameter returns NULL.
E_POINTER
    The value returned if the item parameter returns NULL.
E_FAIL
    The value returned if the named item is not found.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaItem Interface | ISchemaItem Type Property | Using Namespaces in Schemas

Applies to: ISchemaItemCollection Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
ISchemaModelGroup Interface

Provides a collection of some combination of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny. The ISchemaModelGroup object is returned from the modelGroups property of the ISchema interface or from the contentModel property of the ISchemaComplexType interface.

The types of items in the collection are specified in the itemType property of the ISchemaItem interface. The following table shows each itemType and its corresponding collection content.

<table>
<thead>
<tr>
<th>ISchema.itemType value</th>
<th>Content objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM_ALL</td>
<td>A collection of one or more ISchemaElement objects. A collection of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny. Each of these objects can occur zero or more times in the collection.</td>
</tr>
<tr>
<td>SOMITEM_CHOICE</td>
<td>A collection of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny. Each of these objects can occur zero or more times in the collection.</td>
</tr>
<tr>
<td>SOMITEM_SEQUENCE</td>
<td>An empty particle which contains a collection of zero particles. ISchemaComplexType.contentModel will return an empty particle when ISchemaComplexType.contentType = SCHEMACONTENTTYPE_EMPTY.</td>
</tr>
<tr>
<td>SOMITEM_EmptyParticle</td>
<td>Retrieves the group model, based on the schema item type. This type is retrieved by calling particles</td>
</tr>
</tbody>
</table>
ISchemaItem::itemType.

[Script]

**Implementation Syntax**

```javascript
var oSchemaModelGroups = oSchema.modelGroups
var oSchemaModelGroup = oSchemaModelGroups.item(index);
```

**Parameters**

`index`

A long integer. An index of the item within the collection. The first item is zero.

[Visual Basic]

**Implementation Syntax**

```vbs
Set oSchemaModelGroups = oSchema.modelGroups
Set oSchemaModelGroup = oSchemaModelGroups.item(index)
```

**Parameters**

`index`

A long integer. An index of the item within the collection. The first item is zero.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_modelGroups(ISchemaItemCollection** particles);
HRESULT particles->get_item(long index, ISchemaModelGroup** modelGro
```

**Parameters**

`index [in]`
A long integer. An index of the item within the collection. The first item is zero.

*particles* [out,retval]

An object. The ISchemaItemCollection object of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny, if any.

**Versioning**

MSXML 4.0 and later

**Requirements**

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib, msxml2.idl

**Inheritance:** Inherits from the ISchemaParticle interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button [ ] in the upper-left corner of the page.

**See Also**

ISchemaElement Interface | ISchemaComplexType contentModel Property | ISchemaAny Interface | ISchemaItem itemType Property | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
particles Property

Returns a collection of some combination of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny.

The types of items in the collection are specified in the itemType property of the ISchemaItem interface. The following table shows each itemType and its corresponding collection content.

<table>
<thead>
<tr>
<th>ISchema.itemType value</th>
<th>Content objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMITEM_ALL</td>
<td>A collection of one or more ISchemaElement objects.</td>
</tr>
<tr>
<td></td>
<td>A collection of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny. Each of these objects can occur zero or more times in the collection.</td>
</tr>
<tr>
<td>SOMITEM_CHOICE</td>
<td>ISchemaModelGroup, and ISchemaAny.</td>
</tr>
<tr>
<td></td>
<td>Each of these objects can occur zero or more times in the collection.</td>
</tr>
<tr>
<td>SOMITEM_SEQUENCE</td>
<td>ISchemaModelGroup, and ISchemaAny.</td>
</tr>
<tr>
<td></td>
<td>Each of these objects can occur zero or more times in the collection.</td>
</tr>
</tbody>
</table>

[Script]

Implementation Syntax

```javascript
var oParticles = oISchemaModelGroup.particles;
```

Parameters

None.

Return Values
**oParticles**

An object. The ISchemaItemCollection object of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny, if any.

[Visual Basic]

**Implementation Syntax**

Set oParticles = oISchemaModelGroup.particles

**Parameters**

None.

**Return Values**

**oParticles**

An object. The ISchemaItemCollection object of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny, if any.

[C/C++]

**Implementation Syntax**

HRESULT get_particles(ISchemaItemCollection** particles);

**Parameters**

particles [out,retval]

An object. The ISchemaItemCollection object of one or more of the following objects: ISchemaElement, ISchemaModelGroup, and ISchemaAny, if any.

**Return Values**
S_OK
   The value returned if successful.
E_POINTER
   The value returned if the particles object is NULL
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaElement Interface | ISchemaComplexType contentModel Property | ISchemaAny Interface | ISchemaItem itemType Property

Applies to: ISchemaModelGroup Interface
ISchemaNotation Interface

Provides the URIs for the execution programs that are declared in a <notation> declaration. A notation object is returned from the notations property of the ISchema interface.

Properties

publicIdentifier  Retrieves the public identifier URI.

systemIdentifier  Retrieves the system identifier URI.

[Script]

var oNotations = oSchema.notations;
var oNotation = oNotations(index);

Parameters

index  A long integer. An index of the item within the collection. The first item is zero.

[Visual Basic]

Set oNotations = oSchema.notations
Set oNotation = oNotations(index)

Parameters

index  A long integer. An index of the item within the collection. The first item is zero.

[C/C++]

HRESULT get_notations(ISchemaItemCollection** notations);
HRESULT notations->get_item(long index, ISchemaNotation** notation);
Parameters

index
A long integer. An index of the item within the collection. The first item is zero.

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaItem interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Notation Declarations | notation Element | startDTD Method | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Refer
publicIdentifier Property

Returns a string that contains a URI.

[Script]

Implementation Syntax

var strPublicIdentifier = oISchemaIdentityConstraint.publicIdentifier;

Parameters

None.

Return Values

strPublicIdentifier
A string. The public identifier URI.

[Visual Basic]

Implementation Syntax

strPublicIdentifier = oISchemaIdentityConstraint.publicIdentifier

Parameters

None.

Return Values

strPublicIdentifier
A string. The public identifier URI.
Implementation Syntax

HRESULT get_publicIdentifier(BSTR* publicIdentifier);

Parameters

publicIdentifier [out,retval]
   A string. A URI reference.

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the value of the URI is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Using Namespaces in Schemas

Applies to: ISchemaNotation Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM References
systemIdentifier Property

Returns a string that contains a URI.

[Script]

Implementation Syntax

```javascript
var strSystemIdentifier = oISchemaIdentityConstraint.systemIdentifier;
```

Parameters

None.

Return Values

`strSystemIdentifier`

A string. The system identifier URI.

[Visual Basic]

Implementation Syntax

```vbnet
strSystemIdentifier = oISchemaIdentityConstraint.systemIdentifier
```

Parameters

None.

Return Values

`strSystemIdentifier`

A string. The system identifier URI.

[C/C++]
Implementation Syntax

HRESULT get_systemIdentifier(BSTR* systemIdentifier);

Parameters

systemIdentifier [out,retval]
    A string. A URI reference.

Return Values

S_OK
    The value returned if successful.
E_POINTER
    The value returned if the value of the URI is NULL.
E_FAIL
    The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Using Namespaces in Schemas

Applies to: ISchemaNotation Interface
ISchemaParticle Interface

Retrieves cardinality information about the item. The two attributes that represent cardinality are \texttt{minOccurs} and \texttt{maxOccurs}. These attributes indicate the number of times that an item can occur in a document. You obtain a collection of \texttt{ISchemaParticle} objects by using the \texttt{particles} property of the \texttt{ISchemaModelGroup} interface.

Properties

\begin{itemize}
  \item \texttt{maxOccurs} \hspace{1cm} Retrieves the maximum number of times this schema item can occur in the current context. Zero is returned if the item cannot occur. Unbounded (-1) is returned if there is no limit to the number of occurrences.
  \item \texttt{minOccurs} \hspace{1cm} Retrieves the minimum number of times this schema item can occur in the current context. Zero is returned if the item is optional.
\end{itemize}

[Script]

\textbf{Implementation Syntax}

\begin{verbatim}
var oParticles = oModelGroup.particles;
\end{verbatim}

[Visual Basic]

\textbf{Implementation Syntax}

\begin{verbatim}
Set oParticles = oModelGroup.particles
\end{verbatim}

[C/C++]

\textbf{Implementation Syntax}

\begin{verbatim}
HRESULT particles(ISchemaItemCollection** particles);
\end{verbatim}
Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaItem interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Remark

The interfaces ISchemaAny, ISchemaElement, and ISchemaModelGroup are derived from the ISchemaParticle interface. This inheritance provides access to the minOccurs and maxOccurs properties for each of the derived interfaces.

The anyAttribute property of the interfaces ISchemaComplexType and ISchemaAttributeGroup returns an SchemaAny object. The inherited properties minOccurs and maxOccurs are not supported for the ISchemaAny interface when it represents an <anyAttribute> element. Therefore, the return values will always be minOccurs = 0 and maxOccurs = -1 (unbounded).

See Also

ISchemaModelGroup Interface | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
maxOccurs Property

Returns a variant of type VT_BSTR that indicates the maximum number of times an item can occur in the current context of a document. If MaxOccurs returns 0, the schema item should not appear. If maxOccurs returns UNBOUNDED(-1), the number of appearances of the schema item is unlimited.

Example

The following is an example of the maxOccurs attribute used in a schema.

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <complexType name="someelement">
        <choice>
            <element name="book" minOccurs="1" maxOccurs="4"/>
            <element name="magazine" minOccurs="1" maxOccurs="1"/>
        </choice>
    </complexType>
</schema>
```

[Script]

Implementation Syntax

```javascript
var varMaxOccurs = oISchemaParticle.maxOccurs;
```

Parameters

None.

Return Values

`varMaxOccurs`

A variant. The maximum number of times this item can appear in an instance document of this XML Schema document. If the maxOccurs property is unbounded (-1), there is no limit to the number of occurrences.
[Visual Basic]

**Implementation Syntax**

`varMaxOccurs = oISchemaParticle.maxOccurs`

**Parameters**

None.

**Return Values**

`varMaxOccurs`

A variant. The maximum number of times this item can appear in an instance document of this XML Schema document. If the `maxOccurs` property is unbounded (-1), there is no limit to the number of occurrences.

[C/C++]

**Implementation Syntax**

`HRESULT get_maxOccurs(VARIANT* maxOccurs);`

**Parameters**

`maxOccurs` [out,retval]

A variant. The maximum number of times this item can occur in an instance document of this XML Schema document.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the `maxOccurs` parameter is NULL.

To view reference information for Visual Basic or C/C++ only, click the
Language Filter button ▼ in the upper-left corner of the page.

See Also

ISchemaParticle minOccurs Property | element Element

Applies to: ISchemaParticle Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
minOccurs Property

Returns a variant of type VT_BSTR that represents the minimum number of times that an item can occur in a document. A value of zero indicates that the item is optional.

Example

The following is an example of the minOccurs attribute used in an XML Schema.

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="someelement">
    <choice>
      <element name="book" minOccurs="1" maxOccurs="4"/>
      <element name="magazine" minOccurs="1" maxOccurs="1"/>
    </choice>
  </complexType>
</schema>
```

[Script]

Implementation Syntax

```javascript
var varMinOccurs = oISchemaParticle.minOccurs;
```

Parameters

None.

Return Values

`varMinOccurs`

A variant. The minimum number of times the item can appear in an instance document of this XML Schema document. Zero is returned if the item is optional.
**Implementation Syntax**

VarMinOccurs = oISchemaParticle.minOccurs

**Parameters**

None.

**Return Values**

**VarMinOccurs**

A variant. The minimum number of times the item can appear in an instance document of this XML Schema document. Zero is returned if the item is optional.

[C/C++]

**Implementation Syntax**

HRESULT get_minOccurs(VARIANT* minOccurs);

**Parameters**

**minOccurs** [out,retval]

A variant. The minimum number of times the item can appear in an instance document of this XML Schema document. Zero is returned if the item is optional.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the minOccurs parameter is NULL.
To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaParticle maxOccurs Property | element Element

Applies to: ISchemaParticle Interface
ISchemaStringCollection Interface

Provides an object to store an indexed string collection. The string collection is used to store fields, schemaLocations values (URIs), namespaces, or enumerations. The following table lists properties and the types of items they return in their string collections.

<table>
<thead>
<tr>
<th>Property</th>
<th>Items returned in string collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISchemaIdentityConstraint.fields</td>
<td>fields</td>
</tr>
<tr>
<td>ISchema.schemaLocations</td>
<td>schema namespace URIs</td>
</tr>
<tr>
<td>ISchemaType.enumeration</td>
<td>enumerations</td>
</tr>
<tr>
<td>ISchemaAny.namespaces</td>
<td>namespaces</td>
</tr>
</tbody>
</table>

For more information about each of these properties, see the sections for the ISchemaIdentityConstraint, ISchema, ISchemaType, and ISchemaAny interfaces.

Properties

- **item**
  - Retrieves the strings for the index sent to this function.

- **length**
  - Retrieves the number of strings in the collection.

[Script]

**Implementation Syntax**

```javascript
var oSchemaLocations = oISchema.schemaLocations;
```

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oSchemaLocations = oISchema.schemaLocations
```
Implementation Syntax

HRESULT schemaLocations (ISchemaStringCollection** schemaLocations);

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: None.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaIdentityConstraint Interface | ISchema Interface | schemaLocations Property | ISchemaType Interface | ISchemaAny Interface | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Referenc
**item Property**

Returns a string that provides the value of an item. The location of the item is specified in the index location, which is sent to the `item` method in the `index` parameter. The item can be an enumeration, a schema URI, a namespace, or a field, depending on the method and interface used. You can use this method to walk through a collection and examine properties for each item. To walk through the collection, increment the index from 0 to one less than the number returned from the length property.

[Script]

**Implementation Syntax**

```javascript
var strString = oISchemaItemCollection.item(lngIndex);
```

**Parameters**

*lngIndex*

A long integer. The index of the item to be retrieved.

**Return Values**

*strString*

A string. The string at the specified index.

[Visual Basic]

**Implementation Syntax**

```visualbasic
strString = oISchemaItemCollection.item(lngIndex)
```

**Parameters**

*lngIndex*
A long integer. The index of the item to be retrieved.

**Return Values**

*strString*

A string. The string at the specified index.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_item(
    [in] long index,
    [out,retval] BSTR* bstr);
```

**Parameters**

*index [in]*

A long integer. The index of the item to be retrieved.

*bstr [out,retval]*

A string. The string at the specified index.

**Return Values**

*S_OK*

The value returned if successful.

*E_POINTER*

The value returned if the `item` parameter is NULL.

*E_FAIL*

The value returned if the index is out of bounds.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

*ISchemaStringCollection length Property*
Applies to: ISchemaStringCollection Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**length Property**

Returns a long integer that indicates the number of items in the collection.

[Script]

**Implementation Syntax**

```javascript
var lngLength = oISchemaStringCollection.length;
```

**Parameters**

None.

**Return Values**

`lngLength`

A long integer. The number of items in the schema string collection.

[Visual Basic]

**Implementation Syntax**

```vbnet
lngLength = oISchemaStringCollection.length
```

**Parameters**

None.

**Return Values**

`lngLength`

A long integer. The number of items in the schema string collection.

[C/C++]
Implementation Syntax

HRESULT get_length(long* length);

Parameters

length [out,retval]
    A long integer. The number of strings in the collection.

Return Values

S_OK
    The value returned if successful.
E_FAIL
    The value returned if length is NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISchemaStringCollection item Method

Applies to: ISchemaStringCollection Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
# ISchemaType Interface

Provides access to basic type information, and to a DOM representation of the type object. You obtain an ISchemaType object by using the types property of the ISchema interface, the type property of the ISchemaAttribute interface, and the type property of the ISchemaElement interface.

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseTypes</td>
<td>Retrieves the collection of the base-types of the type.</td>
</tr>
<tr>
<td>derivedBy</td>
<td>Retrieves the method by which the type definition was derived.</td>
</tr>
<tr>
<td>enumeration</td>
<td>Retrieves the enumeration facet of the restriction.</td>
</tr>
<tr>
<td>final</td>
<td>Retrieves the final value of the type definition.</td>
</tr>
<tr>
<td>fractionDigits</td>
<td>Retrieves the fractionDigits facet of the restriction.</td>
</tr>
<tr>
<td>length</td>
<td>Retrieves the length facet of the restriction.</td>
</tr>
<tr>
<td>maxExclusive</td>
<td>Retrieves the maxExclusive facet of the restriction.</td>
</tr>
<tr>
<td>maxInclusive</td>
<td>Retrieves the maxInclusive facet of the restriction.</td>
</tr>
<tr>
<td>maxLength</td>
<td>Retrieves the maxLength facet of the restriction.</td>
</tr>
<tr>
<td>minExclusive</td>
<td>Retrieves the minExclusive facet of the restriction.</td>
</tr>
<tr>
<td>minInclusive</td>
<td>Retrieves the minInclusive facet of the restriction.</td>
</tr>
<tr>
<td>minLength</td>
<td>Retrieves the minLength facet of the restriction.</td>
</tr>
<tr>
<td>patterns</td>
<td>Retrieves a string collection that contains the pattern facets of the restriction.</td>
</tr>
<tr>
<td>totalDigits</td>
<td>Retrieves the totalDigits facet of the restriction.</td>
</tr>
<tr>
<td>variety</td>
<td>Retrieves the value of the variety attribute for the type definition.</td>
</tr>
<tr>
<td>whitespace</td>
<td>Retrieves the whitespace facet of the restriction.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isValid</td>
<td>Checks whether the value that has been passed in is a valid instance of this type.</td>
</tr>
</tbody>
</table>
[Script]

**Implementation Syntax**

```javascript
var oSchemaTypes = oSchema.types;
var oSchemaType = oSchemaTypes.item(index)
```

**Parameters**

`index`

A long integer. An index of the item within the collection. The first item is zero.

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oSchemaTypes = oSchema.types
Set oSchemaType = oSchemaTypes.item(index)
```

**Parameters**

`index`

A long integer. An index of the item within the collection. The first item is zero.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_types(ISchemaItemCollection** types);
HRESULT types->get_item(long index, ISchemaType** type);
```

**Parameters**

`index`

A long integer. An index of the item within the collection. The first item is zero.
Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib, msxml2.idl

Inheritance: Inherits from the ISchemaItem interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ❯ in the upper-left corner of the page.

See Also

ISchema types Property | ISchemaAttribute type Property | ISchemaElement type Property | Parts of Data Types | Data Type Facets | SOM Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
baseTypes Property

Returns an ISchemaItemCollection object that contains the base types of the type object being queried. The base types are in a schema item collection and are accessed by the ISchemaType interface.

[Script]

Implementation Syntax

var oBaseTypes = oISchemaType.baseTypes;

Parameters

None.

Return Values

oBaseTypes
An object. The collection of base types of the type.

[Visual Basic]

Implementation Syntax

Set oBaseTypes = oISchemaType.baseTypes

Parameters

None.

Return Values

oBaseTypes
An object. The collection of base types of the type.
[C/C++]

**Implementation Syntax**

HRESULT get_baseTypes(ISchemaItemCollection** baseTypes);

**Parameters**

`baseTypes [out,retval]`

An object. The collection of base types of the type.

**Return Values**

S_OK
The value returned if successful.

E_POINTER
The value returned if the baseTypes object returned is NULL.

E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ![filter](image) in the upper-left corner of the page.

**See Also**

[Primitive and Derived Data Types](#)

Applies to: [ISchemaType Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**derivedBy Property**

Returns a SCHEMADERIVATIONMETHOD constant. This constant indicates how the type definition was derived.

[Script]

**Implementation syntax**

```javascript
var conDerivedBy = objISchemaType.derivedBy;
```

**Parameters**

None.

**Return Values**

`conDerivedBy`

For simple types, the derivedBy value can be one of the following:

- SCHEMADERIVATIONMETHOD_RESTRICTION
- SCHEMADERIVATIONMETHOD_LIST
- SCHEMADERIVATIONMETHOD_UNION

For complex types, the derivedBy value can be one of the following:

- SCHEMADERIVATIONMETHOD_EXTENSION
- SCHEMADERIVATIONMETHOD_RESTRICTION

For any-type the derivedBy value is always as follows:

- SCHEMADERIVATIONMETHOD_EMPTY

[Visual Basic]
**Implementation syntax**

```c
conDerivedBy = objISchemaType.derivedBy
```

**Parameters**

None.

**Return Values**

`conDerivedBy`

For simple types, the `derivedBy` value can be one of the following:

- `SCHEMADERIVATIONMETHOD_RESTRICTION`
- `SCHEMADERIVATIONMETHOD_LIST`
- `SCHEMADERIVATIONMETHOD_UNION`

For complex types, the `derivedBy` value can be one of the following:

- `SCHEMADERIVATIONMETHOD_EXTENSION`
- `SCHEMADERIVATIONMETHOD_RESTRICTION`

For any type, the `derivedBy` value is always as follows:

- `SCHEMADERIVATIONMETHOD_EMPTY`

**[C/C++]**

**Implementation syntax**

```c
HRESULT get_derivedBy(SCHEMADERIVATIONMETHOD* derivedBy);
```

**Parameters**

`derivedBy [out,retval]`

For simple types, the `derivedBy` value can be one of the following:
For complex types, the derivedBy value can be one of the following:

- SCHEMADERIVATIONMETHOD_EXTENSION
- SCHEMADERIVATIONMETHOD_RESTRICTION

For any-type the derivedBy value is always as follows:

- SCHEMADERIVATIONMETHOD_EMPTY

## Return Values

- **S_OK**
  - The value returned if successful.
- **E_POINTER**
  - The value returned if the derivedBy parameter returns NULL.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

## See Also

- [Primitive and Derived Data Types](#)
- [Parts of Data Types](#)

Applies to: [ISchemaType Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**enumeration Property**

Returns a string collection. This collection contains the enumeration values that can be used for the type. For more information about the enumeration facet, see [Data Type Facets](#).

[Script]

**Implementation Syntax**

```javascript
var oEnumeration = oISchemaType.enumeration;
```

**Parameters**

None.

**Return Values**

*oEnumeration*  
An object. The enumeration facet of the restriction. This facet is a collection of BSTRs.

[Visual Basic]

**Implementation Syntax**

```vbnet
Set oEnumeration = oISchemaType.enumeration
```

**Parameters**

None.

**Return Values**

*oEnumeration*
An object. The enumeration facet of the restriction. This facet is a collection of BSTRs.

[C/C++]

**Implementation Syntax**

```c
HRESULT get_enumeration(ISchemaStringCollection** enumeration);
```

**Parameters**

*enumeration* [out, retval]

An object. The collection of BSTRs.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the `minExclusive` parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note** If the facet is not used, the property will return an empty collection.

**See Also**

[Data Type Facets](#) | [Parts of Data Types](#)

Applies to: [ISchemaType Interface](#)
**final Property**

Returns a constant value in the form of a string. This string consists of a combination of the values in the table below. The values in the simple types and complex types columns indicate the method by which a `simpleType` or a `complexType` cannot be derived if it is to be used as a base type for another data type.

The final value can be any combination of the following types:

<table>
<thead>
<tr>
<th>Simple Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMADERIVATIONMETHOD_EXTENSION</td>
<td>SimpleType cannot be used as a base type if derived by extension.</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_LIST</td>
<td>SimpleType cannot be used as a base type if derived by list.</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_UNION</td>
<td>SimpleType cannot be used as a base type if derived by union.</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_RESTRICTION</td>
<td>SimpleType cannot be used as a base type if derived by restriction.</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_ALL</td>
<td>SimpleType cannot be used as a base type.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complex types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMADERIVATIONMETHOD_EXTENSION</td>
<td>ComplexType cannot be used as a base type if derived by extension.</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_RESTRICTION</td>
<td>ComplexType cannot be used as a base type if derived by restriction.</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_ALL</td>
<td>ComplexType cannot be used as a base type.</td>
</tr>
</tbody>
</table>
Implementation Syntax

var conFinal = oISchemaType.final;

Parameters

None.

Return Values

conFinal
An Enum of type SCHEMADERIVATIONMETHOD. The final value of the type definition.

[Visual Basic]

Implementation Syntax

conFinal = oISchemaType.final

Parameters

None.

Return Values

conFinal
An Enum of type SCHEMADERIVATIONMETHOD. The final value of the type definition.

[C/C++]

Implementation Syntax

HRESULT get_final(SCHEMADERIVATIONMETHOD* final);
Parameters

`final [out,retval]`
An Enum of type SCHEMADERIVATIONMETHOD. The base type of the complex type.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the final parameter is NULL.

The following table shows the enumerated list of numeric values that can be used for each of the types listed above.

<table>
<thead>
<tr>
<th>Enumerated value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMADERIVATIONMETHOD_EMPTY</td>
<td>0x0000</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_SUBSTITUTION</td>
<td>0x0001</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_EXTENSION</td>
<td>0x0002</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_RESTRICTION</td>
<td>0x0004</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_LIST</td>
<td>0x0008</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_UNION</td>
<td>0x0010</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_ALL</td>
<td>0x00FF</td>
</tr>
<tr>
<td>SCHEMADERIVATIONMETHOD_NONE</td>
<td>0x0100</td>
</tr>
</tbody>
</table>

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

element Element

Applies to: ISchemaType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
fractionDigits Property

Returns a variant of type VT_BSTR. This variant indicates the maximum number of digits in the fractional part of a value. For more information about the fractionDigits facet, see Data Type Facets.

[Script]

Implementation Syntax

```
var varFractionDigits = oISchemaType.fractionDigits;
```

Parameters

None.

Return Values

```
varFractionDigits
```

A variant. The fractionDigits facet of the restriction.

[Visual Basic]

Implementation Syntax

```
varFractionDigits = oISchemaType.fractionDigits
```

Parameters

None.

Return Values

```
varFractionDigits
```

A variant. The fractionDigits facet of the restriction.
Implementation Syntax

HRESULT get_fractionDigits (VARIANT* fractionDigits);

Parameters

fractionDigits [out,retval]
   A variant. The fractionDigits facet value.

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the fractionDigits parameter is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

   Note  If the facet is not used, the property will return "-1".

See Also

Data Type Facets | Parts of Data Types

Applies to: ISchemaType Interface
length Property

Returns a variant of type VT_BSTR. This variant indicates the number of units of length. For more information about the length facet, see Data Type Facets.

[Script]

**Implementation Syntax**

```javascript
var varLength = oISchemaType.length;
```

**Parameters**

None.

**Return Values**

`varLength`

A variant. The value of the length facet of the restriction.

[Visual Basic]

**Implementation Syntax**

```vbnet
varLength = oISchemaType.length
```

**Parameters**

None.

**Return Values**

`varLength`

A variant. The value of the length facet of the restriction.
Implementation Syntax

HRESULT get_length (VARIANT* length);

Parameters

length [out,retval]
A variant. The value of the length facet.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the length parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Note If the facet is not used, the property will return "-1".

See Also

Data Type Facets | Parts of Data Types

Applies to: ISchemaType Interface
maxExclusive Property

Returns a string. This string indicates a value that is one more than the maximum number of times the type can be used in a document \((x = x+1)\). For more information about the \texttt{maxExclusive} facet, see \href{Data Type Facets}{Data Type Facets}.

[Script]

**Implementation Syntax**

```javascript
var strMaxExclusive = oISchemaType.maxExclusive;
```

**Parameters**

None.

**Return Values**

\textit{strMaxExclusive}

A string. The \texttt{maxExclusive} facet of the restriction.

[Visual Basic]

**Implementation Syntax**

```vba
strMaxExclusive = oISchemaType.maxExclusive
```

**Parameters**

None.

**Return Values**

\textit{strMaxExclusive}

A string. The \texttt{maxExclusive} facet of the restriction.
Implementation Syntax

HRESULT get_maxExclusive (BSTR* maxExclusive);

Parameters

maxExclusive [out,retval]
   A string. The maxExclusive facet string.

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if the maxExclusive parameter is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Note  If the facet is not used, the property will return a NULL string.

See Also

Data Type Facets | Parts of Data Types

Applies to: ISchemaType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
maxInclusive Property

Returns a string that indicates the maximum number of times the type can be used in a document. For more information about the maxInclusive facet, see Data Type Facets.

[Script]

Implementation Syntax

```javascript
var strMaxInclusive = oISchemaType.maxInclusive;
```

Parameters

None.

Return Values

`strMaxInclusive`

A string. The maxInclusive facet of the restriction.

[Visual Basic]

Implementation Syntax

```vbscript
strMaxInclusive = oISchemaType.maxInclusive
```

Parameters

None.

Return Values

`strMaxInclusive`

A string. The maxInclusive facet of the restriction.
Implementation Syntax

HRESULT get_maxInclusive (BSTR* maxInclusive);

Parameters

maxInclusive [out,retval]
A string. The maxInclusive facet string.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the maxInclusive parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Note  If the facet is not used, the property will return a NULL string.

See Also

Data Type Facets | Parts of Data Types

Applies to: ISchemaType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**maxLength Property**

Returns a variant of type VT_BSTR. This variant indicates the maximum number of units in the lexical space of this type. For more information about the maxLength facet, see [Data Type Facets](#).

[Script]

**Implementation Syntax**

```javascript
var varMaxLength = oISchemaType.maxLength;
```

**Parameters**

None.

**Return Values**

`varMaxLength`

A variant. The value of the maxLength facet of the restriction.

[Visual Basic]

**Implementation Syntax**

```vbnet
varMaxLanguage = oISchemaType.maxLength
```

**Parameters**

None.

**Return Values**

`varMaxLanguage`

A variant. The value of the maxLength facet of the restriction.
**Implementation Syntax**

```c
HRESULT get_maxLength (VARIANT* maxLength);
```

**Parameters**

`maxLength ` [out,retval]

A variant. The value of the `maxLength` facet.

**Return Values**

- **S_OK**
  - The value returned if successful.
- **E_POINTER**
  - The value returned if the `maxLength` parameter is NULL.
- **E_FAIL**
  - The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note**  If the facet is not used, the property will return "-1".

**See Also**

- [Data Type Facets](#) | [Parts of Data Types](#)

Applies to: [ISchemaType Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
minExclusive Property

Returns a string. This string indicates a value that is one less than the minimum number of times the type can be used in a document \((x = x-1)\). For more information about the minExclusive facet, see Data Type Facets, in XML Data Types Reference, under XML Schema Reference.

[Script]

Implementation Syntax

```javascript
var strMinExclusive = oISchemaType.minExclusive;
```

Parameters

None.

Return Values

`strMinExclusive`

A string. The minExclusive facet of the restriction.

[Visual Basic]

Implementation Syntax

```vbnet
strMinExclusive = oISchemaType.minExclusive
```

Parameters

None.

Return Values

`strMinExclusive`
A string. The minExclusive facet of the restriction.

[C/C++]

**Implementation Syntax**

HRESULT get_minExclusive (BSTR* minExclusive);

**Parameters**

`minExclusive [out,retval]`

A string. The minExclusive facet string.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the minExclusive parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note**  If the facet is not used, the property will return a NULL string.

**See Also**

[Data Type Facets](#) | [Parts of Data Types](#)

Applies to: [ISchemaType Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
minInclusive Property

Returns a string that indicates the minimum number of times the type can be used in a document. For more information about the minInclusive facet, see Data Type Facets.

[Script]

Implementation Syntax

```javascript
var strMinInclusive = oISchemaType.minInclusive;
```

Parameters

None.

Return Values

`strMinInclusive`
A string. The minInclusive facet of the restriction.

[Visual Basic]

Implementation Syntax

```vbscript
Set strMinInclusive = oISchemaType.minInclusive
```

Parameters

None.

Return Values

`strMinInclusive`
A string. The minInclusive facet of the restriction.
**Implementation Syntax**

HRESULT get_minInclusive (BSTR* minInclusive);

**Parameters**

*minInclusive* [out,retval]

A string. The minInclusive facet string.

**Return Values**

S_OK

The value returned if successful.

E_POINTER

The value returned if the minInclusive parameter is NULL.

E_FAIL

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note**  If the facet is not used, the property will return a NULL string.

**See Also**

[Data Type Facets](#) | [Parts of Data Types](#)

Applies to: [ISchemaType Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**minLength Property**

Returns a variant of type VT_BSTR. This variant indicates the minimum number of units in the lexical space of this type. For more information about the minLength facet, see [Data Type Facets](#).

[Script]

**Implementation Syntax**

```javascript
var varMinLength = oISchemaType.minLength;
```

**Parameters**

None.

**Return Values**

`varMinLength`

A variant. The value of the minLength facet of the restriction.

[Visual Basic]

**Implementation Syntax**

```vbscript
varMinLength = oISchemaType.minLength
```

**Parameters**

None.

**Return Values**

`varMinLength`

A variant. The value of the minLength facet of the restriction.
Implementation Syntax

HRESULT get_minLength (VARIANT* minLength);

Parameters

minLength [out,retval]
A variant. The value of the minLength facet.

Return Values

S_OK
The value returned if successful.
E_POINTER
The value returned if the minLength parameter is NULL.
E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Note  If the facet is not used, the property will return "-1".

See Also

Data Type Facets | Parts of Data Types

Applies to: ISchemaType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
patterns Property

A SchemaStringCollection object is returned. The collection contains the pattern facets defined for the data type being queried.

Each string in the collection is a pattern construction. This construction uses a special regular expression language for its creation. The content value must match this pattern to be considered valid. For more information about the pattern facet, see Data Type Facets.

[Script]

Implementation Syntax

```javascript
var objPatterns = oISchemaType.patterns;
```

Parameters

None.

Return Values

`objPatterns`

An object. A string collection of the pattern facets for the data type.

[Visual Basic]

Implementation Syntax

```vbnet
Set objPatterns = oISchemaType.patterns
```

Parameters

None.
Return Values

$objPatterns$
   An object. A string collection of the pattern facets for the data type.

[C/C++]

Implementation Syntax

HRESULT get_patterns(ISchemaStringCollection** patterns);

Parameters

$patterns$ [out,retval]
   An object. A string collection of the pattern facets for the data type.

Return Values

S_OK
   The value returned if successful.
E_POINTER
   The value returned if $patterns$ parameter is NULL.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Note  If the facet is not used, the property will return an empty string collection.

See Also

Data Type Facets | Parts of Data Types

Applies to: ISchemaType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**totalDigits Property**

Returns a variant of type VT_BSTR. This variant indicates the maximum number of in a value. This total includes the digits in the fraction portion of the value. For more information about the totalDigits facet, see Data Type Facets.

[Script]

**Implementation Syntax**

```javascript
var varTotalDigits = oISchemaType.totalDigits;
```

**Parameters**

None.

**Return Values**

`varTotalDigits`  
A variant. The totalDigits facet of the restriction.

[Visual Basic]

**Implementation Syntax**

```vbnet
varTotalDigits = oISchemaType.totalDigits
```

**Parameters**

None.

**Return Values**

`varTotalDigits`  
A variant. The totalDigits facet of the restriction.
[C/C++]

**Implementation Syntax**

HRESULT get_totalDigits (VARIANT* totalDigits);

**Parameters**

`totalDigits` [out,retval]

A variant. The value of the totalDigits facet.

**Return Values**

S_OK
The value returned if successful.

E_POINTER
The value returned if the totalDigits parameter is NULL.

E_FAIL
The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**Note**  If the facet is not used, the property will return "-1".

**See Also**

[Data Type Facets](#) | [Parts of Data Types](#)

Applies to: [ISchemaType Interface](#)
variety Property

Returns a SCHEMATYPEVARIETY constant. This constant indicates the value of the type definition.

[Script]

Implementation syntax

var conVariety = objISchemaType.variety;

Parameters

None.

Return Values

conVariety
For simple types, the variety value can be one of the following:

SCHEMATYPEVARIETY_ATOMIC
SCHEMATYPEVARIETY_LIST
SCHEMATYPEVARIETY_UNION

For complex types, the variety value is always as follows:

SCHEMATYPEVARIETY_NONE

[Visual Basic]

Implementation syntax

conVariety = objISchemaType.variety
Parameters

None.

Return Values

\textit{conVariety}

For simple types, the \textit{variety} value can be one of the following:

\begin{itemize}
  \item SCHEMATYPEVARIETY\_ATOMIC
  \item SCHEMATYPEVARIETY\_LIST
  \item SCHEMATYPEVARIETY\_UNION
\end{itemize}

For complex types, the \textit{variety} value is always as follows:

\begin{itemize}
  \item SCHEMATYPEVARIETY\_NONE
\end{itemize}

[C/C++]

Implementation syntax

\begin{verbatim}
HRESULT get_variety(SCHEMATYPEVARIETY \* variety);
\end{verbatim}

Parameters

\textit{variety} [out,retval]

For simple types, the \textit{variety} value can be one of the following:

\begin{itemize}
  \item SCHEMATYPEVARIETY\_ATOMIC
  \item SCHEMATYPEVARIETY\_LIST
  \item SCHEMATYPEVARIETY\_UNION
\end{itemize}

For complex types, the \textit{variety} value is always as follows:

\begin{itemize}
  \item SCHEMATYPEVARIETY\_NONE
\end{itemize}
Return Values

S_OK
   The value returned if successful.

E_POINTER
   The value returned if the variety parameter is NULL.

See Also

Parts of Data Types

Applies to: ISchemaType Interface
whitespace Property

Returns an integer that indicates a value from the SCHEMAWHITESPACE enumerated list. For more information about the whitespace facet, see Data Type Facets.

The following table shows the SCHEMAWHITESPACE enumerated list.

<table>
<thead>
<tr>
<th>Enumerated value</th>
<th>Integer value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAWHITESPACE_NONE</td>
<td>-1</td>
</tr>
<tr>
<td>SCHEMAWHITESPACE_PRESERVE</td>
<td>0</td>
</tr>
<tr>
<td>SCHEMAWHITESPACE_REPLACE</td>
<td>1</td>
</tr>
<tr>
<td>SCHEMAWHITESPACE_COLLAPSE</td>
<td>2</td>
</tr>
</tbody>
</table>

[Script]

Implementation Syntax

```
var intWhitespace = oISchemaType.whitespace;
```

Parameters

None.

Return Values

`intWhitespace`

An integer. The whitespace facet of the restriction.

[Visual Basic]

Implementation Syntax

```
intWhitespace = oISchemaType.whitespace
```
Parameters

None.

Return Values

\textit{intWhitespace}

An integer. The whitespace facet of the restriction.

[C/C++]

Implementation Syntax

\texttt{HRESULT get_whitespace(SCHEMAWHITESPACE* whitespace);}

Parameters

\textit{whitespace [out,retval]}

An integer. The whitespace facet string.

Return Values

\texttt{S_OK}

The value returned if successful.

\texttt{E_POINTER}

The value returned if the whitespace parameter is NULL.

\texttt{E_FAIL}

The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

\textbf{Note} If the facet is not used, the property will return "SCHEMAWHITESPACE\_NONE".

See Also

\texttt{Data Type Facets | Parts of Data Types}
Applies to: ISchemaType Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SOM Reference
**IsValid Method**

Returns a Boolean true or false. If true, the data sent to the `isValid` method is valid according to the type definition of the `ISchemaType` object being queried. If false, the data is not valid.

[Script]

**Implementation Syntax**

```javascript
var bolValid = oISchemaType.isValid(strData);
```

**Parameters**

`strData`

A string. The data to be validated.

**Return Values**

`bolValid`

A Boolean value. The result of validation of the passed-in data.

[Visual Basic]

**Implementation Syntax**

```visualbasic
bolValid = oISchemaType.isValid(strData)
```

**Parameters**

`strData`

A string. The data to be validated.

**Return Values**
bolValid
   A Boolean value. The result of validation of the passed-in data.

[C/C++]

Implementation Syntax

HRESULT isValid(BSTR data, VARIANT_BOOL* valid);

Parameters

data [in]
   A string. The data to be validated.
valid [out,retval]
   A Boolean value. The result of validation of the passed-in data.

Return Values

S_OK
   The value returned if successful.
S_INVALIDARG
   The value returned if the data string is NULL.
S_POINTER
   The value returned if the valid return value is NULL.
S_NOTIMPL
   The value returned if itemType property is
      SOMITEM_DATATYPE_QNAME or SOMITEM_COMPLEXTYPE.
E_FAIL
   The value returned if something else is wrong.

To view reference information for Visual Basic or C/C++ only, click the
Language Filter button ✐ in the upper-left corner of the page.

Remarks

Documents containing Qname or NOTATION types or types derived from them
cannot be validated currently with XSD. Therefore, validation will fail with
E_FAIL and the valid parameter will be returned as VARIANT_TRUE and
isValid will return S_FALSE.

See Also

ISchemaItem itemType Property

Applies to: ISchemaType Interface
XSLT Developer's Guide

This guide is intended to help you, the XSLT developer, understand the basic principles and techniques of programming with XSLT using the MSXML 5.0 SDK.

The following topics provide more information about XSLT.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter Kit</td>
<td>Provides a high-level overview for beginning XSLT developers. This includes a Hello, World! XSLT example to get you started writing an XSLT style sheet.</td>
</tr>
<tr>
<td>Concepts</td>
<td>Examines various features of XSLT in detail. Provides a series of task-based exercises as a hands-on introduction to writing XSLT solutions.</td>
</tr>
<tr>
<td>How Do I?</td>
<td></td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Starter Kit

This Starter Kit provides a quick introduction to the XSLT technology as implemented in Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office. At the end of the section, you will have the information you need to create simple XSLT solutions as you continue to learn and explore the technology. This section contains the following topics:

- **What is XSLT?**
  A brief overview of XSLT as a W3C standard and as a programming language.

- **Behind the Scenes**
  A glance at XSLT processes, the tree of nodes representation, and two ways to link XSLT style sheets with source XML documents.

- **Frequently Asked Questions about XSLT**
  Find quick answers and learn from the experience of other developers.

- **Creating XSLT Solutions**
  A brief discussion of the requirements for creating an XSLT solution.

- **Deploying XSLT Solutions**
  A brief discussion of some issues you might encounter when deploying XSLT solutions in Windows or Internet Explorer.

- **Supported XSLT Features**
  A brief discussion of XSLT supported features and compliance with standards.

- **XSLT Utilities and Samples You Can Download**
Provides links to other XSLT resources.
What is XSLT?

Extensible Stylesheet Language Transformations (XSLT) has evolved from the early Extensible Stylesheet Language (XSL) standard. XSL specifies a language definition for XML data presentation and data transformations. Data presentation means displaying data in some format and/or medium. Presentation is about style. Data transformation means parsing an input XML document into a tree of nodes, and then converting the source tree into a result tree. Transformation is about data exchange.

Because these functionalities serve two distinct purposes, XSLT was proposed and later accepted as a separate standard for XML data transformation only. XSL is now generally referred to as XSL Formatting Objects (XSL-FO), to distinguish it from XSLT. The future of XSL-FO as a standard is uncertain, because much of its functionality overlaps with that provided by cascading style sheets (CSS) and the HTML tag set. If cross-vendor compatibility is important, you might want to avoid XSL-FO until it becomes a standard fully accepted by the Worldwide Web Consortium.

**Note**  MSXML versions 4.0 and later fully implement and support XSL Transformations (XSLT) Version 1.0 (W3C Recommendation 16 November 1999). If your XML application requires a later version of XSLT, Microsoft strongly recommends moving to the newer System.Xml framework classes, because all future XML development efforts will be focused there.

This section contains the following topics:

- [XSLT is a Programming Language](#)
- [Hello, World! (XSLT)](#)
- [XSLT is Rule-based](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
XSLT is a Programming Language

From a programmer's point of view, XSLT is a programming language for processing XML data—that is, transforming XML documents. As such, it supports the following:

- A small set of flexible data types: Boolean, number, string, node-set, and external objects.
- Programming flow-control: `<xsl:if>`, `<xsl:for-each>`, `<xsl:choose>`, and so on.

XSLT adopts the XPath language syntax for expressions. There are also built-in and custom extension functions for building expressions for XML data queries and transformations. For a complete list of XSLT language features, see XSLT Reference and XPath Reference in this documentation.

An XSLT style sheet is an XML document. This means that XSLT instructions are expressed as XML elements, and an XSLT style sheet can transform other XSLT style sheets, or even itself.

The following topic provides a simple example of an XSLT style sheet, hello.xsl. This example transforms a simple XML document (hello.xml) into an HTML document to be displayed in a Web browser.

- **Hello World! (XSLT)**
Hello, World! (XSLT)

The following example shows a simple but complete XML document transformed by an XSLT style sheet. The source XML document, hello.xml, contains a "Hello, World!" greeting from "An XSLT Programmer".

**XML (hello.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<hello-world>
    <greeter>An XSLT Programmer</greeter>
    <greeting>Hello, World!</greeting>
</hello-world>
```

The source document contains the `<?xml-stylesheet ...?>` processing instruction to link it to the XSLT style sheet, hello.xsl. The XSLT file contains instructions for transforming the source document into an HTML document.

**XSLT (hello.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:template match="/hello-world">
        <HTML>
            <HEAD>
                <TITLE></TITLE>
            </HEAD>
            <BODY>
                <H1><xsl:value-of select="greeting"/></H1>
                <xsl:apply-templates select="greeter"/>
            </BODY>
        </HTML>
    </xsl:template>
    <xsl:template match="greeter">
        <DIV>from <I><xsl:value-of select="."/></I></DIV>
    </xsl:template>
</xsl:stylesheet>
```
The effect of the transformation is to display the "Hello, World!" message on a Web page, followed by the signature of the greeter. This XSLT style sheet illustrates the basic features required of every XSLT style sheet:

- An XSLT style sheet typically starts with the `<?xml version="1.0"?>` processing instruction, declaring it as an XML file.
- The root element is `<xsl:stylesheet>`. The version attribute specifies the version of the XSLT specification. The `xmlns:xsl` attribute allows you to define a namespace alias (xsl) for use with every XSLT element in the document (e.g., `<xsl:value-of>` instead of simply `<value-of>`).
- The content of this root element consists of a set of template rules declared within the `<xsl:template>` elements. These rules specify how various elements in the source XML document are to be transformed into result elements.

The example above contains two simple template-rules. Together, they transform the contents in the source XML document into an HTML page. More complex XSLT style sheets contain either more template rules or template rules that are more complex.

**Note** This introductory sample is for explanatory purposes. To test the sample in Internet Explorer, you need one more file: an HTML file containing a script. This is because current versions of Internet Explorer do not ship with MSXML version 4.0 or later. For more information, and to run the Hello, World! sample, see [Initiate XSLT in a Script](Initiate XSLT in a Script).
XSLT is Rule-based

The "Hello, World!" example in the previous topic points to an important feature of XSLT programming: It is rule-based and declarative. In XSLT, rules are called template rules. A template rule is an instruction to transform a specified source element in a particular way. To transform a set of data, you must provide a complete set of rules. However, each rule is independent of every other.

The order in which rules are declared (or executed) is not significant. This makes XSLT programming side-effect-free. Although you can introduce side-effects by calling extension functions from within XPath expressions, the native way of programming with XSLT is without any side-effects. In this way, programming with XSLT is significantly different from imperative or procedure-oriented programming.
Behind the Scenes

A C/C++ programmer uses a compiler, a linker, and a loader to build the application to run on a machine. A scripter relies on script engines (or hosts) to interpret instructions written in a script. Behind the scenes, these processors put programs in action. Similarly, XSLT processors put the XML document and XSLT style sheets together as data and code, and invoke template rules to produce the desired output. The topics in this section provide a glance at the XSLT processes involved in executing any XSLT solutions. For an in-depth review of the subject, see Understanding the XSLT Process.

This section contains the following topics.

- XSLT Processing at a Glance
- Tree Representation and XPath
- Linking XSLT to a Source XML Document
XSLT Processing at a Glance

XSLT processing involves the following basic steps:

- Reading the XML source document and the associated XSLT style sheets.
- Parsing XML files, and then their associated XSLT files, into trees of nodes. Each node corresponds to an XML document element or attribute.
- Applying XSLT transformation to the source trees.
- Producing result trees according to the specification of the XSLT style sheet.
- Serializing the result trees as output files. These files output XML, HTML, or other text formats.

MSXML versions 4.0 and later support an XML parser and a built-in XSLT processor. The parser translates the textual (or serialized) XML documents (including the XSLT style sheets, which are XML documents themselves) into tree representations. The XSLT processor works with these representations.
Tree Representation and XPath

The XSLT processor does not work on raw XML data directly out of an XML document file. Instead, the raw XML data is first parsed into an XML DOM object that represents the XML elements as a tree of nodes. This tree representation allows you to build expressions in the XPath language syntax to query XML document elements. A template rule is invoked when the specified element, as expressed in an XPath pattern, is matched.

In the Hello, World! example, the template rule (`<xsl:template match="/hello-world"/>`) uses an XPath expression (`/hello-world`). This expression specifies that the template rule is applied to the top-level `<hello-world>` element in the source XML document. If you use `match="hello-world"` and the source document has multiple occurrences of the `<hello-world>` element, the template rule is applied to all of them.

If a pattern specified in an XSLT style sheet does not match any element in the source document, the associated template rule is ignored. This is possible if you make mistakes when you build the XPath expression. Therefore, proficiency in the XPath language is crucial for writing an XSLT style sheet effectively. To learn more about XPath, see the XPath Developer's Guide and XPath Reference, in this documentation.
Linking XSLT to a Source XML Document

Instructing the XSLT processor to do a transformation is often referred to as linking an XSLT style sheet to an XML document. There are two ways to do this.

- Embed an XSLT style sheet inside the source XML document. In the Hello, World! example, the `<xsl:stylesheet type="text/xsl" href="hello.xsl"/>` processing instruction does this. When you run the source XML file from within Internet Explorer, the transformation is applied automatically. However, you have little control as to which XSLT processor to invoke, unless you use MSXML 4.0 or later on Windows XP computers. If you are not running Windows XP, the processor used is likely to be the one shipped with Internet Explorer—that is, MSXML 2.0.

- Explicitly call the `transformNode` or `transformNodeToObject` method on the Document Object Model (DOM) of the source XML document. Do this from a script, or from another programming language that allows you to create the DOM for the source XML document and the XSLT style sheet. This method allows you full control over which version of MSXML you want to use. Currently, this is the only way to use MSXML 4.0 or later, unless you use MSXML 4.0 or later with Windows XP.

For more information about using MSXML 5.0 with Windows XP, see MSXML 5.0 and Windows XP.

For an example of linking to a source document via script, see Initiate XSLT in a Script.
Frequently Asked Questions about XSLT

The following are frequently asked questions about XSLT.

Why do my transformations fail using Internet Explorer?

How do I run XSLT transformations programmatically?

Do I need to use a different XSLT namespace with Internet Explorer?

I have installed Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office. Why isn't my application using it?

Does MSXML 5.0 provide a 100% compliant XSLT processor?

Does Internet Explorer 5.5 include a fully compliant XSLT processor?

Why was the MSDN online documentation for XSL so different from the XSLT specification?

Where can I find documentation about the compliance of MSXML?

How can I call MSXML from the command prompt to do batch processing of XSLT?

What can I do if I wrote lots of XSL using the old Internet Explorer version of XSL?

I updated a style sheet I wrote previously to transform ADO-persisted XML recordsets. I now get an "undeclared reference to namespace prefix" error. What do I need to do to fix this?

Will MSXML support XSLT 2.0?

Why do my transformations fail using Internet Explorer?
You might have an earlier version of MSXML running with Internet Explorer on your machine. MSXML versions 2.6 and earlier only support the XSL standard. MSXML versions 3.0 and later support XSLT 1.0. To use XSLT transformations, update the system by installing MSXML 3.0 or later.

You can install MSXML 3.0 in replace mode so that it becomes the default XML/XSLT processor for Internet Explorer 5.0 or 5.5.

You cannot, however, install MSXML 4.0 or later in such a way that it becomes the default XML/XSLT processor, because MSXML 4.0 and later can only be installed in side-by-side mode. Until Internet Explorer ships with MSXML 4.0 or later as its default XML parser, you can use MSXML 4.0 and later with Internet Explorer in the following ways only:

- **In Windows XP** - Although you cannot install MSXML 4.0 or later as the default parser, in Windows XP you can set MSXML as the default XML/XSLT processor by using a manifest. For more information, see [MSXML 5.0 and Windows XP](#).
- **For earlier versions of Windows** - The MSXML 4.0 and later features are available in Internet Explorer only via scripting. Embedded linking will not work for MSXML 4.0 or later with any current versions of Internet Explorer. For an example, see [Initiate XSLT in a Script](#).

How do I run XSLT transformations programmatically?

XSLT transformations can be initiated programmatically using any language that supports COM interfaces. Such programming languages include Microsoft JScript, VBScript, Visual Basic, C++, and even Perl. To run XSLT transformations programmatically, you must create two XML DOM objects, one for the XML source document and the other for the XSLT style sheet. Then, you call the transformNode() function on the XML source document with the XSLT style sheet XML DOM object as the argument. The following example in JScript illustrates these points.

**JScript File (test.js)**

```javascript
var xmlstr = "<object><name>apple</name><color>Red</color></object>"
var xsltstr = '<?xml version="1.0"?>' +
```
Try It!

1. Copy and paste the code into a file, and save it as test.js.
2. Type the "test.js" command from a command window.

Output

The output is "Red apple".

Do I need to use a different XSLT namespace with Internet Explorer?

No. Use the standard xmlns:xsl="http://www.w3.org/1999/XSL/Transform" syntax. If Internet Explorer returns an error when you use this namespace, it is
likely that you are using an earlier version of MSXML that does not support XSLT. If you have older XSL files and do not want to convert them to XSLT files, you can still use the namespace declaration xmlns:xsl="http://www.w3.org/TR/WD-xsl". To convert XSL files to XSLT files, use the XSL to XSLT Converter1.1 available from MSDN Online Downloads.

**I have installed Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office. Why isn't my application using it?**

If you embed your XSLT style sheet in your source XML document, MSXML 5.0 will not be used. For more information, see [Why do my transformations fail in Internet Explorer?](#), above.

If you call your transformation from a programming language, it is likely that you are not using the correct ProgID. You must use the version-dependent ProgID. The following Visual Basic code shows the correct syntax to create a DOMDocument object into which you load an XSLT file. The code uses the version-dependent ProgID.

```vbnet
Dim xsltDoc
Set xsltDoc = CreateObject("Msxml2.DOMDocument.5.0")
```

For more information about ProgIDs and syntax, see [GUID and ProgID Information](#). For more information about creating the DOMDocument object, see [Using DOMDocument](#).

**Does MSXML 5.0 provide a 100% compliant XSLT processor?**

Yes. MSXML 5.0 is fully compliant with the XSLT specification. For more information, see [Supported XSLT Features](#).

**Does Internet Explorer 5.5 include a fully compliant XSLT processor?**
Internet Explorer 5.5 is shipped with a version of MSXML that supports only XSL—not XSLT. However, you can use MSXML 5.0 with the help of client-side or server-side scripting. For more information, see Workarounds to Version Independence.

Why was the MSDN online documentation for XSL so different from the XSLT recommendation?

Because XSLT is a different technology from XSL.

Where can I find documentation about MSXML compliance?

Each new release of MSXML has a Bug List Page that describes known problems, such as coding mistakes or features that are not fully implemented. To find this page, search MSDN. For full conformance disclosure, see Supported XSLT Features.

If you find a bug or implementation point that is not clearly documented, please send feedback to the XML documentation team by using the XML Documentation Feedback form. To use this form, click the Feedback icon (the envelope) at the top-right corner of any page of this documentation.

How can I call MSXML from the command prompt to do batch processing of XSLT?

Because MSXML is a COM object, you can write VBScript, JScript, or other Windows Script Host (WSH) files to launch MSXML from the command prompt. Microsoft provides an XSLT command line utility, msxsl.exe, that performs command-line XSL transformations using the Microsoft XSL processor. Msxsl.exe is a small (~11K) command line utility that invokes Msxml5.dll to perform the actual work of the transformation. Msxsl.exe is available for download at the XML Developer Center at MSDN.
What can I do if I wrote lots of XSL using the old Internet Explorer version of XSL?

If you would like to upgrade XSL files so that they are compliant with the XSLT recommendation, you can use the [XSL to XSLT 1.1 Converter](https://msdn.microsoft.com/en-us/library/windows/desktop/dn621208%28v=vs.85%29.aspx) available from MSDN Downloads.

I updated a style sheet I wrote previously to transform ADO-persisted XML recordsets. I now get an "undeclared reference to namespace prefix" error. What do I need to do to fix this?

The reason for this error involves namespaces that ActiveX Data Objects (ADO) 2.6 uses when you use the `adPersistXML` formatting option with the `Save` method to persist an ADO recordset as XML.

For example, the following line shows the syntax used to persist the current ADO recordset as XML in a Visual Basic application.

```vbs
rs.Save "c:\temp\nwind.xml ", adPersistXML
```

The persisted XML output includes several namespace prefixes: "s", "dt", "rs", and "z". When you use XSLT style sheets to transform ADO-persisted XML, these namespaces must be declared for the transformation to succeed.

When using ADO to persist data as XML, you might encounter the following error text when you attempt to apply XSLT to your persisted XML.

'undeclared reference to namespace prefix'

For example, a style sheet might reference the XSLT namespace URI as follows:

```xml
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:template match="/">
     <xsl:element name="catalog">
       <xsl:value-of select="/z:row/@isbnnumber"/>
     </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```
This example, though valid as XSLT, produces the undeclared namespace error described above. To fix the problem, style sheets that specify the XSLT namespace URI must also declare any of the ADO namespaces that are used as input.

The following example shows how to update the previous style sheet code to declare these namespaces.

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
    xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
    xmlns:rs='urn:schemas-microsoft-com:rowset'
    xmlns:z='#RowsetSchema' version='1.0'>
    <xsl:template match="/">
        <xsl:element name="catalog">
            <xsl:value-of select="//z:row/@isbnnumber"/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

By adding the ADO namespace declarations above to your style sheet, you can correct the error and permit it to successfully transform ADO-persisted XML.

Style sheets that reference the older XSL namespace (like the following sample) do not produce this error.

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <xsl:element name="catalog">
            <xsl:value-of select="//z:row/@isbnnumber"/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Therefore, these style sheets are not affected and do not require any changes for this issue.

Return to top

Will MSXML support XSLT 2.0?
No. MSXML versions 4.0 and later fully implement and support XSL Transformations (XSLT) Version 1.0 (W3C Recommendation 16 November 1999). If your XML application requires a later version of XSLT, Microsoft strongly recommends moving to the newer System.Xml framework classes, because all future XML development efforts will be focused there.

Return to top
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Creating XSLT Solutions

It is relatively simple to create an XSLT solution. The minimum requirements for implementing an XSLT solution using the MSXML 5.0 SDK are as follows:

- Create one or more XSLT style sheet in a text editor.
- Install Microsoft XML (MSXML) 5.0 on all of your systems that will run the solutions. This includes both clients and servers.

For more information about the system requirements for using MSXML 5.0, see Installing and Registering the MSXML 5.0 SDK.

**Note** MSXML 5.0 can only be installed in side-by-side mode. To access it from an application, you must use the version-dependent CLSID or ProgID to instantiate the parser explicitly. This means that you cannot access MSXML 5.0 through an embedded XSLT style sheet from Microsoft Internet Explorer, unless you have a version of Internet Explorer that ships with MSXML 5.0.
Deploying XSLT Solutions

With MSXML 5.0, you can create XSLT solutions with any programming language that supports COM, such as Microsoft® JScript®, Microsoft Visual Basic® Scripting Edition (VBScript), Microsoft Visual Basic, and C++. Therefore, XSLT solutions can be desktop-based or Web-based. In fact, both the Windows® core system and Internet Explorer are released with various versions of MSXML, usually earlier versions than MXSML 5.0. This section discusses potential issues with deploying XSLT solutions in Internet Explorer and Windows environments.

The following topics discuss XSLT deployment issues.

- [Deploying XSLT in Internet Explorer](#)
- [Deploying XSLT on the Client](#)
- [Deploying XSLT on the Server](#)
Deploying XSLT in Internet Explorer

Internet Explorer 5.0 was released with earlier versions of Microsoft XML (MSXML) that do not support XSLT. To deploy your XSLT solution as a web application using Internet Explorer, you have the following options:

- If you want to use the MSXML 5.0 features, use the version dependent ProgIDs for MSXML 5.0 in a script. Valid ProgIDs are discussed in GUID and ProgID Information.
- If MSXML 5.0 features are not required, use XSL, the language that preceded XSLT. For more information about XSL, see the XSL documentation included with the MSXML 2.5 SDK, available from MSDN®.
- If you wish to avoid client-side upgrade of MSXML 5.0, use XSLT on the server with Active Server Pages (ASP) to transform XML documents and send HTML output to the client. With this option, you do not need to be concerned about the version of MSXML running on the client.

See Also

Deploying XSLT on the Client | Deploying XSLT on the Server
Deploying XSLT on the Client

Different versions of the Microsoft Windows operating system are released with different versions of MSXML. For this reason, deploying your XSLT solutions based on MSXML 5.0 can be tricky if you are not sure which MSXML processors will be invoked on a client. To ensure that all clients use the same version, use version-dependent ProgIDs from a script or another COM-enabled programming language to link your XSLT style sheets with source XML documents.

One problem associated with deploying XSLT solutions is that MSXML 3.0 and later are not currently available, widely or totally, in Internet Explorer 5.5. (Internet Explorer 5.5 is currently the only XML-enabled Web browser, although that situation is changing.) This limits any client-side XSLT solution to the realm of an intranet, rather than the Internet, until Internet Explorer ships with MSXML 3.0 and later.
Advantages of Transforming on the Client

There are several reasons for having client-side deployment of an XSLT solution. The primary reason is to reuse the client-side caching to avoid downloading the same documents repeatedly. For example, you might have several XML documents that use the embedding method to link to the same XSLT style sheet. With client-side caching, the style sheet is downloaded once and only once. The next downloaded XML document will use the cached version of the style sheet. This helps to reduce download time.

While server-dominated systems have a limited ability to save state, session states can be maintained on the client with the help of client-side scripting. This makes your applications easier for Windows users to use. You can also take advantage of capabilities that come with more sophisticated browsers, such as Internet Explorer 5.5: behaviors, drag and drop interactions, DHTML, ActiveX controls, and more.

Lastly, by using the ServerXMLHTTP component in client-side scripting, you can send XML information directly to the client without forcing a page update. The XMLHTTP component and XSLT are a powerful combination. They allow you to pull in just the information that you need from the server, transform it to a suitable form of output, reduce the bandwidth of information being transmitted, and reduce the loss of context that converting XML into HTML entails.

See Also

Deploying XSLT in Internet Explorer | Deploying XSLT on the Server
Deploying XSLT on the Server

Server-side transformation of XML with Active Server Pages (ASP) is a viable solution for sending HTML to downstream browsers. To avoid the incompatibility of different versions of MSXML, use version-dependent ProgIDs from within an ASP page to link your XSLT style sheets with source XML documents.
Advantages of Transforming on the Server

Transforming the content on the server has the following advantages.

- **Single Point of Entry.** Because the server can determine the client capabilities, the same page can service any kind of device, and can also consolidate a number of sections on the site into a single manageable interface.
- **Smaller Downloads.** You send to the server only the minimum amount of code necessary to display the page. Script, in particular, can be significantly reduced.
- **Error Control.** It is easier to handle errors transparently on the server than it is on the client.
- **Data Access and Security.** Data access on the client is an expensive operation, because it involves not just moving information but also controlling access. By moving much of this to the server, you can effectively limit the amount and type of information that needs to be sent.
- **Reduced Coding.** You can reuse existing blocks of both scripting and markup code. This makes it possible to create web pages more quickly.
- **Better Content Management.** By keeping your transformations on the client, you can more effectively integrate documents transparently. This means that users see only a complete document, not the various sub-documents that generate the document. This also makes it easier to make major changes to pages without disrupting the underlying content.
- **Stability of Platform.** With server-side content generation, you have complete control over the platform used to create the output. This means that you do not have to worry as much about versioning issues, problematic memory situations, and instability of the client. You can also plan your code resources more intelligently, and use features such as Microsoft Transaction Server (MTS) to better distribute your internal resources.
- **Business Logic.** If most of your data is XML-based, XSLT makes a very effective tool for writing business logic to both validate and manipulate the XML. Because business logic typically involves proprietary (and potentially insecure) information, performing these transformations on the server can give you improved performance and security.
Disadvantages of Transforming on the Server

The major disadvantage to using XSLT on the server is that state management becomes more problematic. Because you can save XML code in server-side session variables, this can have an adverse impact on performance (especially when dealing with Web farms). Therefore, XSLT solutions should be largely stateless. (Some possible exceptions are cookies, query and form string parameters, and similar lightweight state tokens.)

Another solution for handling state is to look at such solutions as the Simple Object Access Protocol (SOAP). SOAP has rapidly become the protocol of choice for exchanging messages between components outside of Windows proper, and can also be used internally through such services as Microsoft® C#™ and Visual Studio® 7. Because SOAP is an XML protocol, it can also be manipulated with XSLT, making XSLT-oriented messaging systems feasible. For more information about SOAP, see the MSDN XML Developer Center.

See Also

Deploying XSLT in Internet Explorer | Deploying XSLT on the Client
Supported XSLT Features

The implementation of MSXML 5.0 is in full compliance with the XSL Transformations (XSLT) Version 1.0 W3C Recommendation, 16 November 1999, and with the XML Path Language (XPath) Version 1.0 W3C Recommendation, 16 November 1999.

In addition, MSXML 5.0 maintains backward compatibility with the MSXML 2.0 and 2.5 implementations of Section 2 of the December 1998 XSL Working Draft of the Extensible Stylesheet Language (XSL). It does no support XSL formatting objects.

**Note**  For information about support for XSL, see the MSXML 2.5 documentation available at MSDN®.

Where the XSLT recommendation provides a fallback behavior in cases of error (for example, two matching templates with the same priority), MSXML performs the fallback behavior.

MSXML provides two mechanisms for implementing extension functions: embedded script implementations (<msxsl:script>) and external objects (addObject). MSXML does not provide a mechanism for implementing extension elements.

**See Also**

[addObject Method](#) | [msxsl:script Element](#)

**Other Resources**

[XML SDK 2.5 Documentation](#) | [XSL Transformations (XSLT) Version 1.0](#) | [XML Path Language (XPath) Version 1.0](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
XSLT Utilities and Samples You Can Download

Microsoft provides the following utilities, samples, and documentation, available for download from MSDN®.

Other Resources

XSLT Samples Viewer
This is an interactive, easy way to see a working example of each Extensible Stylesheet Language Transformations (XSLT) element and function. For each element or function, the viewer provides a common XML document and an XSLT file. To see the results of applying the XSLT file to the document, just click the Result button in the viewer.

Internet Explorer Tools for Validating XML and Viewing XSLT Output
Installing this tool adds the View XSL Output and Validate XML commands to the right-click menu of Microsoft® Internet Explorer. The View XSL Output command enables you to view the output tagging produced by an Extensible Stylesheet Language (XSL) or XSLT file. The Validate XML command enables you to validate XML against an embedded schema when loading XML using the Internet Explorer MIME viewer.

MSXSL.EXE Command Line Transformation Utility
Unlike hard coding an XSLT file name in an XML file, this utility enables you to specify any combination of XML and XSLT files at the command line and invoke Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office to perform the transformation.

XSLT Code Samples on MSDN
These samples show some of the ways that you can use XSLT to dynamically transform XML content.

XSL to XSLT Converter 1.1
This tool converts the XSL files included with Microsoft Internet Explorer 5.0 to XSLT-compliant files.

MSXML 2.5 Documentation
Because MSXML 5.0 fully supports XSLT, the MSXML 5.0 SDK is
specific to XSLT, not XSL. If you want to develop XSL files instead of XSLT files, download the MSXML 2.5 SDK for XSL-specific information.

**XSL Debugger**

This simple HTML-based debugger steps through the execution of an XSL file. The debugger identifies the current positions in both the XSL file and XML documents. You can also set and clear breakpoints.
Concepts

This section discusses various XSLT concepts in detail. It contains the following topics:

- Understanding the XSLT Process
- Template-Driven Transformations
- Data-Driven Transformations
- Displaying Transformation Results
- Using XSLT with the DOM or SAX
- Using XPath Expressions to Select Nodes
- Using XSLT Variables and Parameters
- Sorting with XSLT
- Working with Namespaces and Prefix Aliases
- Using msxsl:node-set() to Process Result-Tree Fragments
- Using Named Template Rules in XSLT
- Inserting Automatic Numbering Using XSLT
- Importing and Including Style Sheets
- Using XSLT Keys to Increase Performance
- Creating Interactive XSLT
- Controlling White Space with XSLT
Understanding the XSLT Process

The following is a summary of the XSLT process.

1. An XML file and an XSLT style sheet are read and parsed into the source and style sheet trees, respectively.
2. The template rules declared in the XSLT style sheet are applied to each matching XML element in the source tree. The application of the template rules produces a result tree.
3. The result tree is serialized, often as a character stream.

The following illustration presents a visual introduction to the various steps of the XSLT process. It shows the various input files and output destinations for an XSLT transformation. The most common transformation produces HTML output from an XML data source. Click on a colored area of this illustration to find out more about that component of the XSLT transformation.
XSLT Document Map

The following is a typical XSLT document. Click on a colored piece of text for more information about that component of an XSLT file.

Complete listings of the files used here can be found in Sample Files for the XSLT Document Map.
See Also

Document Map (XML Developer's Guide)
Sample Files for the XSLT Document Map

This example includes the XSLT file used in the XSLT Document Map.

The example demonstrates using the key() function and <xsl:key> element to group data into categories.

The mode attribute is used to produce two different tables from the same XML data. For more information about modes, see xsl:apply-templates Element.

XML File (books.xml)

Use the Sample XML File (books.xml). Below the line <?xml version='1.0'?>, add the following line:

<?xml-stylesheet type="text/xsl" href="books.xsl"?>

Ancillary XML File (ancillary.xml)

The books.xsl file loads the following XML file to demonstrate the document() function:

<?xml version='1.0'?>
<document>
  <watermark>Scootney Publishing</watermark>
  <copyright>Copyright 2000, Scootney Publishing. All Rights Reserved</copyright>
</document>

XSLT File (books.xsl)

This file is the same one that is shown in the document map. It provides examples of common tagging within an XSLT file.

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:variable name="recentdate">2001-01-01</xsl:variable>
<xsl:variable name="ancillary" select="document('ancillary.xml')"/>
<xsl:key name="categ_key" match="book" use="genre"/>
<xsl:key name="author_key" match="book" use="author"/>

<xsl:template match="/">
  <html>
    <head>
      <title>Scootney Publishing: Special Exhibit</title>
      <style>
        h1 {font-family: Arial,Helvetica,Univers,sans-serif; font-size: 18pt; background-color: #00CCCC; width: 90%}
        h2 {font-family: Arial,Helvetica,Univers,sans-serif; font-size: 14pt; width: 90%}
        h3 {font-family: Arial,Helvetica,Univers,sans-serif; font-size: 12pt; background-color: gold; width: 90%; border-width: 1; border-style: solid}
        h4 {font-family: Arial,Helvetica,Univers,sans-serif; font-size: 11pt; background-color: gold; width: 33%; border-width: 1; border-style: solid}
      </style>
    </head>
    <body>
      <div style="position:absolute;font-size:96; font-family:Times New Roman;color:#F0F0F0;z-index:-1">
        <xsl:value-of select="$ancillary//watermark"/>
      </div>
      <h1>&#xA0;Scootney Publishing: Special Exhibit Catalog</h1>
      <!-- Group by catalog in first pass, then by author. -->
      <xsl:apply-templates select="catalog" mode="grp_categ"/>
      <xsl:apply-templates select="catalog" mode="grp_author"/>
      <p/>
      <div style="font-size:9">
        <xsl:value-of select="$ancillary//copyright"/>
      </div>
    </body>
  </html>
</xsl:template>

<xsl:template match="catalog" mode="grp_categ">
  <h2>Titles Grouped by Genre</h2>
  <xsl:for-each select="book[count(. | key('categ_key', genre)[1])]">
    <h3>&#xA0;<xsl:value-of select="genre"/></h3>
    <xsl:for-each select="key('categ_key', genre)">
      <xsl:sort select="title"/>
      <table border="0" width="75%">
        <tr>
          <th width="10%" align="right">Title</th>
          ...
        </tr>
      </table>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
<table>
<thead>
<tr>
<th>Author</th>
<th>Copyright</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Summary**

You can view the output in Microsoft® Internet Explorer.
See Also

Document Map (XML Developer's Guide) | Grouping Data by Examining the Keyed Node-set
XML Declaration in an XSLT File

An XSLT file is an XML file. Although it is not required, you should use an XML declaration, such as `<?xml version="1.0" encoding="UTF-8"?>`, at the beginning of the XSLT style sheet. This allows other applications to recognize the data types contained in the file.

See Also

XML Declaration (XML Developer's Guide)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
XSLT Style Sheet Declaration

The following is a typical XSLT style sheet declaration.

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

The style sheet declaration is the `<xsl:stylesheet>` element. The attributes of this element must declare the version of XSLT, and must include at least one namespace declaration. For more information, see the [xsl:stylesheet Element reference topic](#).
Namespace Prefix Declarations

An XSLT style sheet must have at least one namespace declaration as an attribute of the `<xsl:stylesheet>` element. For MSXML versions 4.0 and later, this is `<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`. This particular namespace declaration (i.e., xsl) is required for all the XSLT-defined elements, including the `<xsl:stylesheet>` element itself. Additional namespace prefixes can also be added. For example, to use the MSXML custom extension element, `<msxsl:script>`, you must add a namespace declaration for the msxsl prefix, as follows:

```xml
<xsl:stylesheet version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:msxsl="urn:schemas-microsoft-com:xslt">
```

The default namespace has no prefix and is usually used for literal result elements in a template rule. The following is an example of a literal result element:

```xml
<DIV>XSLT Output</DIV>
```

See Also

[Using Namespaces in Documents](XML Developer's Guide) | [Declaring Namespaces for XSLT] | [Working with Namespaces and Prefix Aliases] | [Declaring the msxsl: Namespace Prefix]
Namespace Prefixes in XSLT

The following are some commonly used namespace prefixes in an XSLT style sheet.

```xml
<?xml version='1.0'?><xsl:stylesheet version='1.0'
    xmlns='urn:schemas-mycompany-com:xslt'
    xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
    xmlns:msxsl='urn:schemas-microsoft-com:xslt'
    xmlns:fo='http://www.w3.org/1999/XSL/Format'>
```

In addition, user-defined namespace prefixes might be needed. For example, if you use `<msxsl:script>` to implement an extension function for an XPath expression, you must specify a namespace prefix to be used with the extension function. Suppose you implemented the following `doSomething` function.

```xml
<msxsl:script language="JScript" implements-prefix="user">
    function doSomething(nodeset) {
        // result = something about the nodeset
        return result;
    }
</msxsl:script>
```

You could use it in a template rule, as follows:

```xml
<xsl:value-of select="user:doSomething(.)"/>
```

You would have to add the user namespace declaration to the `<xsl:stylesheet>` element:

```xml
xmlns:user=http://mycompany.com/mynamespace
```

**Note** It is recommended that you use the default namespace for literal result elements, such as `<HTML>` and `<H2>`.

**See Also**

[Working with Namespaces and Prefix Aliases](#) | [Declaring the msxsl: Namespace Prefix](#)
Output Method Declarations

After an XSLT transformation produces a result tree, the `<xsl:output>` element is used by the processor to determine how to serialize the result tree. The value of its `method` attribute can be `xml`, `html`, or `text`. The `<xsl:output>` element is usually placed after the `<xsl:stylesheet>` element. If the output is not serialized, but is instead made available as a tree (for example, in the DOM API), this element is ignored. This element must be a child of the `<xsl:stylesheet>` element.

The following are output method declarations.

```xml
<xsl:output method="xml"/>
<xsl:output method="html"/>
<xsl:output method="text"/>
```

See Also

[Controlling Output Format with the media-type Attribute] | [xsl:output Element]
XSLT Variables and Parameters

In XSLT, variables identify generically named objects, from which a value can be retrieved as needed. A variable can be used as a cache for a value that is used multiple times. It also provides a more efficient mechanism for setting a value or for referring to a particular result tree fragment.

XSLT parameters are used to pass data. Local parameters pass data from one template rule to another. Global parameters pass a value into a style sheet from outside. In Microsoft Internet Explorer, parameters are passed to a style sheet via scripting.

For detailed information about variables and parameters, see Using XSLT Variables and Parameters.

See Also

Passing Parameters Using Named Templates | xsl:variable Element | xsl:param Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Template Rules

Each `<xsl:template>` element or template rule consists of two components: a match pattern, and the template itself.

```xml
<xsl:template match="region">
[template]
</xsl:template>
```
**Match Patterns**

The match pattern consists of a `match` attribute and an XML Path Language (XPath) expression, which specifies which portions of the source tree are processed by the template rule. For example, the following match pattern processes all `<region>` elements in the source tree.

```xml
<xsl:template match="region">
</xsl:template>
```

Similarly, the following match pattern selects all the `books_sold` attributes for processing.

```xml
<xsl:template match="@books_sold">
</xsl:template>
```

Because match patterns are XPath-based, they are not limited to selecting only an element from the source tree. They can also select text, processing instructions, comments, CDATA declarations, and so on. For more information, see [Using XPath Expressions to Select Nodes](#).
Templates

Each time the template locates something in the source tree that matches the match pattern, it places content in the result tree. That is, the template rule instantiates the template for each match.

Exactly what goes into a given template depends on how the template is to transform the source document. If transforming to HTML, for example, the template might consist of HTML markup, plain text content, comments, scripts, and so on.

See Also

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Literal Result Elements in XSLT Files

An XSLT file can contain elements that are in the XSLT namespace, such as <xsl:if>. These elements are interpreted as XSLT code to be processed. Elements in the XSLT file that are not in the XSLT namespace, such as <HTML>, <BODY>, <P>, and <H2>, are treated as literal result elements, and are copied to the output with minimal processing. HTML elements in the XSLT file must be closed: Be sure to include a closing slash, such as <BR/> or </P>.

XPath Expressions

An XPath expression selects a node or nodes that match the specified pattern and criteria. The expression is run against a node tree representation of an XML document.

See Also

Using XPath Expressions to Select Nodes | XPath Syntax | XPath Examples | Location Path Examples
Comments in XSLT

Comments in XSLT work the same as comments in XML. You can comment out a block of XSLT elements. You cannot comment out text inside a start tag, such as `<xsl:value-of <!-- mycomment--> select="book">`. This causes an error.

See Also

Generating Comments, Processing Instructions, and Elements | xsl:comment Element | Comments (XML Developer's Guide)
<xsl:for-each> Loops

The <xsl:for-each> element establishes the context for iteration. The XSLT transformation instructions within the for-each loop are applied to each of the nodes selected by the select attribute of the <xsl:for-each> element. Each source element selected by <xsl:for-each> becomes a new context against which operations declared within the <xsl:for-each> are performed.

Example

```xml
<xsl:for-each select="/book">
  <tr>
    <td>
      <xsl:value-of select="author"/>
    </td>
    <td>
      <xsl:value-of select="title"/>
    </td>
  </tr>
</xsl:for-each>
```

See Also

<xsl:for-each Element>
XPath Functions and XSLT Functions

XSLT has a variety of predefined functions.

An expression in XSLT can include a function call.

A function returns a value.

See Also

Using Functions in XPath Expressions | Using the key() Function | XSLT Functions | XPath Functions
Conditional Statements

If the expression in an `<xsl:if>`, `<xsl:when>`, or `<xsl:otherwise>` element evaluates to true, the template body inside that conditional element is instantiated.

There are two kinds of conditional constructions: if, and choose/when/otherwise. For a simple statement that either does or does not perform an action, use if. To perform one action for one condition but a different action for a different condition, use choose/when/otherwise.

The `test` attribute contains the expression to evaluate, and can be a comparison or an XPath expression. The expression evaluates to true if the expression returns any of the following:

- A non-empty node-set
- A non-zero number
- A non-empty result tree fragment
- A non-empty string

Example of `<xsl:if>`

```xml
<xsl:for-each select="/book">
  <tr>
    <td>
      <xsl:value-of select="title"/>
    </td>
    <td>
      <xsl:attribute name="bgcolor">lightyellow</xsl:attribute>
      <xsl:if test="price &gt; 10">
        <xsl:attribute name="bgcolor">lightgreen</xsl:attribute>
      </xsl:if>
      <xsl:value-of select="price"/>
    </td>
  </tr>
</xsl:for-each>
```

Example of `<xsl:choose>`
<xsl:for-each select="//book">
  <div>
    <xsl:choose>
      <xsl:when test="self::*[genre = 'Romance']">
        <xsl:attribute name="style">background-color: pink</xsl:attribute>
      </xsl:when>
      <xsl:when test="self::*[genre = 'Fantasy']">
        <xsl:attribute name="style">background-color: lightblue</xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="style">background-color: lightgreen</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:value-of select="title"/>
  </div>
</xsl:for-each>

See Also

Defining Conditional Templates Using <xsl:if> and <xsl:choose> | xsl:choose Element | xsl:if Element
The `document()` XSLT function allows you to incorporate another XML file into the running context. This function enables you to select and extract XML data from multiple XML files.

In the context of XSLT, the words "import" and "include" refer to incorporating other XSLT files, rather than XML files.

- The `<xsl:import>` element imports another style sheet into the current style sheet. The imported definitions and template rules have lower priority.
- The `<xsl:include>` element includes another style sheet into the current style sheet. The included definitions and template rules have the same priority as the style sheet that includes them. The `<xsl:include>` element is only allowed as the child of the `<xsl:stylesheet>` element.

**See Also**

[document Function](#) | [Importing and Including Style Sheets](#)
The XSLT Processor

The XSLT processor works with a source tree, applies a style sheet tree to the source tree, and produces a result tree. The source tree and the result tree are the XML DOM for the source XML file and the XSLT style sheet, respectively. The trees are produced by the MSXML parser.

The following illustration shows the processing relationship among the XSLT processor, XSLT tree, source tree, and result tree. Click on a colored area in the illustration to find out about that particular part of this relationship.
Working with a Source Tree

To locate the data in a transformation, the XSLT processor uses the XPath language to navigate the source tree of the document.

XPath treats an XML document as a tree of nodes. In general, a node corresponds to an element or an attribute in the XML document, except for the root node. The root node represents the source document as whole and contains, but does not correspond to, the top-level element in the XML document. A collection of nodes is called a node-set.

The following table shows seven types of nodes. A document does not necessarily contain all seven node types.

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Represents an attribute of an XML element. For example, the <code>&lt;book title=&quot;The Encyclopedia of Cows&quot;/&gt;</code> element has a corresponding attribute node with title as the node name and &quot;The Encyclopedia of Cows&quot; as the node value. Notice that xmlns:ns=&quot;a_namespace_uri&quot; corresponds to a namespace node, but not to an attribute node.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Represents a sequence of consecutive characters that forms non-markup character data of an XML element. For example, for the <code>&lt;book&gt;The Encyclopedia of Cows&lt;/book&gt;</code> element in the XML document, there is a corresponding text node for the text of &quot;The Encyclopedia of Cows&quot;. However, for <code>&lt;book&gt;The Encyclopedia of &lt;animal&gt;Cows&lt;/animal&gt;&lt;/book&gt;</code>, there will be two text nodes: one for the text of &quot;The Encyclopedia of &quot; and the other for &quot;Cows&quot;.</td>
</tr>
<tr>
<td>Text</td>
<td>Corresponds to a distinct namespace prefix that is in scope for an element. A namespace is declared with an xmlns:attribute of an</td>
</tr>
</tbody>
</table>
Namespace element. For example, for `<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`, the namespace node refers to the "xmlns:xsl="http://www.w3.org/1999/XSLT/Transform" part of the element. Namespace nodes might or might not be present in a document.

Represents a processing instruction declared in the source XML document, such as `<?xml-stylesheet type="text/xsl" href="myXSLT.xsl"?>`. However, `<?xml version="1.0"?>` does not correspond to a processing node. Notice that a processing-instruction node does not represent processing instructions within the document type declaration of the XML document.

Comment

Represents a comment in the XML document. For example, `<!-- this is a comment -->` is represented by a comment node. Notice that a comment node does not represent comments within the document type declaration of the XML document.

See Also

[XPath Nodes Example](#) | [The XSLT Processor](#)
XPath Nodes Example

To help illustrate the nodes used by the XML Path Language (XPath), the following graphic represents the source tree for the first <book> element in the XML Document Map.

This source tree represents the nodes that are accessible using XSLT. If you compare the source file to the source tree represented here, you will see that not all of the objects within the document appear in this source tree. For example, the source tree does not have nodes for the <?xml-stylesheet type="text/xsl" href="show_books.xsl"?> and <!DOCTYPE catalog SYSTEM "catalog.dtd"> elements that appear in the document prolog. These missing objects still exist in the DOM. You can access them programmatically, although you cannot access them directly using XPath and, by extension, XSLT.
Although information in the prolog of a document is not typically accessible, XPath and XSLT are able to access it because the document has been fully parsed by the time the XSLT processor accesses the document tree.

For information about the name and string values associated with the nodes in the previous graphic, see XPath Node Names and String Values. For information about character references (such as $\#x47;$ and &amp;), see Character and Entity References.

**See Also**

Working with a Source Tree | Using XPath to Select Nodes | The XSLT
Processor
# Names and Values of XPath Nodes

In addition to specifying a node's relationship to other nodes in the tree, XPath defines a name and a string value for each node type, as follows:

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Node Name</th>
<th>Node Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>Empty string (&quot;&quot;)</td>
<td>Concatenated value of text in all descendant element nodes and text nodes.</td>
</tr>
<tr>
<td>Element</td>
<td>Element name</td>
<td>Concatenated value of text in all descendant element nodes and text nodes.</td>
</tr>
<tr>
<td></td>
<td>(expanded form, including namespace URI, if any)</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Attribute name</td>
<td>The value of the attribute.</td>
</tr>
<tr>
<td></td>
<td>(expanded form, including namespace URI, if any)</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>Empty string (&quot;&quot;)</td>
<td>All text in that node (multiple end-of-line sequences normalized to single newline).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Everything in the processing instruction, except for the target and the white space that follows the target.</td>
</tr>
<tr>
<td>Processing-</td>
<td>The target of the processing instruction</td>
<td></td>
</tr>
<tr>
<td>instruction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td>Empty string (&quot;&quot;&quot;)</td>
<td>All text between the opening &lt;!-- and closing -- &gt; delimiters.</td>
</tr>
<tr>
<td></td>
<td>Namespace prefix (excluding &quot;:&quot;), or empty string (&quot;&quot;), if no prefix is declared for the namespace</td>
<td>The URI which is bound to the namespace.</td>
</tr>
</tbody>
</table>

## Example

The following are some of the node names and values associated with the node
tree that is presented as a diagram in XPath Nodes Example.

<table>
<thead>
<tr>
<th>Node</th>
<th>Name</th>
<th>String Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>Empty string (&quot;&quot;)</td>
<td>&quot;<a href="http://www.example.microsoft.com/catalogbk101;Ambercrombie">http://www.example.microsoft.com/catalogbk101;Ambercrombie</a>, KimXMLDeveloper's &amp;uideComputer44.952000-10-01and&amp;amp.&quot; plus text from the remaining &lt;book&gt; element</td>
</tr>
<tr>
<td>Comment</td>
<td>Empty string (&quot;&quot;')</td>
<td>&quot;catalog last updated 2000-10-01&quot;</td>
</tr>
<tr>
<td>&lt;catalog&gt; element</td>
<td>&quot;catalog&quot;</td>
<td>&quot;<a href="http://www.example.microsoft.com/catalogbk101;Ambercrombie">http://www.example.microsoft.com/catalogbk101;Ambercrombie</a>, KimXMLDeveloper's &amp;uideComputer44.952000-10-01and&amp;amp.&quot; plus text from the remaining &lt;book&gt; element</td>
</tr>
<tr>
<td>&lt;book&gt; element</td>
<td>&quot;book&quot;</td>
<td>&quot;bk101;Ambercrombie, KimXMLDeveloper's &amp;uideComputer44.952000-10-01and&amp;amp.&quot;</td>
</tr>
<tr>
<td>xmlns attribute</td>
<td>&quot;id&quot;</td>
<td>&quot;<a href="http://www.example.microsoft.com/catalog">http://www.example.microsoft.com/catalog</a>&quot;</td>
</tr>
</tbody>
</table>

For information about references (such as &x47; and &amp;), see Character and Entity References.

See Also

Processing Text Strings by Using String Functions | Working with a Source Tree | Using XPath Expressions to Select Nodes | The XSLT Processor
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Processing Template Rules

When the XSLT processor begins to process the XSLT tree, the processor looks for the template rule that points to the document root element in the source tree.

At its simplest, the template rule that points to this root element is as follows:

```xml
<xsl:template match="/firstelement"/>
```

Here the value of the match pattern indicates that this rule applies to the first element (represented by `firstelement`) of the root node (represented by the slash, `/`). In the Sample XML File (books.xml), the simplest template rule that points to the document root element is the following:

```xml
<xsl:template match="/catalog"/>
```

After the XSLT processor locates this initial template rule, the processor evaluates the rule and proceeds as follows:

- If the rule has no conditional statements, it most likely directs the processor to simply transform this document root element. In this case, the processor transforms the root element and then looks for a template rule that matches the next highest element in the source tree. The processor transforms that element, and then looks for the next highest element. This process continues for all the elements in the source tree.
- If the rule contains conditional statements, the processor evaluates each statement and executes the appropriate action. Often, each conditional statement directs the XSLT processor through a series of template rules that are specific to the conditional statement. Essentially, this means that the XSLT processor ignores all template rules associated with other conditional statements.
Controlling Processing

As you might have noticed, the processing of an XSLT file is not procedural like it is in most programming languages. That is, the XSLT processor does not sequentially process each template rule as it appears in the XSLT tree.

Under certain circumstances, however, the sequence in which the style sheet is processed can be determined, and to a limited extent controlled:

- **Using Conditional Processing** — XSLT provides the `<xsl:if>` and `<xsl:choose>` elements for testing conditions. By using these elements, an XSLT file can direct the processor through a specific series of templates.
- **Importing Other Style Sheets** — An XSLT file can import other XSLT files. The order in which the corresponding `<xsl:import>` elements appear in the original file determines the import precedence, which determines the order in which the processor applies the other XSLT files.
Handling Template Rule Conflicts

While processing an XSLT tree, the XSLT processor might encounter a conflict that must be resolved:

- If the processor finds multiple template rules that match a given node, the processor needs to decide which of these template rules to use. For more information about how the processor makes this decision, see Processing Multiple Matching Templates Rules.
- If there are no matches, the processor uses one of the built-in template rules to decide how to handle the node. For more information about these built-in template rules, see Understanding the Built-in Template Rules.

See Also

The XSLT Processor
Processing Multiple Matching Template Rules

If more than one template rule matches a given node in the source tree, MSXML gives the highest priority to the template rule that matches the most specific pattern. Specifically, the conflict is resolved as follows:

First, MSXML selects the pattern with the highest import precedence. An XSLT file can import other XSLT files, and the order in which the corresponding `<xsl:import>` elements appear in the original file determines the import precedence. If there is still more than one match after this step, proceed to the next step.

The second step is to assign a numeric priority to each conflicting pattern, as shown in the following table. Select the pattern with the highest priority. If there is still more than one match after this step, proceed to the next step.

<table>
<thead>
<tr>
<th>Assign a priority of...</th>
<th>If the match pattern...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (zero)</td>
<td>Is an element or attribute name.</td>
</tr>
<tr>
<td>-0.25</td>
<td>Includes the wildcard character (*).</td>
</tr>
<tr>
<td>-0.5</td>
<td>Consists of only a node test, such as text(), for selecting nodes.</td>
</tr>
<tr>
<td>0.5</td>
<td>Does not match any of the other criteria.</td>
</tr>
</tbody>
</table>

**Note** If the pattern contains a pipe symbol (|) that specifies alternate XPath expressions, treat each expression as if it were a separate match pattern.

Finally, if everything else is equal, the last matching pattern is used as default.
See Also

Processing Template Rules | Understanding the Built-in Template Rules
Understanding the Built-in Template Rules

MSXML supports various built-in template rules when the XSLT processor cannot find a template rule explicitly declared.
Rules for Root and Element Nodes

The built-in template rule for the root and element nodes is as follows:

```xml
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

This rule instructs the processor to recursively process all children of the root node and of all element nodes, according to applicable template rules explicitly declared in the style sheet. To enforce transformation in which only the explicitly declared template rules are used, override this built-in rule as follows:

```xml
<xsl:template match="*|/"/>
```
Rules for Text and Attribute Nodes

The built-in template rule for text or attribute nodes is as follows:

```xml
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

For every text or attribute node, this template specifies that the text content of each matched node is to be output straight to the result tree.

**Note** Because XSLT treats newline characters, along with other white space characters, as text nodes, this built-in rule applies to white space characters as well. For more information about XSLT and white space, see [Overriding `<xsl:preserve-space>` and `<xsl:strip-space>`](#).
Rules for Comment Nodes and Processing-Instruction Nodes

The following is the built-in template rule for comment and processing-instruction nodes.

```xml
<xsl:template match="processing-instruction()|comment()"/>
```

The effect of this built-in template rule is to suppress output in the result tree. If you need to reproduce comments in the result tree, supply the following template rule to override the built-in one:

```xml
<xsl:template match="processing-instruction()"/>
<xsl:template match="comment()">
    <xsl:value-of select="."/>
</xsl:template>
```

See Also

Processing Template Rules | Processing Multiple Matching Templates Rules
Linking to an XSLT File

There are two way to link an XSLT style sheet to an XML source document:

- Embed the style sheet in the source document using the `<?xml-stylesheet type="text/xsl" href="myxsl.xsl"?>` processing instruction.
- Use a script or a COM-enabled programming language to Load the XML and XSLT files into DOM. Call the `transformNode` method to apply the transformation.

From a performance standpoint, it does not matter whether you link using a style sheet processing instruction or using the DOM. The difference between these two methods focuses on the underlying data, as outlined in the following table.

<table>
<thead>
<tr>
<th>Link Using...</th>
<th>If...</th>
</tr>
</thead>
</table>
| A style sheet processing instruction | • The data is fairly static (changes little, if at all).  
• There is no need for dynamic transformations which include the passing of parameters. |
| The DOM | • The data is dynamic.  
• You want to dynamically evaluate the data by passing parameters or performing dynamic transformations.  
• You want a more modular approach that is easier to maintain. |

**Note** For ease in learning XSLT, many of the samples in this section use a style sheet processing instruction to link to the XSLT file. These samples would work just as well if you linked them using the DOM instead.

**See Also**

[Linking Using a Processing Instruction](#) | [Linking Using the DOM](#) | [Displaying Transformation Results](#) | [The XSLT Processor](#)
Linking Using a Processing Instruction

If you link to an XSLT file using a style sheet processing instruction, that instruction needs to be at the beginning of an XML document, following the XML declaration (if there is one) and the document type declaration (if there is one). For example, the style sheet processing instruction needs to appear just after `<?xml version ="1.0"?>` in the Sample XML File (books.xml), as shown in bold in the following document fragment:

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="books.xsl" ?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
  </book>
  ....
</catalog>
```

See Also

Processing Instructions | Linking to an XSLT File | Linking Using the DOM

Other Resources

W3C Associating Style Sheets with XML Documents, Version 1.0
Linking Using the DOM

You can apply an XSLT style sheet to an XML document or fragment by calling the `transformNode` method or the `transformNodeToObject` method on the source document or fragment. You can invoke these methods from a script or a COM-enabled programming language, such as C++. The style sheet and the source document or fragment are represented by instances of XML DOM. This means that you need to create two DOM instances, using the appropriate ProgID ("Msxml2.DOMDocument.5.0" for MSXML 5.0). Then, load the source document and the style sheet into the respective DOM instances before calling either of the transformation methods. The following Jscript code snippet illustrates this point:

```javascript
var xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
var xsldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmldoc.load("sample.xml");
if (xmlDoc.parseError.errorCode <> 0) {
    var myErr = xmlDoc.parseError;
    alert("You have error " + myErr.reason);
} else {
    xsldoc.load("sample.xsl");
    output = xmldoc.transformNode(xsldoc);
}
```

Alternatively, you can use:

```javascript
var outdoc = new ActiveXObject("Msxml2.DomDocument.5.0");
xmldoc.transformNodeToObject(xsldoc, outdoc);
```

The output from the two methods differs as follows:

**transformNode**

The output of this method is a string whose value is a textual representation of the result tree. Typically, this value is HTML code for display in a Web page. Or, you can persist the result tree by saving the string into a file.

**transformNodeToObject**

This method places the result of the transformation into another object, which can then be manipulated further (as with a second transformation).

When using either method, the source object does not have to be a complete
Document object. It can be only a node of such an object. If it is only a node, the XSLT processor treats that node, and all of its descendants, as if they were a complete document. Likewise, the XSLT object can be a complete, stand-alone XSLT file, or a node representing an embedded style sheet within an XSLT file.

See Also

transformNode Method | transformNodeToObject Method | Using XSLT with the DOM or SAX | Linking to an XSLT File | Linking Using a Processing Instruction
Using XSLT with Cascading Style Sheets

You can use CSS, either as a stand-alone style sheet or as a part of an XSLT style sheet, to render an XML document in a Web browser. The following topics compare the uses of the two types of style sheets.

- [Using CSS as a Stand-alone Style Sheet](#)
- [Using CSS within an XSLT Style Sheet](#)
- [Comparing XSLT to Cascading Style Sheets](#)
Using CSS as a Stand-alone Style Sheet

CSS can provide a static and restrictive rendering of an XML document for Web display. The order of the source XML elements determines the layout of display. Because you cannot manipulate the source document content with CSS, you have very little run-time control of the display. The following example illustrates how an XML document uses CSS to display the content.

**XML Source document (sample.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/css" href="sample.css"?>
<book-review>
  <book>
    <headline><title>A Good Book</title></headline>
    <headline><author>Robert Brown</author></headline>
    <headline><publisher>The Publisher</publisher></headline>
    <headline><date>Feb. 16, 2000</date></headline>
  </book>
  <Review>
    <title>A Good Book</title> by <author>Rober Brown</author>, published by <publisher>The Publisher</publisher> on <date>Feb. 16, 2000</date>, is indeed a very good book.
  </Review>
</book-review>
```

**Cascading Style Sheet (sample.css)**

```css
title {display:inline; color:red}
author {display:inline; font-style:italic}
publisher {display:inline; color:olive}
date {display:inline; color:blue}
review {display:block; font-size:12}
book {display:block; font-size:16; font-weight:bold}
headline {display:block; margin:1em}
```

Copy both the XML source document and the cascading style sheet, above. Paste them into the files, and save the files as **sample.xml** and **sample.css**, respectively. Open the **sample.xml** with Internet Explorer to see the resulting display.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Using CSS within an XSLT Style Sheet

You can utilize the benefits of both CSS and XSLT by using CSS within an XSLT style sheet. The following example shows you how to do this:

**XML document (sample.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="sample.xsl"?>
<book-review>
  <book>
    <headline><title>A Good Book</title></headline>
    <headline><author>Robert Brown</author></headline>
    <headline><publisher>The Publisher</publisher></headline>
    <headline><date>Feb. 16, 2000</date></headline>
  </book>
  <Review><title>A Good Book</title> by <author>Rober Brown</author>, published by <publisher>The Publisher</publisher> on <date>Feb. 16, 2000</date>, is indeed a very good book.</Review>
</book-review>
```

**XSLT Style Sheet with CSS (sample.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <STYLE>
          .title {color:red}
          .author {font-style:italic}
          .publisher {color:olive}
          .date {color:blue}
        </STYLE>
      </HEAD>
      <BODY>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```
Comparing XSLT to Cascading Style Sheets

Although XPath, CSS, and XSLT can work together to transform an XML data set, they each have a different role. The following is a brief review of these technologies and their roles.

- XPath formulates an expression to select a specified piece of data or data set. An XPath expression typically serves as a pattern to match this specific data in the source document.
- Cascading style sheets provide a centralized and modular way to format HTML output. Although CSS can be linked directly to an XML source document, it is more effective to use CSS as a compliment within XSLT.
- XSLT is the method by which you manipulate and transform (i.e., add, delete, query, sort, etc.) the XML data in a source document. XSLT uses XPath expressions to work with data elements. The output of XSLT transformations is typically another XML document, an HTML page, a text file, or a database entry.

This means that the real purpose of XSLT is not to format data, but to transform and manipulate data. The formatting aspect is best left for CSS, especially for large and complex documents.

If the content of an XML document can be displayed in the order as specified in the source document, you can use CSS without any help from XSLT. To display the output in any other order, you must use XSLT, possibly with the help of CSS.

The following sections summarize the advantages and disadvantages of XSLT and CSS.
Advantages of XSLT

- Can reorder, add, or delete elements. This is because XSLT is aware of the structure of the XML document.
- Can incorporate content from more than one document, or split a single document into many documents.
- Able to access and manipulate the comments, processing instructions, and attribute names and values in an XML document.
- Written in XML, so there is no third language to be learned.
- Offers programming-like logic, which includes an almost infinite variety of conditional statements. Can perform operations, such as arithmetic as well logical operations.
- Can include text and graphics.
- Can generate a variety of output: static or dynamic HTML, XML, RTF, PDF, comma-separated-values (CSV) flat files, or a stream of XSL Formatting Objects (XSL-FO).
Disadvantages of XSLT

- XSLT is not a style language in itself. Therefore, the application that handles the output from XSLT must be able to style it accordingly.
- Requires character escaping to output special markup characters, such as < and &.
- Uses more memory and processor power, because reordering, adding, deleting, and manipulating elements requires a tree representation of the document in memory.
- Is more difficult to learn to use.
Advantages of CSS

- Is easier to use than XSLT.
- Is suitable for simple documents.
- Uses less memory. Because it cannot reorder a document, CSS does not build a tree representation of the document.
- Begins displaying the document as soon as it begins processing the document.
- Has widespread support in the developer community.
Disadvantages of CSS

- Cannot reorder, add, delete, or perform operations on elements.
- Offers no access to non-elements, such as attributes and their values, processing instructions, and so on.
- Cannot style non-textual content (such as images) referred to in an XML document.
- Supports limited features of CSS2 for some applications.
- Uses a different syntax than XML, and therefore requires completely separate parsers and APIs.
- Outputs only data suitable for viewing in a Web browser.

Other Resources

Cascading Style Sheets
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
XSLT Output

Just as CSS brought new richness and interactivity to static HTML pages, XSLT extends the value of XML data. XSLT allows you not only to present XML data, but also to transform it into completely new data that is designed specifically for a particular user, media, or client.

With XSLT, you can do the following:

- **Transform data so it can be exchanged between disparate business systems.**

  XSLT excels at mapping one XML-based representation into another. For example, a book retailer might represent a customer order based on one XML Schema, while a book supplier might represent it with another XML Schema. With XSLT, a customer order taken by the book retailer can be transformed into an XML-based representation that matches the book supplier's XML Schema. Once transformed, the book retailer can send the order to the book supplier in an XML format that the book supplier understands.

- **Transform XML documents into HTML documents for a variety of browsers.**

  XSLT transformations are not only useful for backward compatibility for older browsers. They also transform data so that it can be rendered on the new Internet devices on the market, such as Internet-enabled handheld PCs, mobile phones, and TV set-top boxes.

- **Transform XML data into a variety of media, including HTML documents for the Web and PDF format for print.**

  With XSLT, you can output XML to a variety of media, including HTML for the Web and Package Definition Format (PDF) for print.

- **Dynamically filter information to target a specific user or user group.**
For example, for an XML-based catalog, you can use one style sheet to present one set of information to retail customers, while using another style sheet to present information to wholesale customers.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Template-Driven Transformations

In general, XSLT transformations fall into two categories: template-driven and data-driven. This section provides information about template-driven transformations. The Data-Driven Transformations section follows.

In a template-driven transformation, the sequence of the output is determined by template-rules used in the transformation. The following is an example of a simple template-driven transformation.

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <HTML>
            <HEAD>
                </HEAD>
            <BODY>
                <H1><xsl:value-of select="/books/book/title"/></H1>
                <H3><xsl:value-of select="/books/book/author"/></H3>
                <P>
                    <SPAN>Abstract:</SPAN>
                    <SPAN><xsl:value-of select="/books/book/abstract"/></SPAN>
                </P>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```

Here, the style sheet expects certain data elements from a source document, which might look like this:

```xml
<?xml version="1.0"?>
<books>
    <book>
        <title>Synchronized Jamming</title>
        <author>Kari Hensien</author>
        <abstract>
            A post modern flight of fancy.
        </abstract>
    </book>
</books>
```
The style sheet then decides how the output should appear. This means that a template-driven transformation produces its result by pulling the specified data elements from the source document, and putting them through the appropriate templates.

To process multiple book entries, a template-driven XSLT style sheet can use `<xsl:for-each>` to set up a loop to process each entry. It can also use `<xsl:sort>` to process the entries in specified order, or it can use `<xsl:if>` or `<xsl:choose>` to selectively process some entries based on a certain condition. A template-driven transformation is suitable for processing an information-rich source document, as long as the data is well structured and the result pattern is predictable.

On the other hand, data-driven transformations are ideal for processing a data source when the data is poorly structured, or when the output depends on recursive processing. In practice, template-driven and data-driven transformations are often combined in a single style sheet for a wide variety of XML applications.

The following topics walk you through some common tasks for writing a template-driven XSLT style sheet.

- [Creating and Populating an HTML Template](#)
- [Generating Well-Formed HTML Using XSLT](#)
- [Selecting and Outputting Attributes](#)
- [Sorting XML Using `<xsl:sort>`](#)
- [Defining Conditional Templates Using `<xsl:if>` and `<xsl:choose>`](#)

**See Also**

- [Using XPath Expressions to Select Nodes](#) | [Starter Kit](#) | [Data-Driven Transformations](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Creating and Populating an HTML Template

Consider the XML data in `portfolio.xml` in the example below. The data structure has a repetitive pattern: the top-level element, `<portfolio>`, contains three `<stock>` elements of the same structure. We want to construct a template-driven XSLT style sheet to display all the stocks in a table, so that each row of this table shows the name, symbol, and price for one stock. To do this, we will first create a template of the table in HTML. Then we’ll populate the resulting template with the appropriate XML data values.

The following is the table template in HTML.

```html
<HTML>
  <BODY>
    <TABLE BORDER="2">
      <TR>
        <TD>Symbols</TD>
        <TD>Name</TD>
        <TD>Price</TD>
      </TR>
      <!-- Repeat the following row for each stock. -->
      <TR>
        <!-- symbol goes here -->
        <!-- name goes here -->
        <!-- price goes here -->
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

To populate the above template, you could manually replace the comments with data from the XML file. You can also, however, perform this replacement with XSLT. To do so, use XSLT instructions (such as `<xsl:value-of>`) with XPath expressions to locate data in the XML file, and to insert the data into the HTML template. The following XSLT fragment provides an example.

```xml
<HTML>
  <BODY>
    <TABLE BORDER="2">
      <!-- Populate the template with data from the XML file. -->
  </BODY>
</HTML>
```
The `<xsl:for-each>` instruction sets up a loop that will iterate through all the `<stock>` nodes immediately under `<portfolio>` in the XML data source. For each stock, the `<xsl:value-of>` instructions extract the symbol, name, and price. The results are inserted into the `<TD>` cells of the HTML table. Because this sample contains three `<stock>` elements, the template will generate three rows.

The `select` attribute describes how to find a set of elements in the source document. It takes an XPath expression, which identifies the specified node in the source tree. An XPath expression is also called a pattern. Using an XPath expression is similar to navigating a file system, where a forward slash (/) selects subdirectories relative to the current directory. In an XSLT style sheet, a path can be relative (e.g., "name") or absolute (e.g., "/portfolio/stock/name"). With relative XPath expressions, the navigation starts at the current node and proceeds down into the XML data hierarchy, selecting all the nodes that match the pattern. In this example, the pattern "portfolio/stock" starts at the document root, and drills down through the `<portfolio>` element to select the three `<stock>` children.

For many style sheets, simple XPath expressions of concatenating node names and the `/` operators are sufficient. For others, more complex XPath expressions are needed. For more information, see Introduction to XPath Syntax.

Within the `<xsl:for-each>` element, further navigation and operations can proceed, relative to the selected element. When you drill down to select children of each `<stock>` element, you can use the relative pattern for those children. For
example, in the XSLT fragment above, we used `<xsl:value-of select="name">`, instead of `<xsl:value-of select="/portfolio/stock/name">`.

To turn this HTML template into an XSLT template rule, we wrap the template under the `<xsl:template match="/">` element. This template rule applies to the root node, which serves as the starting point (or the current context) for the enclosing template. For more information, see Defining Match Patterns in `<xsl:template>`.

To obtain the complete XSLT style sheet containing this HTML table template, we need to add an `<xsl:stylesheet>` element to hold the template rule. The complete listing of the XSLT style sheet is shown in `portfolio.xsl`, below. We also added the `<?xml version="1.0"?>` XML declaration to emphasize that the XSLT style sheet is also an XML document.

The entire XSLT file, including the HTML that comprises the template, must be well-formed. For more information about authoring or converting to well-formed HTML, see Generating Well-Formed HTML Using XSLT.

**Example**

To display the data from `portfolio.xml` in a Web browser using `portfolio.xsl`, add the following processing instruction to the XML file, so that the style sheet is linked when the XML file is opened in Internet Explorer.

```xml
<?xml-stylesheet type="text/xsl" href="portfolio.xsl"?>
```

**XML File (portfolio.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="portfolio.xsl"?>
/portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="
    <stock exchange="nyse">
        <name>zacx corp</name>
        <symbol>ZCXM</symbol>
        <price dt:dt="number">28.875</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zaffymat inc</name>
        <symbol>ZFFX</symbol>
        <price dt:dt="number">92.250</price>
```

<stock exchange="nasdaq">
  <name>zysmergy inc</name>
  <symbol>ZYSZ</symbol>
  <price dt:dt="number">20.313</price>
</stock>

XSLT File (portfolio.xsl)

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE BORDER="2">
          <TR>
            <TD>Symbol</TD>
            <TD>Name</TD>
            <TD>Price</TD>
          </TR>
          <xsl:for-each select="portfolio/stock">
            <TR>
              <TD><xsl:value-of select="symbol"/></TD>
              <TD><xsl:value-of select="name"/></TD>
              <TD><xsl:value-of select="price"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>

XSLT Processor Output

<HTML>
  <BODY>
    <TABLE BORDER="2">
      <TR>
        <TD>Symbol</TD>
        <TD>Name</TD>
        <TD>Price</TD>
      </TR>
      <TR>
        <TD>ZCXM</TD>
        <TD>zacx corp</TD>
        <TD>28.875</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCXM</td>
<td>zacx corp</td>
<td>28.875</td>
</tr>
<tr>
<td>ZFFX</td>
<td>zaffymat inc</td>
<td>92.250</td>
</tr>
<tr>
<td>ZYSZ</td>
<td>zysmergy inc</td>
<td>20.313</td>
</tr>
</tbody>
</table>
Generating Well-Formed HTML Using XSLT

Well-formed HTML conforms to the rules of XML. This means that the same HTML tags are applicable, but the stricter XML syntax is required. For example, `<BR>` is not a well-formed HTML tag, but `<BR/>` is. `<H1>...</h1>` is not well-formed, but `<H1>...</H1>` or `<h1>...</h1>` is. An XSLT style sheet is itself XML, and it is important that any HTML within it be well-formed. The following are some basic rules to follow as you write or convert to well-formed HTML.
All tags must be closed

HTML allows certain end tags to be optional, such as <P>, <LI>, <TR>, and <TD>. XML requires all tags to be closed explicitly.

```
<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
</table>
| <P>  This is an HTML paragraph.  
<P>or two. | <P>This is an HTML paragraph.</P>  
<P>or two.</P> |
```

Leaf nodes must also be closed by placing a forward slash (/) within the tag:

```
<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;IMG src=&quot;sample.gif&quot; width=&quot;10&quot; height=&quot;20&quot;&gt;</td>
<td>&lt;IMG src=&quot;sample.gif&quot; width=&quot;10&quot; height=&quot;20&quot;/&gt;</td>
</tr>
</tbody>
</table>
```
No overlapping tags

XML does not allow start and end tags to overlap, but enforces a strict hierarchy within the document.

<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;B&gt;Bold &lt;I&gt;Bold and Italic&lt;/B&gt; Italic&lt;/I&gt;</code></td>
<td><code>&lt;B&gt;Bold&lt;/B&gt; </code>&lt;I&gt;<code>&lt;B&gt;Bold and Italic&lt;/B&gt; Italic&lt;/I&gt;</code></td>
</tr>
</tbody>
</table>
**Case matters**

Choose a consistent case for start and end tags. The examples in this SDK generally use uppercase for HTML elements.

<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;B&gt;&lt;i&gt;Hello!&lt;/i&gt;&lt;/b&gt;</code></td>
<td><code>&lt;B&gt;&lt;I&gt;Hello!&lt;/I&gt;&lt;/B&gt;</code></td>
</tr>
</tbody>
</table>
**Quote your attributes**

All attributes must be surrounded by quotation marks, either single or double.

<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;IMG src=sample.gif width=10 height=20&gt;</code></td>
<td><code>&lt;IMG src='sample.gif' width=&quot;10&quot; height=&quot;20&quot; /&gt;</code></td>
</tr>
</tbody>
</table>
Use a single root

Shortcuts that eliminate the `<HTML>` element as the single top-level element are not allowed.

<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;TITLE&gt;Shortcut markup&lt;/TITLE&gt;</code></td>
<td><code>&lt;TITLE&gt;Clean markup&lt;/TITLE&gt;</code></td>
</tr>
<tr>
<td><code>&lt;BODY&gt;</code></td>
<td><code>&lt;BODY&gt;</code></td>
</tr>
<tr>
<td><code>&lt;P&gt;Amazing that this HTML works.&lt;/P&gt;</code></td>
<td><code>&lt;P&gt;Not nearly so amazing that this well-formed HTML works.&lt;/P&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/BODY&gt;</code></td>
<td><code>&lt;/BODY&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/HTML&gt;</code></td>
<td><code>&lt;/HTML&gt;</code></td>
</tr>
</tbody>
</table>
Use fewer named entities

XML defines only a minimal set of built-in named entities. These are as follows:

- `&lt;` — `<`
- `&gt;` — `>`
- `&amp;` — `&`
- `&quot;` — `"`
- `&apos;` — `'`

Therefore, you should avoid using other named HTML entities. When in doubt, always use the numeric character reference for the character of interest. For example, for non-breaking spaces, use `&#160;` or `&xA0;` instead of `&nbsp;`. For emphatic dashes, use `&#8212;` instead of `&mdash;`. Also, in Internet Explorer, the numeric character reference of `&#151;` is treated as the named entity of `&mdash;`, but will not be resolved to `&#8212;` in MSXML.
Escape script blocks

Script blocks in HTML can contain characters that cannot be parsed, namely < and &. These must be escaped in well-formed HTML by using character entities, or by enclosing the script block in a CDATA section.

In addition, Microsoft® JScript® (compatible with the ECMA 262 language specification) comments terminate at the end of the line, so it is important to preserve the white space within script blocks containing comments. By default, the xml:space attribute value normalizes white space by compressing adjacent white space characters into a single space. This destroys the new line that terminates the JScript comment. Any JScript following the comment is treated as part of the comment and ignored, often resulting in script errors. The CDATA notation also ensures that the white space is preserved.

The following HTML script block contains both an unparsable character (<) and JScript comments. The well-formed script block uses CDATA to encapsulate the script.

<table>
<thead>
<tr>
<th>HTML</th>
<th>Well-formed HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;SCRIPT&gt;</code></td>
<td><code>&lt;SCRIPT&gt;</code>&lt;![CDATA[</td>
</tr>
<tr>
<td>// checks a number against 7</td>
<td>// checks a number against 7</td>
</tr>
<tr>
<td>function less-than-seven(n) { function less-than-seven(n) {</td>
<td></td>
</tr>
<tr>
<td>return n &lt; 7;</td>
<td>return n &lt; 7;</td>
</tr>
<tr>
<td>}</td>
<td>} ]]&gt;`&lt;/SCRIPT&gt;</td>
</tr>
</tbody>
</table>

Not all scripts will fail if they are not escaped in this way. However, it is highly recommended that you habitually escape them. This ensures not only that the script will work if it contains escaped characters or comments now, but that it will continue to work if these characters are added in the future.
Selecting and Outputting Attributes

The XPath syntax for selecting an attribute from a source document is to prefix the @ symbol to the name of the attribute. For example, \(<xsl:value-of select="@exchange"/>\) selects the value of an exchange attribute defined for the current element. The example in this topic extracts the value of the exchange attribute of \(<stock>\) elements, and inserts it into the output. For more information about selecting attributes, see Using XPath Expressions to Select Nodes.

There are two ways to generate an attribute in the result tree. The first way is to add the attribute to an output element. For example, in portfolio-attributes.xsl, below, a BORDER attribute is inserted directly into the output \(<TABLE>\) element. This way is useful if you know the attribute value when you write the style sheet.

Alternatively, you can use the \(<xsl:attribute>\) instruction to create an attribute for any output element. This is useful when you must assign a data value from the source to the attribute at run time. For example, the following code fragment inserts a TITLE attribute to the outgoing \(<TR>\) element.

\<TR>
\t\<xsl:attribute name="TITLE">\<xsl:value-of select="symbol"/>\is listed on the \<xsl:value-of select="@exchange"/>\stock exchange.\</xsl:attribute>\n......\</TR>\n
In this fragment, the attribute value is the content of the \(<xsl:attribute>\) element — that is, whatever is enclosed between the \(<xsl:attribute>\) ... \</xsl:attribute> tags. For the stock with the ZCXM symbol in this example, the TITLE attribute created above has a value of "ZCXM is listed on the nyse stock exchange". This produces the output \(<TR>\) element, as follows:

\<TR TITLE="ZCXM is listed on the nyse stock exchange">\n......\</TR>\n
You can use both of the above methods to add attributes to a single element,
even in a single rule or operation. Be aware, however, of the following limitations.

- You cannot add an attribute to an element that already has an attribute of that name.
- Attributes added with `<xsl:attribute>` must appear before any child elements are added to the element.

Example

This example uses an XML source file and an XSLT style sheet.

XML File (portfolio-attributes.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="portfolio-attributes.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="preserve">
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="NASDAQ">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="NASDAQ">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

XSLT File (portfolio-attributes.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE BORDER="2">
          <TR>
            <TD>Symbol</TD>
            <TD>Name</TD>
```

```
Formatted Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCXI</td>
<td>zacx corp</td>
<td>28.875</td>
</tr>
<tr>
<td>ZFFX</td>
<td>zaffnyat inc</td>
<td>92.250</td>
</tr>
<tr>
<td>ZYSZ</td>
<td>zysmergy inc</td>
<td>20.313</td>
</tr>
</tbody>
</table>
Sorting XML Using `<xsl:sort>`

You can use the `<xsl:sort>` instruction to sort XML elements so that they are processed (and so that the results are output) in a specific order. Sorting is applicable to a template rule (`<xsl:template>`) or a loop (`<xsl:for-each>`). So `<xsl:sort>` elements are declared as children of an `<xsl:apply-templates>` element or an `<xsl:for-each>` element.

The `<xsl:sort>` element has a select attribute whose value is an XPath expression. This attribute defines a sort key. Multiple `<xsl:sort>` elements are allowed for each template rule or each loop. The first `<xsl:sort>` element defines the primary sort key, the second `<xsl:sort>` element specifies the secondary sort key, and so on. In the example below, employees are sorted by their last name. Employees who share the same last name are further sorted by their first name. When two sort keys are identical, the resulting order is the same as the order they appear in the source document.

When used in `<xsl:for-each>` elements, `<xsl:sort>` elements must occur before any other child elements.

You can sort items in either ascending order or descending order. You can specify the sort order by assigning the appropriate string value ("ascending" or "descending") to the order attribute of the `<xsl:sort>` element. The default order is "ascending".

There are three types of sort keys: QNames, text, and numbers. QNames and text are sorted alphabetically. Numbers are sorted by their numerical values. The default data type is text. To specify a data type explicitly, assign "QName", "text", or "number" to the data-type attribute of the `<xsl:sort>` element. When data-type is set to "number", the sort key is first converted to a number. A qualified name with a prefix is expanded into an expanded name. The expanded name identifies the data type.

For more information about sorting, see [Sorting with XSLT](#).

The following example sorts a list of employees by name.
Example

This example uses an XML source file and an XSLT style sheet.

XML File (sortempl.xml)

<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="sortempl.xsl"?>
<employees>
  <employee>
    <name>
      <given>Loni</given>
      <family>Ota</family>
    </name>
  </employee>
  <employee>
    <name>
      <given>Leonard</given>
      <family>Zuvela</family>
    </name>
  </employee>
  <employee>
    <name>
      <given>John</given>
      <family>Chen</family>
    </name>
  </employee>
  <employee>
    <name>
      <given>Jay</given>
      <family>Dirkson</family>
    </name>
  </employee>
  <employee>
    <name>
      <given>Anthony</given>
      <family>Dirkson</family>
    </name>
  </employee>
  <employee>
    <name>
      <given>Tony</given>
      <family>Dirkson</family>
    </name>
  </employee>
</employees>

XSLT File (sortempl.xsl)
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="employees">
    <ul>
      <xsl:apply-templates select="employee">
        <xsl:sort select="name/family"/>
        <xsl:sort select="name/given"/>
      </xsl:apply-templates>
    </ul>
  </xsl:template>

  <xsl:template match="employee">
    <li>
      <xsl:value-of select="name/given"/>
      <xsl:text>	</xsl:text>
      <xsl:value-of select="name/family"/>
    </li>
  </xsl:template>
</xsl:stylesheet>

**Formatted Output**

- John Chen
- Anthony Dirkson
- Jay Dirkson
- Tony Dirkson
- Loni Ota
- Leonard Zuvela

**See Also**

[Qualified Names](#) | [Expressions](#)
Defining Conditional Templates Using `<xsl:if>` and `<xsl:choose>`

Conditional templates are used to produce result tree fragments when certain conditions exist within the source document. Conditional templates are defined with the `<xsl:if>` and `<xsl:choose>` elements.

This section contains the following topics.

- [Defining Conditional Templates Using `<xsl:if>`](#)
- [Defining Conditional Templates Using `<xsl:choose>`](#)
Defining Conditional Templates Using `<xsl:if>`

The XML file in the example below contains portfolio data. The `<stock>` element has an `exchange` attribute. Suppose you want to generate some output only when this attribute has a certain value. You can create another row in the table, and put the attribute value in this row. Alternatively, you can indicate stocks from a particular exchange by noting them with an asterisk (*).

The `<xsl:if>` element provides a mechanism for conditionally inserting structure into the output tree. In the following example, the `<xsl:if>` element inserts an asterisk after the symbol for those stocks listed on the NASDAQ stock exchange. The contents of `<xsl:if>` can be simple text, as in this example, or any structure allowed by XSLT, such as elements or attributes.

In the sample XSLT file below, the `test` attribute takes a pattern. If the query described by the pattern selects one or more nodes, the `<xsl:if>` template is inserted. If the selection is empty, the conditional is skipped. In this case, the query checks to see if the `<stock>` element has an `exchange` attribute. If so, the query then checks to see if the value of the `exchange` attribute is equal to "nasdaq".

For more information about qualifying a pattern using brackets ([ ]) and constructing patterns that compare values, see [Introduction to XPath Syntax](#).

**Example**

**XML File (portfolio-if.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="portfolio-if.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="
    <stock exchange="nyse">
        <name>zacx corp</name>
        <symbol>ZCXM</symbol>
        <price dt:dt="number">28.875</price>
    </stock>
```
XSLT File (portfolio-if.xsl)

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE BORDER="2">
          <TR>
            <TD>Symbol</TD>
            <TD>Name</TD>
            <TD>Price</TD>
          </TR>
          <xsl:for-each select="portfolio/stock">
            <TR>
              <TD><xsl:value-of select="symbol"/>
                <xsl:if test="@exchange[.='nasdaq']">*</xsl:if></TD>
              <TD><xsl:value-of select="name"/></TD>
              <TD><xsl:value-of select="price"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
        <P>* Listed on Nasdaq stock exchange</P>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>

Formatted Output
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCXM</td>
<td>zacx corp</td>
<td>28.875</td>
</tr>
<tr>
<td>ZFFX*</td>
<td>zafymat inc</td>
<td>92.250</td>
</tr>
<tr>
<td>ZYSZ*</td>
<td>zysmergy inc</td>
<td>20.313</td>
</tr>
</tbody>
</table>

* Listed on Nasdaq stock exchange
Defining Conditional Templates Using <xsl:choose>

The <xsl:choose> element provides a mechanism for either/or processing: <xsl:choose> contains a series of <xsl:when> elements that are tested in order from top to bottom until a match is found. An <xsl:otherwise> element can be used to insert a template if no match is found.

The following code can be added to the example in the previous topic to color-code the rows by price. Rows 0-25 are displayed in green, rows 25-50 are displayed in blue, and rows 50 and higher are displayed in red. The color is changed by conditionally generating a portion of the value of the STYLE attribute on the table row.

<TR>
  <xsl:attribute name="STYLE">color:
    <xsl:choose>
      <xsl:when test="price[. &lt; 25]">green</xsl:when>
      <xsl:when test="price[. &lt; 50]">blue</xsl:when>
      <xsl:otherwise>red</xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
  <TD>
Example

This example uses an XML source file and an XSLT style sheet.

XML File (portfolio-choose.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="portfolio-choose.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
```
<name>zaffymat inc</name>
<symbol>ZFFX</symbol>
<price dt:dt="number">92.250</price>
</stock>
<stock exchange="nasdaq">
 <name>zysmergy inc</name>
 <symbol>ZYSZ</symbol>
 <price dt:dt="number">20.313</price>
</stock>
</portfolio>

XSLT File (portfolio-choose.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE BORDER="2">
          <TR>
            <TD>Symbol</TD>
            <TD>Name</TD>
            <TD>Price</TD>
          </TR>
          <xsl:for-each select="portfolio/stock">
            <TR>
              <xsl:attribute name="STYLE">color:
                <xsl:choose>
                  <xsl:when test="price[. &lt; 25]">green</xsl:when>
                  <xsl:when test="price[. &lt; 50]">blue</xsl:when>
                  <xsl:otherwise>red</xsl:otherwise>
                </xsl:choose>
              </xsl:attribute>
              <TD><xsl:value-of select="symbol"/></TD>
              <TD><xsl:value-of select="name"/></TD>
              <TD><xsl:value-of select="price"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
        <P>* Listed on Nasdaq stock exchange</P>
        <P>Price key: <SPAN STYLE="color:green">less than 25</SPAN>,
          <SPAN STYLE="color:blue">25-50</SPAN>,
          <SPAN STYLE="color:red">50+</SPAN>.</P>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```
## Formatted Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCXM</td>
<td>zacx corp</td>
<td>28.875</td>
</tr>
<tr>
<td>ZFFX*</td>
<td>zaffymat inc</td>
<td>92.250</td>
</tr>
<tr>
<td>ZYSZ*</td>
<td>zysmergy inc</td>
<td>20.313</td>
</tr>
</tbody>
</table>

*Listed on Nasdaq stock exchange

**Price key:** less than 25, 25-50, 50+.  

Data-Driven Transformations

Template-Driven Transformations are useful for processing source documents with regular and homogenous data structures. Data-driven transformations, however, are usually better for processing source documents with complex and irregular data structures. The following book review article is an example of an irregular data structure.

<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="book-review.xsl"?>
<book-review>
    <title>A Good Book</title> by <author>The Good Writer</author>,
published by <publisher>The Publisher</publisher> on <date>A Good Day</date>,
is indeed a good book. However, the one titled <title>
A Bad Book</title> by the same publisher is very bad. This review
is left to wonder whether this is because <title>A Bad Book</title>
was written by <author>A Bad Author</author>, or because it was
published on <date>A Bad Date</date>.
</book-review>

The elements contained in <book-review> can be arbitrary, and there is no
intrinsic order of elements. The best way to display the content in this case is to
follow the order in which the elements appear in the source document. To do
this, you cannot set up templates to decide where different data elements appear
in the output, and feed the source document into the predefined template, as you
would in a template-driven transformation. In a data-driven transformation,
templates are reactive — that is, they are called whenever a matching data
element is encountered.

The following is an example of a data-driven XSLT style sheet for displaying the
book review article above.

XSLT File (book-review.xsl)

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/"
        <HTML>
            <HEAD>
            </HEAD>
```
The following topics discuss data-driven transformations in more detail.

- Handling Irregular Data Hierarchies
- Defining Match Patterns in `<xsl:template>`
- Debugging a Style Sheet
- Using the Built-in Template Rules
- Translating from One XML Schema to Another
- Generating Comments, Processing Instructions, and Elements
Handling Irregular Data Hierarchies

One of the benefits of using XML is that it allows you to model irregular data hierarchies, including data with the following characteristics.

- Collections of heterogeneous elements
- Structures with many optional elements
- Structures where the order is not apparent
- Recursive structures
- Structures with complex containment requirements

The `pole.xml` example in this section contains heterogeneous collections of elements, with a complex set of containment requirements. A `<section>` element can contain an arbitrary collection of `<title>` elements, `<p>` elements, `<list>` elements, and so on. Many elements are indeed optional: A `<section>` does not have to contain `<p>` or `<list>` elements, or other `<section>` elements. It is important to preserve the order of most elements in the output — that is, the first `<section>` comes before the second `<section>`. The structure is recursive because a `<section>` element can contain other `<section>` elements. The `<emph>` element is allowed anywhere. For example, you might add it to the `<title>` element, as follows:

```xml
<title>To the Pole... <emph>and Back!</emph></title>
```

The mechanism for handling data-driven transformations is similar to event-handling in traditional programming languages. Template rules or fragments are analogous to event handlers, which are called whenever the specified events occur. An element matched in the source document triggers a call to the specified template rule or fragment. A template rule or fragment can be called either by its name or according to the matched element type.

The next topic provides a [Dissection of the pole.xml Example].
Dissection of the pole.xml Example

Before we try the pole.xml example, let's dissect it, bit by bit.

```
<xsl:template match="/">
  <HTML>
    <BODY>
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>
```

This is the template rule for the root element of the source document. It states that all the child elements will be processed according to the template rules defined for them, either explicitly or implicitly. Notice that this rule does not stipulate the order in which the child elements are processed. There are three children under the top-level element: one <title> element and two <section> elements. They will be processed in the order in which they appear in the source document.

The complexity of the <section> elements is similar to that of the root element. Each contains various child elements without any apparent order. Therefore, the elements can be processed in a similar way:

```
<xsl:template match="section">
  <DIV>
    <xsl:apply-templates />
  </DIV>
</xsl:template>
```

Because a <section> element can contain other <section> elements recursively, the above template rule will be applied recursively.

The <title> element appears as a child element of both <document> and <section>. If you want all the titles to appear in the same way, use a template rule similar to following:

```
<xsl:template match="title">
  <H1><xsl:value-of select="text()"/></H1>
</xsl:template>
```
Here we have assumed that the <title> element is a text node. If <title> contains other elements as well, we need to modify the above template so it can handle either case:

```xml
<xsl:template match="title">
    <H1><xsl:apply-templates /></H1>
</xsl:template>
```

This will work even if <title> is still a text node, and you do not explicitly define a template rule for a text node. The default template rule for a text node will be called instead.

The template-rule above matches any <title> found in the source document. However, you might want to distinguish the document title from the section titles. To do this, we need two template rules:

```xml
<xsl:template match="document/title">
    <H1><xsl:apply-templates /></H1>
</xsl:template>

<xsl:template match="section/title">
    <H2><xsl:apply-templates /></H2>
</xsl:template>
```

To complete the transformation of the source document, we also need the following template rules for the remaining elements:

```xml
<xsl:template match="p">
    <P><xsl:apply-templates /></P>
</xsl:template>

<xsl:template match="list">
    <UL>
        <xsl:for-each select="item">
            <LI><xsl:apply-templates /></LI>
        </xsl:for-each>
    </UL>
</xsl:template>

<xsl:template match="emph">
    <I><xsl:apply-templates /></I>
</xsl:template>
```

Finally, we need an explicit definition of the template rule for a text node:
The following topics provide more information about handling irregular data hierarchies.

- [Combining Data-Driven and Template-Driven Transformations](#)
- [Example of Transforming Irregular Data Hierarchies](#)
Combining Data-Driven and Template-Driven Transformations

In practice, it might be most effective to combine data-driven transformations with template-driven ones. The following is an example.

```xml
<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE><xsl:value-of select="document/title"/></TITLE>
    </HEAD>
    <BODY>
      <H1><xsl:value-of select="document/title"/></H1>
      <xsl:apply-templates select="document/section"/>
    </BODY>
  </HTML>
</xsl:template>
```

Here, the template rules for the top-level element are template-driven. They stipulate that, in the output, the title of the document always precedes the sections of the document — even when the order is different in the source document. The data-driven transformations apply only to the sections of the source document.

The following topic provides the complete source files and output for this example.

- [Example of Transforming Irregular Data Hierarchies (pole.xml)](pole.xml)
Example of Transforming Irregular Data Hierarchies

This example illustrates the data-driven model of XSLT processing. With the data-driven model, you can create isolated templates for the types of nodes you expect to see in the output, without too much consideration of their structure. In places where the structure is locally known, you can use `<xsl:for-each>` and `<xsl:value-of>` to populate the template. For example, `<list>` and `<item>` elements appear in a regular and predictable structure. The capacity to switch smoothly between data-driven and template-driven transformation is an important feature of XSLT.

XML File (pole.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="poleeasy.xsl"?>
<document>
  <title>To the Pole and Back</title>
  <section>
    <title>The First Day</title>
    <p>It was the <emph>best</emph> of days, it was the <emph>worst</emph> of days.</p>
    <list>
      <item><emph>best</emph> in that the sun was out.</item>
      <item><emph>worst</emph> in that it was 39 degrees below zero.
    </list>
  </section>
  <section>
    <title>Lunch Menu</title>
    <list>
      <item>ice cream</item>
      <item>popsicles</item>
    </list>
  </section>
  <section>
    <title>The Second Day</title>
    <p>Ditto the first day.</p>
  </section>
</document>
```

XSLT File (poleeasy.xsl)
<?xml version='1.0'?><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<HTML>
<HEAD>
<TITLE><xsl:value-of select="document/title"/></TITLE>
</HEAD>
<BODY>
<H1><xsl:value-of select="document/title"/></H1>
<xsl:apply-templates select="document/section"/>
</BODY>
</HTML>
</xsl:template>
<xsl:template match="section">
<DIV>
<H2><xsl:value-of select="title"/></H2>
<xsl:apply-templates />
</DIV>
</xsl:template>
<xsl:template match="p">
<P><xsl:apply-templates /></P>
</xsl:template>
<xsl:template match="list">
<UL>
<xsl:for-each select="item">
<LI><xsl:apply-templates /></LI>
</xsl:for-each>
</UL>
</xsl:template>
<xsl:template match="emph">
<I><xsl:apply-templates /></I>
</xsl:template>
<xsl:template match="text()">
<xsl:value-of select="."/>
</xsl:template>
<xsl:template match="title"/>
</xsl:stylesheet>

Formatted Output
To the Pole and Back

The First Day

It was the best of days, it was the worst of days.

- best in that the sun was out.
- worst in that it was 39 degrees below zero.

Lunch Menu

- ice cream
- popsicles

The Second Day

Ditto the first day.

Processor Output

The white space has been adjusted for clarity.

<HTML>
<HEAD>
<TITLE>To the Pole and Back</TITLE>
</HEAD>
<BODY>
<H1>To the Pole and Back</H1>
<DIV>
<H2>The First Day</H2>
<P>It was the <I>best</I> of days, it was the <I>worst</I> of days.
</P>
<UL>
<LI><I>best</I> in that the sun was out.</LI>
<LI><I>worst</I> in that it was 39 degrees below zero.</LI>
</UL>
</DIV>
<DIV>
<H2>Lunch Menu</H2>
<UL>
<LI>ice cream</LI>
<LI>popsicles</LI>
</UL>
</DIV>
<DIV>
<H2>The Second Day</H2>
<P>Ditto the first day.</P>
</DIV>
</BODY>
</HTML>
Defining Match Patterns in `<xsl:template>`

The `match` attribute on the `<xsl:template>` element contains a `pattern` expression. The syntax is the same as that used to select nodes with `<xsl:for-each>`, `<xsl:value-of>`, and `<xsl:apply-templates>` elements. However, the pattern is used in quite a different way.

In a select pattern, the pattern describes a query down from a particular node to locate a new set of nodes. A `match` pattern is quite different. The `match` pattern does not locate anything new, but rather compares a specific node against a pattern to see if the node matches that pattern, and thus whether or not to use a particular template.

Here are some of the differences between the use of `match` patterns and select patterns.

<table>
<thead>
<tr>
<th><strong>Select pattern</strong></th>
<th><strong>Match pattern</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Starts with a node (the context), and identifies more nodes.</td>
<td>Starts with a node (the target), and yields a true/false (match/didn't match) result.</td>
</tr>
<tr>
<td>Starts at the context and drills down into the data.</td>
<td>Starts by testing the node against the node target, and then checks the context, including ancestry and descendants.</td>
</tr>
<tr>
<td>Represents a query into an XML document.</td>
<td>Represents a pattern of contextual information against which a particular node can be tested. Match patterns are quite similar to selectors in cascading style sheets (CSS).</td>
</tr>
</tbody>
</table>

The simplest match pattern is just a single element name, such as "section", or node type, such as "text()". If a text node matches the "text()" pattern, the associated template will be used. If a `<section>` element matches the "section" pattern, the associated template will be used.
Matches become more complex when they are used to test not only a node, but also the context in which the node appears in the source document. The pattern "section/title" will match <title> elements that are children of <section> elements. Any <title> elements that are not within a <section> (for example, the document title) do not match this pattern, and a separate template must be defined (such as "document/title"). The target of the pattern (the term farthest to the right) represents the node to be tested. This is similar to a select pattern in which the target represents the nodes to select.

The following table lists some match patterns and their meanings.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning when used as a match pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>match=&quot;title&quot;</td>
<td>Matches &lt;title&gt; elements.</td>
</tr>
<tr>
<td></td>
<td>Matches &lt;title&gt; elements that are children of &lt;section&gt; elements.</td>
</tr>
<tr>
<td></td>
<td>Matches &lt;title&gt; elements contained within &lt;section&gt; elements.</td>
</tr>
<tr>
<td></td>
<td>Matches &lt;title&gt; elements that are children of &lt;section&gt; elements, and that have a short-name attribute.</td>
</tr>
</tbody>
</table>

Matches
For more information, see Introduction to XPath Syntax and XPath Syntax.

You can add another template to the pole.xml sample to provide different formatting (H3 instead of H2) for sections that appear in the context of another section. For example:

```xml
<xsl:template match="section">
  <DIV>
    <H2><xsl:value-of select="title"/></H2>
    <xsl:apply-templates />
  </DIV>
</xsl:template>
```
Notice that a \texttt{<section>} within a section will match the "section" pattern as well as the "section/section" pattern. Which template will be selected? The rule is as follows: Templates later in the style sheet override earlier templates. To ensure that the XSLT processor checks the "section/section" pattern first, place it farther down in the style sheet. Most of the templates in this sample are mutually exclusive, so their order is unimportant.
Debugging a Style Sheet

Because the interaction of a style sheet with a data file can be complex, and style sheet errors are not always obvious, this section provides troubleshooting advice for the style sheet author.

Developing in the browser can be difficult, because the error can be in so many places and can be hidden by the browser. To see error messages, you can use a script at the command prompt, or run the MSXSL command prompt utility. For more information, see XSLT Utilities and Samples You Can Download.

The following are some common scenarios for debugging.

**Problem:**

No output is produced.

**Solution:**

When you view an XML file with an XSLT style sheet in a browser, errors in the source file or the style sheet are reported. These errors include XML that is not well-formed, as well as XSLT syntax errors and run-time errors. These errors are not automatically reported when you use XSLT from script or on data islands. For sample code to detect, format, and report these errors, see Performing Error Handling with XSLT. Alternatively, use direct browsing as a development tool for reporting errors. For more information, see Using the Default Style Sheet.

If no errors are reported, the problem is most likely an error in the style sheet design, which can be diagnosed and corrected by the style sheet author. Begin by verifying that the root template of the style sheet is being executed. To do this, use the method outlined in the following section.

**Problem:**

The expected template is not being used.

**Solution:**
If the contents of a template, including the root template, are not appearing in the output, check to see if the template is being called by adding an explicit trace message.

```xml
<xsl:template match="address">
   !! inside address template !!
   <xsl:apply-templates select="zipcode"/>
</xsl:template>
```

If the trace message appears in the output, the template is being executed, and the problem is likely that this `<xsl:apply-templates>` is not returning anything.

If the trace message does not appear, the problem might be that the `match` pattern is in error. Check the spelling and case of element and attribute names, and check that the full context for the match is present in the source data. Also determine which `<xsl:apply-templates>` element should invoke the template and ensure that it is selecting the correct nodes.

If the template uses the root pattern (`/`) and is not being called, check that no template later in the style sheet uses the root pattern, or omits the `match` attribute (matching all nodes, including the document root). If the style sheet is being executed against a node other than the document root, a template matching that element is used instead of the root template.

**Problem:**

No results are generated from the `select` pattern.

**Solution:**

One of the most frequent errors preventing output is `select` patterns that do not match the input data. Fortunately these are easy to fix. The debug line added to the example below uses `<xsl:value-of>` to display the content of the `<address>` element.

```xml
<xsl:template match="address">
   <pre><xsl:value-of select="."/></pre>
   <xsl:apply-templates select="zipcode"/>
</xsl:template>
```

When comparing the actual XML data to the `select` pattern, you might find spelling mistakes in element and attribute names. For example, the data could
show <zip-code> or <zipCode> as the correct element name. The context of the query can be checked: Is <zipcode> a child of <address>/? The source data can also be checked for accuracy: Does this <address> have a <zipcode>?

For complex patterns, it is often helpful to progressively simplify the pattern to isolate the problem.

**Problem:**

Elements with the XSLT namespace appear in the output.

**Solution:**

Elements from the XSLT namespace appearing in the output indicate that the XSLT namespace declaration is using an incorrect URL. The XSLT processor does not recognize these elements as XSLT elements, and therefore treats them as output elements.

The correct namespace identifier for XSLT is "http://www.w3.org/1999/XSL/Transform".

**Problem:**

You need to highlight the elements which are not handled by a template.

**Solution:**

Often it is helpful to author a style sheet incrementally, adding templates for elements one at a time. Adding this template to your style sheet will show you visually those elements that do not yet have templates associated with them.

```xml
<xsl:template match="*">
  <span style="background-color:yellow">
    <xsl:attribute name="title">&lt;xsl:value-of select="."/&gt;&lt;/xsl:attribute>
    <xsl:apply-templates/>
  </span>
</xsl:template>
```

Add this template near the top of your style sheet so that it does not override templates that handle specific elements. This template will display a yellow background behind the content of any element without a specific template in existence. Running the mouse over the yellow areas will display a ToolTip
showing those elements that must still be handled by templates.

See Also

Displaying Transformation Results | Using the Default Style Sheet
Using the Built-in Template Rules

MSXML versions 4.0 and later support the following two built-in template rules, as recommended by the November 1999 XSLT Recommendation. These templates make it easier to create style sheets for irregular data by passing text from the source document to the output automatically.

The first such built-in template rule is for element nodes and the root node. It works like this:

```xml
<xsl:template match="/|*">
    <xsl:apply-templates/>
</xsl:template>
```

The second built-in template rule is for text and attribute nodes. It works as follows:

```xml
<xsl:template match="text()">
    <xsl:value-of select="."/>
</xsl:template>
```

The built-in template for processing instructions and comments does nothing, as shown in the following:

```xml
<xsl:template match="processing-instruction()|comment()"/>
```

On their own, these templates provide for a complete, though simple, transformation of the source document. Elements and the document root are traversed, although they do not generate any output themselves. The values of text nodes and CDATA sections are passed through. Comments and processing instructions are not processed and thus do not appear in the output. The end result is that these templates by themselves strip all markup from the source document but preserve the text.

To override these built-in templates, specify your own templates in the style sheet so that every node is handled.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Translating from One XML Schema to Another

An XSLT style sheet can be applied to translate data from one XML grammar to another. Suppose that you receive stock quotes marked up in the following format:

```xml
<?xml version="1.0"?>
<investments>
  <item type="stock" exch="nyse" symbol="ZCXM" company="zacx corp" price="28.875"/>
  <item type="stock" exch="nasdaq" symbol="ZFFX" company="zaffymat inc" price="92.250"/>
  <item type="stock" exch="nasdaq" symbol="ZYSZ" company="zysmergy inc" price="20.313"/>
</investments>
```

You want to add this information to your portfolio of the following format.

```xml
<?xml version="1.0"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>
```

This amounts to translating from one XML schema to another. You can use XSLT to transform the `<investments>` data into the `<portfolio>` grammar. When you do this, your templates will contain output elements such as `<stock>`, instead of HTML elements. The following XSLT style sheet is an example.
Notice that in the resulting document, the `<portfolio>` element generated by this XSLT also declares a namespace for data types used to scope the `dt` attribute in the enclosing `<price>` element. The `dt` namespace prefix that precedes the `dt` attribute (`dt:dt="number"`) indicates that the number type is defined in `urn:schemas-microsoft-com:datatypes`.

The translation can be performed in reverse with a similar style sheet, converting `<portfolio>` data to the `<investments>` grammar.
Note that `<xsl:attribute>` is used not only to copy an attribute value to a new attribute, but also to convert element content to attribute values.
Generating Comments, Processing Instructions, and Elements

When generating XML, it is often useful to generate comments, processing instructions, and elements that are named programmatically. XSLT provides a set of commands for creating specific kinds of nodes in the output. Just as the 
<xsl:attribute> element can insert an attribute into the tree, XSLT provides a similar set of elements for inserting other types of nodes. For more information about the <xsl:attribute> element, see Selecting and Outputting Attributes.

The <xsl:comment> element inserts a comment into the output. Comments within the style sheet are not passed through to the output, but are treated as comments on the style sheet itself. When a comment is required in the output, place the comment text within an <xsl:comment> element.

The <xsl:processing-instruction> element inserts a processing instruction into the output. The name attribute specifies the name of the attribute, and the content of the element becomes the text of the processing instruction.

The <xsl:element> element provides an alternative mechanism for creating elements in the output. The name attribute specifies the name of the element. The following two ways of creating a <DIV> element produce identical results.

1)  <DIV class="menuItem">
    Choose me
  </DIV>

2)  <xsl:element name="DIV">
    <xsl:attribute name="class">menuItem</xsl:attribute>
    Choose me
  </xsl:element>

You must use the <xsl:attribute> element to add attributes to elements that have been created using the <xsl:element> element. You cannot add attributes to these elements directly.

Because XSLT allows output elements to be specified directly, there are only a few situations in which <xsl:element> is useful. For example, you can use
<xsl:element> as an escaping mechanism for creating XSLT elements in the output, which allows a style sheet to generate other style sheets.

**Example**

The following example converts style sheets to use the <xsl:element> syntax. It uses <xsl:element> to generate elements in the XSLT namespace, and inserts processing instructions and a comment at the beginning of the file.

This style sheet has five template rules.

1. The first template adds a boilerplate to the root of the output document. It also selects all comments and the document element to be copied to the output.
2. The second template copies comments, processing instructions, and text nodes to the output.
3. The third template converts elements to the <xsl:element> syntax.
4. The fourth template converts attributes to the <xsl:attribute> syntax.
5. The fifth template ensures that any elements from the XSLT namespace are copied to the output with their attributes, and not converted by the third template.

**XSLT File (convert.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="/">
        <xsl:processing-instruction name="xml-stylesheet" type="text/xsl" href="style.xsl"/>
        <xsl:comment>Style sheet converted automatically to &lt;xsl:element&gt; syntax</xsl:comment>
        <xsl:apply-templates select="comment()"/>
        <xsl:apply-templates select="*"/>
    </xsl:template>

    <!-- Copy text, comments and processing instructions. -->
    <xsl:template match="comment() | processing-instruction() | text()">
        <xsl:copy>
            <xsl:apply-templates />
        </xsl:copy>
    </xsl:template>
```
When you apply this XSLT style sheet to the following XML document,

**XML File (convert.xml)**

```xml
<?xml version="1.0"?>
<book>
  <title>title</title>
  <author>
    <last-name>last name</last-name>
    <first-name>first name</first-name>
  </author>
  <publisher>publisher</publisher>
  <date>date</date>
</book>
```
<price currency="usd">10.0</price>
</book>

you get the following output from the XSLT processor.

**XLST Processor Output:**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<!-- Style sheet converted automatically to <xsl:element> syntax -->
<xsl:element name="book" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">
    <xsl:element name="title">title</xsl:element>
    <xsl:element name="author">
        <xsl:element name="last-name">last name</xsl:element>
        <xsl:element name="first-name">first name</xsl:element>
    </xsl:element>
    <xsl:element name="publisher">publisher</xsl:element>
    <xsl:element name="date">date</xsl:element>
    <xsl:element name="price">
        <xsl:attribute name="currency">usd</xsl:attribute>
        10.0
    </xsl:element>
</xsl:element>
```
Displaying Transformation Results

This section contains the following topics.

- Displaying XML Files in a Browser
- Using the Default Style Sheet
- Using an External Style Sheet
- Using Cascading Style Sheets to Display Results
- Using Namespaces with CSS
- XSLT Requirements for Viewing XML in a Browser
- Accessing XML and XSLT Documents from Script
- Serving Out XML Files from an HTTP Server
Displaying XML Files in a Browser

You can use Microsoft® Internet Explorer 5.0 or later to view XML documents in the browser, just as you would view HTML pages. Unlike HTML, XML does not predefine display properties for specific elements. Therefore, XML requires a separate style sheet that specifies how the XML data should be displayed. This separation of XML content from its presentation allows the content to be easily repurposed.

To view an XML file in Internet Explorer, you can specify a style sheet the following ways.

- Use the default style sheet, which presents the file as a collapsible tree.
- Specify a specific style sheet at the top of the XML file, in an `href` attribute such as the following:

  ```xml
  <?xml-stylesheet type="text/xsl" href="myfile.xsl" ?>
  ```

- Specify any style sheet at the command prompt, for example:

  ```bash
  c:\bat\msxsl mydata.xml mytransform.xsl -o myoutput.html
  ```

See Also

[Using the Default Style Sheet](#) | [Using an External Style Sheet](#) | [XSLT Utilities and Samples You Can Download](#)
Using the Default Style Sheet

If an XML document does not contain a valid style sheet processing instruction, Microsoft® Internet Explorer 5.0 or later displays the XML as an expanding and collapsing tree.

```xml
<?xml version="1.0" ?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <release_years>1998 2000</release_years>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
  <book id="bk102">
    <author>Rolls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <release_years>2000</release_years>
    <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
  </book>
</catalog>
```

This tree view of the XML data is a convenient way to check that your XML is well-formed. To check a document, simply open the file in Internet Explorer. If there are any errors caused by the XML data not being well-formed, Internet Explorer lists these errors, with their line numbers, onscreen.

**Tip** You can also use Internet Explorer to check that your XSLT style sheets and XML Schema documents are well-formed.

Internet Explorer displays an XML document asynchronously. This means that it starts displaying the XML document before the document has been completely downloaded. If an error occurs toward the end of a long document, most of the document might already be onscreen before Internet Explorer inserts an error
message. In this case, you might have to scroll to the bottom of the page to view the error message.

**Note**  When displaying XML documents, Internet Explorer loads the specified document type definition (DTD) or XML Schema, but does not report validation errors.

**See Also**

[Using an External Style Sheet](#) | [Debugging a Style Sheet](#)
Using an External Style Sheet

In addition to its default style sheet, Microsoft® Internet Explorer supports style sheets written as either cascading style sheets or XSLT files. To use one of these style sheets to display an XML document, you must include a style sheet processing instruction in the XML document that indicates the type and location of the style sheet.

**Note**  Although Internet Explorer automatically uses the style sheet processing instruction to display XML directly in the browser, other uses of XML, such as data islands, ignore this instruction.

A style sheet processing instruction must be at the beginning of an XML document, following the XML declaration (if there is one) and the document type declaration (if there is one), as shown in the following XML document.

```xml
<?xml version="1.0" ?>
<!DOCTYPE region >
<?xml-stylesheet type="text/css" href="region.css" ?>
<region>
  <name>Western Region</name>
  <quarter number="1" books_sold="24000"/>
  <quarter number="2" books_sold="38600"/>
</region>
```

An XML document can include multiple style sheet processing instructions. Internet Explorer 5.0 looks first for a processing instruction of type text/xsl and uses the first one it finds. Otherwise, it searches for text/css processing instructions and cascades them together using the same process as the HTML `<LINK>` element. Other attributes that are allowed on the `<LINK>` element can also be specified in the style sheet processing instruction.

For the World Wide Web Consortium (W3C) recommendation for style sheet processing instruction syntax, see [Associating Style Sheets with XML Documents](#).
Using the Default Style Sheet
Using Cascading Style Sheets to Display Results

For Web-based XML applications, it is often important not only to transform XML into HTML (a primary use of XSLT), but also to format the resulting HTML. An XSLT transformation can easily emit `<FONT>` elements, `bgcolor` attributes for `<BODY>` elements, and so on, just as easily as it can emit other HTML.

For a number of important reasons, however, your HTML code will benefit from using cascading style sheets (CSS) to control the appearance of a page and other aspects of presentation. Luckily, you can combine the transformational strengths of XSLT with the formatting capabilities of CSS.

This section contains the following topics.

- Incorporating `<STYLE>` Elements into an XSLT File
- Using an External Cascading Style Sheet
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Incorporating `<STYLE>` Elements into an XSLT File

An XSLT style sheet can emit HTML `<STYLE>` elements, including CSS specifications, directly into the HTML that results from the XSLT transformation. This option works best when the number of CSS rules is small and easily managed.

To place a `<STYLE>` element and its CSS rules into the result tree of a transformation, you can include them in the XSLT template that instantiates the HTML `<HEAD>` element. The `book.xsl` file below illustrates this.

In `book.xsl`, the first template rule matches the root element of the XML document, `<book>`. When it finds a match in the source tree, it instantiates `<HTML>` and `<HEAD>` elements in the result tree.

The `<HEAD>` element includes both a `<TITLE>` element and a `<STYLE>` element. The `<STYLE>` element includes a single CSS rule, which says to display all `<H1>` elements in the result tree in the indicated font.

Finally, this template rule says to instantiate a `<BODY>` element, and then to process any children of the matching `<book>` element. This is the purpose of the `<xsl:apply-templates/>` element. During this process, the template rule also checks for other matching template rules, and transforms them as necessary.

The second template rule is invoked by the `<xsl:apply-templates/>` element of the first template rule. This second template rule instantiates an `<H1>` element. The content of the `<H1>` element is that of a `<book_title>` element in the source tree.

The result of applying this transformation to the sample source document is shown below.

**Example**

**XML File (book.xml)**
XSLT File (book.xsl)

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="book">
    <HTML>
      <HEAD>
        <TITLE>Book Info</TITLE>
        <STYLE>
          H1 {font-family: Arial, Univers, sans-serif; font-size: 36pt }
        </STYLE>
      </HEAD>
      <BODY><xsl:apply-templates/></BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="book_title">
    <H1><xsl:value-of select="."/></H1>
  </xsl:template>
</xsl:stylesheet>

XSLT Processor Output

<html>
  <head>
    <title>Book Info</title>
    <style>
      h1 {font-family: Arial, Univers, sans-serif; font-size: 36pt }
    </style>
  </head>
  <body>
    <h1>Jambing on the Trixles</h1> Randall, Tristan
  </body>
</html>

Formatted Output
Jambing on the Trixles

Randall, Tristan

See Also

Using the <STYLE> Element with Older Browsers
Using the <STYLE> Element with Older Browsers

Most of this section describes a best practice XSLT-to-HTML transformation. However, older browsers cannot process a client-side XSLT-to-HTML transformation, so this section applies to server-side or static transformations only.

Some browsers, particularly older ones, do not recognize the <STYLE> element at all. As with any unrecognized HTML elements, the default behavior of such browsers is simply to display the content of the unrecognized element—in this case, the CSS code itself.

To prevent this behavior, in a stand-alone HTML document you would embed the CSS code in an HTML comment. This masks the CSS code from browsers that do not understand the <STYLE> element, but still makes it available to browsers that do understand the <STYLE> element.

To create a comment in the result tree of an XSLT transformation, you use an <xsl:comment> element whose content is the text of the comment. For example, a backward-compatible version of the template rule for processing the <BOOK> element from book.xsl would look like bookbackwcmp.xsl, shown below.

```
Note You cannot simply place the literal <!-- and --> comment delimiters in the result tree to create the desired effect. Doing so creates a comment in the XSLT style sheet itself.
```

Example

XML File (bookbackwcmp.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="bookbackwcmp.xsl" ?>
<book>
    <author>Randall, Tristan</author>
</book>
```
XSLT File (bookbackwcmp.xsl)

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="book">
    <HTML>
        <HEAD>
            <TITLE>Book Info</TITLE>
            <STYLE>
                H1 {font-family: Arial, Univers, sans-serif; font-size: 36pt }
            </STYLE>
        </HEAD>
        <BODY><xsl:apply-templates/></BODY>
    </HTML>
</xsl:template>

<xsl:template match="book_title">
    <H1><xsl:value-of select="."/></H1>
</xsl:template>

</xsl:stylesheet>

Formatted Output

Jambing on the Trixles

Randall, Tristan
Using an External Cascading Style Sheet

Instantiating a <STYLE> element in the result tree is simple and effective for fairly small CSS rule sets. A better solution for larger blocks of CSS code is to place them in a separate, external CSS style sheet file, and then instantiate, in the result tree, an HTML <LINK> element that refers to the external file.

Assume that the CSS style sheet is in a file called book.css. The XSLT style sheet bookextcss.xsl refers to this external CSS style sheet. A template rule for processing the <book> element is shown below. The values of the rel and type attributes should always be "stylesheet" and "text/css", respectively. The value of the href attribute will be the URI of whatever style sheet is to be used.

Example

XML File (bookextcss.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="bookextcss.xsl" ?>
<book>
  <author> Randall, Tristan</author>
</book>
```

CSS File (book.css)

```css
H1

{ font-family: Arial, Univers, sans-serif;
  font-size: 36pt
}
```

XSLT File (bookextcss.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver

<xsl:template match="book">
  <HTML>
```
XSLT Processor Output

<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<TITLE>Book Info</TITLE>
</HEAD>
<BODY><H1>Jambing on the Trixles</H1> Randall, Tristan</BODY>
</HTML>

Formatted Output

Jambing on the Trixles

Randall, Tristan

See Also

Incorporating <STYLE> Elements into an XSLT File
Using Namespaces with CSS

You must declare namespaces in the document element when using XML with a cascading style sheet. Locally scoped namespaces will parse correctly, but CSS does not currently describe a mechanism for resolving namespace prefixes. Therefore, CSS rules might not apply correctly to elements in the locally scoped namespace. Placing all namespace declarations on the document element ensures that they are all global and, therefore, that there will be no collision of prefixes. The default namespace cannot be used: All namespaced elements with namespaces must have an explicit prefix. In addition, CSS selectors can specify a namespaced element only by its prefix. Therefore, you must ensure that the prefix is consistent between the XML document and the style sheet.

Here is an example of a style sheet for elements with the HTML prefix.

**CSS File (htmlprefix.css)**

```css
HTML\:IMG {
    border: 2px solid black;
}
HTML\:A:visited HTML\:IMG {
    border-color: grey;
}
HTML\:A:active HTML\:IMG {
    border-color: red;
}
HTML\:A:link HTML\:IMG {
    border-color: blue;
}
```

Notice that the "::" namespace character must be escaped by a leading backslash to differentiate it from a pseudo-class.
Using the HTML Namespace with CSS

The HTML namespace is treated specially when browsing XML with CSS. Elements from the HTML namespace are displayed as they would appear in HTML. This allows access to capabilities that are not yet provided by CSS. Some examples of useful HTML elements to embed are <TABLE>, <A>, <IMG>, <SCRIPT>, and <STYLE>.

For example, you can add a link and logo to the following restaurant review sample. First, you must declare the HTML namespace at the top of the document, and then use the HTML prefix on the embedded HTML elements. HTML embedded this way must be well-formed XML, so the <IMG> element needs a minimized end tag.

```xml
<story xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">
    ...
    <restaurant>
        <name>Red Apple Inn</name>
        <logo>
            <HTML:A href="javascript:alert('Visit the Red Apple Inn!')">
                <HTML:IMG src="red-apple.gif" height="50" width="200"/>
            </HTML:A>
        </logo>
    </restaurant>
    ...
</story>
```

In Microsoft® Internet Explorer 5.0, the prefix must remain HTML or html in order for the elements to be interpreted as HTML elements.

An <HTML:STYLE> block can be used to embed a CSS style sheet within an XML document. This block will augment any style sheets pointed to by style sheet processing instructions. When there is no external style sheet, there still must be a style sheet processing instruction present to indicate that the XML document should be displayed using the CSS style sheet language, even though no href attribute is specified.

The following example shows how the review.css style sheet can be embedded into the XML document using the HTML namespace, the <HTML:STYLE> element, and the style sheet processing instruction without an href attribute.

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/css"?>
```
XSLT Requirements for Viewing XML in a Browser

XSLT files that display an XML file in a Web browser have stricter requirements than those used for transforming XML to a general HTML file. For more information about authoring an XSLT style sheet, see Template-Driven Transformations. When authoring a style sheet for use in direct browsing, be aware of the following limitations.

- The style sheet must target HTML as its output.
- The style sheet applies to the root of the source document. Style sheets written to process against the document element will generally not display correctly. For information about the difference between the document root and the document element, see How the DOM Defines the Context for XPath Expressions.
- The XSLT style sheet must come from the same URL scheme (for example, http) and host name (for example, www.microsoft.com) as the XML source. The style sheet download follows the same security policies as those for fetching an external entity. Attempting to read a style sheet from a different domain results in an access violation error unless the security option Access data across domains is selected from the Internet Options dialog box.

Microsoft® Internet Explorer 5.0 uses a style sheet to display an XML document when that document contains a style sheet processing instruction with type "text/xsl". The following example specifies that the document should be processed with review.xsl.

```
<?xml-stylesheet type="text/xsl" href="review.xsl" ?>
```
Accessing XML and XSLT Documents from Script

The HTML that results from browsing XML with an XSLT style sheet is fully scriptable using the Dynamic HTML (DHTML) Object Model. In addition, there are two properties available on the DHTML document object that provide script access to the XML and XSLT documents.

- The `document.XMLDocument` property returns the root node of the XML source document.
- The `document.XSLDocument` property returns the root node of the XSLT style sheet document.

Modifications to these two documents through the DOM do not automatically cause updates to the resulting HTML tree, but rather offer a hook that you can use to wire up the specific updates you need.

Consider the following example, which shows how the menu is dynamically sorted in the restaurant review XSLT sample, `review.xsl`. First, add the links that trigger sorting to the style sheet.

```html
<P class="tagline">
    <A href="javascript:sort('price')">Sort menu by price</A>
    <A href="javascript:sort('description')">Sort menu by description</A>
</P>
```

Next, write the `sort` function to apply the sort to the menu data and update the display. The script accesses the DOM for the XSLT style sheet with the `XSLDocument` property, and uses DOM calls to change attributes representing sort keys.

```javascript
function sort(key) {
    // Find the "xsl:sort select" attributes in the style sheet.
    var s = document.XSLDocument.selectNodes(
```
In this example, the style sheet is modified to generate a different view of the XML. The same mechanisms can be used to change the XML source data and to generate updated views of this data.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer Guide
Serving Out XML Files from an HTTP Server

XML documents can be delivered through a standard HTTP server by registering the "text/xml" Multipurpose Internet Mail Extension (MIME) type on the server. Installing Microsoft® Internet Explorer 5.0 on the server registers the XML MIME type and associates it with the .xml and .xsl file extensions. This association is used by Microsoft Internet Information Services (IIS) and the Microsoft Personal Web Server (PWS) to determine the MIME type.

When using either of these servers on systems that do not have Internet Explorer 5.0 installed, the XML MIME type can be registered by running the register-xml.reg file. This file inserts the appropriate keys into the registry. The changes will not take effect until the computer has been restarted.

Alternatively, for IIS on systems that do not have Internet Explorer 5.0 installed, the XML MIME type can be registered using the Internet Service Manager console.

1. Select Default Web Site and open the Properties dialog box.
2. Select the HTTP Headers tab, click the File Types button, and select New Type.
3. Type .xml in the Associated extension box, and text/xml in the Content type (MIME) box. Click OK.

The .xsl extension can be registered in a similar fashion.

The .xml and .xsl extensions can also be registered to the XML MIME type manually from the File Types dialog box under Folder Options. Extensions indicating other types of XML files for purposes of browsing must be manually registered as MIME type "text/xml".
Declaring the XML MIME Type in an Active Server Page

When generating XML dynamically using Active Server Pages (ASP), the content type must be declared in the .ASP file itself. The `ContentType` property of the `Response` object indicates that the result is XML. Include the following line in Active Server Pages that are delivering XML.

```
<% Response.ContentType = "text/xml" %>
```

**Important** If the XML MIME type is not registered on the server, the MIME type sent to Internet Explorer in the HTTP header will be incorrect — often "text/plain". Although Internet Explorer 5.0 often can detect and process these documents, conflicts can arise between the various detection algorithms. For example, part of Channel Definition Format (CDF) file detection involves looking for `<CHANNEL>` tags — an XML document delivered with an incorrect MIME type and that uses elements named `<CHANNEL>` could mistakenly be handed off to the CDF processor instead of displayed as an XML document. Avoid these tricky and hard-to-debug problems by making sure your server correctly sends the XML MIME type.

Internet Explorer 5.0 also recognizes documents of MIME type "application/xml" as XML. For more information, see [Internet Engineering Task Force Request for Comments 2376, XML Media Types](https://tools.ietf.org/html/rfc2376).

**See Also**

[Deploying XSLT Solutions](https://msdn.microsoft.com/en-us/library/ms956041)
Using XSLT with the DOM or SAX

Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office makes XSLT services available from the Document Object Model (DOM). This integration of XSLT with the DOM makes XML applications simpler and more powerful. XSLT becomes a general service for locating and manipulating XML data in Web pages, server-side applications, and stand-alone applications.

XSLT functionality is accessed by two pairs of DOM methods:

- `selectNodes` and `selectSingleNode`
  These methods perform XPath queries on a DOM document.

- `transformNode` and `transformNodeToObject`
  These methods perform complete XSLT transformations on a DOM node.

The following topics describe how to use XSLT services in your application.

- Using XSLT with the DOM
- Using XSLT with SAX
- Using XSLT within ASP
- Getting Transformation Results as a DOM Document
- Performing Error Handling with XSLT
- Transforming XML Data Islands in an HTML Page
Using XSLT with the DOM

MSXML uses the DOM as the underlying tree representation of XML documents and XSLT style sheets. This enables you to implement XSLT solutions by manipulating the DOM objects in a script, or in a COM-enabled programming language. To do this, you must perform the following tasks:

- Load the source document and the XSLT file into DOM instances.
- Apply the transformation by calling either the `transformNode` method or the `transformNodeToObject` method on the source document object. Set the style sheet object as the input parameter.

The following are some of the advantages of using XSLT with the DOM.

- You can select a specified version of the XSLT run-time library with version-dependent ProgIDs.
- You can use the built-in error-detection and reporting mechanism of the DOM for exception handling in XSLT.
- You can reduce the overhead of an iterative XSLT process by compiling and caching the style sheet into an `XSLTemplate` object.

The following topics discuss the use of XSLT with the DOM. JScript examples are included.

- [Using XSLT with the DOM from an HTML Page](#)
- [Implementing Error Handling with XSLT and the DOM](#)
- [Increasing Performance by Using the XSLTemplate Object](#)
Using XSLT with the DOM from an HTML Page

The `qsalesregion.html` file (below) uses XSLT with the DOM, from a script in an HTML page. The most important part of the example code is the Jscript function, `transformIt()`. This function loads a source document, `region.xml`, and a style sheet, `region.xsl`, into two DOM instances. The XML document is loaded into `objSrcTree`, and the XSLT document is loaded into `objXSLT`. To ensure that MSXML 5.0 is used in this application, the code specifies the version dependent ProgID, `MSXML2.DOMDocument.5.0`. The XSLT transformation occurs when the `objSrcTree.transformNode(objXSLT)` method is invoked. The result document is displayed within a `<DIV id="transformedXML">` HTML tag.

The `transformIt()` function is called when the HTML page is first loaded as specified by the `<BODY onload="transformIt()">` HTML tag.

The three files used in this example — `qsalesregion.html`, `region.xml`, and `region.xsl` — are listed below.

**Example**

**HTML File (qsalesregion.html)**

```html
<HTML>
<HEAD>
<TITLE>Quarterly Sales, by Region</TITLE>
<SCRIPT language='JScript'>
<!--
function transformIt() {
  // Associate the result tree object with any element(s) whose
  // id attribute is "transformedXML."
  var objResTree = document.all['transformedXML'];
  // Declare two new MSXML DOMDocument objects.
  var objSrcTree = new ActiveXObject('MSXML2.DOMDocument.5.0');
  var objXSLT = new ActiveXObject('MSXML2.DOMDocument.5.0');
  // Load the two DOMDocuments with the XML document and the
  // XSLT style sheet.
  objSrcTree.load('region.xml');
  objXSLT.load('region.xsl');

```
// Use the transformNode method to apply the XSLT to the XML.
strResult = objSrcTree.transformNode(objXSLT);
// Assign the resulting string to the result tree object's
// innerHTML property.
objResTree.innerHTML = strResult;
return true;
}

</SCRIPT>
</BODY]
</HTML>

XML File (region.xml)

Note that this document contains no <?xml-stylesheet?> processing instruction.

<?xml version="1.0" ?>
<region year="1999">
 <name>Western Region</name>
 <quarter number="1" books_sold="24000"/>
 <quarter number="2" books_sold="38600"/>
</region>

XSLT File (region.xsl)

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <!-- Template rule for region element(s): -->
 <xsl:template match="region">
  <H1><xsl:value-of select="@year"/> Results:</H1>
  <xsl:apply-templates/>
 </xsl:template>

 <!-- Template rule for name element(s): -->
 <xsl:template match="name">
  <!-- Instantiate a level-2 heading -->
  <H2><xsl:value-of select="."/></H2>
  <!-- Instantiate a table with heading cells. -->
  <TABLE>
   <TR>
    <TH align="left">Quarter</TH>
    <TH align="right">Qty</TH>
   </TR>
   <!-- For each <quarter> sibling of the <name> element,
instantiate a table row and two cells. Data in the cell will be the values of the <quarter> element's number and books_sold attributes. -->
<xsl:for-each select="../quarter">
<TR>
<TD align="left"><xsl:value-of select="@number"/></TD>
<TD align="right"><xsl:value-of select="@books_sold"/></TD>
</TR>
</xsl:for-each>
</TABLE>
</xsl:template>
</xsl:stylesheet>

Formatted Output

1999 Results:

Western Region

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24000</td>
</tr>
<tr>
<td>2</td>
<td>38600</td>
</tr>
</tbody>
</table>

HTML Output File with Result Tree from the DOM

After the transformation, Internet Explorer would display the document with the contents of region.xml, as if its HTML code were as follows. Generated HTML is in bold.

<HTML>
<HEAD>
<TITLE>Quarterly Sales, by Region</TITLE>
</HEAD>
<BODY>
<DIV>
<H1>1999 Results</H1>
<H2>Western Region</H2>
<TABLE>
<TR>
<th align="left">Quarter</th>
<th align="right">Qty</th>
</TR>
<tr>
<td align="left">1</td>
<td align="right">24000</td>
</tr>
</TABLE>
</DIV>
</BODY>
</HTML>
See Also

Implementing Error Handling with XSLT and DOM
Implementing Error Handling with XSLT and the DOM

This topic adds error handling functionality to the script in the previous topic, [Using XSLT with the DOM from an HTML Page](#).

XSLT does not provide a native error-detecting and error-reporting mechanism for determining whether a transformation works as intended, or for handling exceptions when they occur. To implement error-handling while processing XSLT or XML, use the following DOM facilities:

- Set the `validateOnParse` property of a `DOMDocument` object. When this property is set to "false" before the object is loaded, the parser checks to see if the document is well-formed. When set to "true", the parser validates the XML document against a DTD or its XML Schema definitions.

- After an XSLT style sheet (or any other XML file) is loaded, check the `errorCode` property of the `parseError` object (that is, `xml.dom.parseError.errorCode`, where `xml.dom` is a DOM instance of the XML document). If the value of this property is "0", the document was parsed without error. Any other value indicates that an error has occurred. To examine the nature of the error, check other properties of the `parseError` object. For example, the `xml.dom.parseError.reason` property describes the cause of the error, and `xml.dom.parseError.line` indicates where the error occurred. For more information, see [IXMLDOMParseError Members](#), in XML DOM Reference.
Error Handling with the Region.xsl Example

To validate the `region.xsl` style sheet, add the following JScript fragments to the `transformIt()` function within the HTML file, `qsalesregion.html`. For more information about the files for this example, see Using XSLT with the DOM from an HTML Page.

To add the following code to the `transformIt()` function

1. Insert the following line immediately after the declaration of the `objXSLT` object variable.
   ```javascript
   objXSLT.validateOnParse = true;
   ```

2. Insert the following lines immediately after the `objXSLT.load('region.xsl')` statement.
   ```javascript
   if (objXSLT.parseError.errorCode != 0) {
       var strErrMsg = "Problem Parsing Style Sheet:<br />" + " Error #: " + objXSLT.parseError.errorCode + " Description: " + objXSLT.parseError.reason + " In file: " + objXSLT.parseError.url + " Line #: " + objXSLT.parseError.line + " Character # in line: " + objXSLT.parseError.linepos + " Character # in file: " + objXSLT.parseError.filepos + " Source line: " + objXSLT.parseError.srcText;
       objResTree.innerHTML = strErrMsg;
       return false;
   }
   ```

The resulting file is shown in XSLT HTML DOM File with Error Handling.

Similar logic for validating the XML document, `region.xml`, could be provided at analogous locations in the script.

See Also

Performing Error Handling with XSLT
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
### XSLT HTML DOM File with Error Handling

The following example adds error handling to qsalesregion.html. The lines added for error handling are in bold.

**HTML File (qsalesregionwerrchk.html)**

```html
<HTML>
<HEAD>
<TITLE>Quarterly Sales, by Region</TITLE>
<SCRIPT language='JScript'>
<!--
function transformIt() {
  // Associate the result tree object with any element(s) whose
  // id attribute is "transformedXML."
  var objResTree = document.all['transformedXML'];
  // Declare two new MSXML DOMDocument objects.
  var objSrcTree = new ActiveXObject('MSXML2.DOMDocument.5.0');
  var objXSLT = new ActiveXObject('MSXML2.DOMDocument.5.0');
  objXSLT.validateOnParse = true;
  // Load the two DOMDocuments with the XML document and the
  // XSLT style sheet.
  objSrcTree.load('region.xml');
  objXSLT.load('region.xsl');
  if (objXSLT.parseError.errorCode != 0) {
    var strErrMsg = "Problem Parsing Style Sheet:<br />
    +" Error #: " + objXSLT.parseError.errorCode + "<br /
    +" Description: " + objXSLT.parseError.reason + "<br />
    +" In file: " + objXSLT.parseError.url + "<br />
    +" Line #: " + objXSLT.parseError.line + "<br />
    +" Character # in line: " + objXSLT.parseError.linestart + "<br />
    In file: " + objXSLT.parseError.filepos + "<br />
    +" Character # in file: " + objXSLT.parseError.srcText;
    objResTree.innerHTML = strErrMsg;
    return false;
  }
  // Use the transformNode method to apply the XSLT to the XML.
  strResult = objSrcTree.transformNode(objXSLT);
  // Assign the resulting string to the result tree object's
  // innerHTML property.
  objResTree.innerHTML = strResult;
</SCRIPT>
</HEAD>
</HTML>
```
return true;
}
-->
</SCRIPT>
<BODY onload='transformIt()'>
<DIV id='transformedXML'></DIV>
</BODY>
</HTML>
Increasing Performance by Using the XSLTemplate Object

Calling the `transformNode` or `transformNodeToObject` method does not always allow for maximum performance, because the style sheet object passed into the method is a DOM node, and must be recompiled every time the method is called. Compiling the style sheet means setting up all its template rules in an executable state. If you need to call these methods repeatedly, you should cache the compiled XSLT style sheet in an `XSLTemplate` object, and use this object to perform your transformation. This helps reduce overhead, and increases the performance of your XSLT application.

This section demonstrates the use of the `XSLTemplate` object with a sample Web application for reading book reviews. For simplicity, we'll assume that a small bookstore puts all its book reviews in a single XML file, `BookReview.xml`. To read the review of particular book, a reader submits a book title from our ReviewReader application. This triggers a transformation — that is, the application applies an XSLT style sheet to the XML file, with the new book title as its input. The review of the book is the output HTML.

This example uses the following three source files: `BookReview.xml`, `BookReview.xsl`, and `BookReview.htm`. These files are listed below.

**Example: BookReview Web Application Using XSLTemplate**

**BookReview.xml**

```xml
<?xml version='1.0'?>
<book-review>
  <book>
    <title>A Good Book</title>
    <author>The Good Writer</author>
    <publisher>The Publisher</publisher>
    <date>A Good Day</date>
    <Review>
      <title>A Good Book</title> by <author>The Good Writer</author>
      published by <publisher>The Publisher</publisher> on <date>Good Day</date>, is indeed a good book.
    </Review>
  </book>
</book-review>
```
A Bad Book by The Bad Writer, published by The Publisher on A Bad Day, is indeed a bad book.

A So-so Book by The So-so Writer, published by The Publisher on A So-so Day, is indeed a so-so book.
To perform your transformations with XSLTemplate

As shown above, to use the XSLTemplate object to cache a compiled XSLT style sheet and perform transformations, your must perform the following tasks.

1. Load the appropriate XSLT style sheet as a FreeThreadedDOMDocument object. Be sure to set the async property of this object to "false".
2. Create an XSLTemplate object. Set its stylesheet property to the XSLT style sheet created in the Step 1. Notice that loading and caching are carried out in the init() function when the HTML page is first loaded.
3. Load the source XML document as a DOMDocument object. In this example,
this XML document is loaded once in the init() function as well. In other applications, this is not necessarily the case.

4. Create an XSLProcessor object for each transformation. The following steps are needed to complete the transformation.

5. Assign the source XML document object to the input property of the XSLProcessor object created in Step 3.

6. Call the addParameter method on the XSLProcessor object to pass any required global variables used in the XSLT style sheet. In the example above, the style sheet uses a title parameter to select a book with the specified title: `<xsl:apply-templates select="//book[title=$title]" />`.

7. Optionally, assign an output stream object (such as a recordset object in ADO or a Response object in ASP) to the output parameter of the XSLProcessor object. Choose this option if you do not want to use the default output stream object.

8. Call the transform method on the XSLProcessor object to start the transformation.

9. Retrieve the result of the transformation from the output parameter of the XSLProcessor object.

In addition to passing a parameter to an XSLT style sheet, as described in Step 6, you can also pass an object (a DOM node), if necessary. For more information about how to pass an object to an XSLT style sheet by way of an XSLProcessor object, see the XSLProcessor.addObject method.
Using XSLT with SAX

With MSXML versions 4.0 and later, you can also use the output property of the IXSLProcessor interface to accept objects that implement SAX interfaces. This enables you to fire SAX events from XSLT transformations, and to catch those events in your SAX-based application.

Example: Using XSLT with SAX from Script

This example shows how to use the SAX writer from within a Web script to accept and write the output of an XSLT processor transformation. The transformation is performed on the Sample XML File (books.xml).

To complete this example, you must create and use the following two additional files:

- **Books.html**, an HTML document file used to host and run the sample code. This code will be used in Internet Explorer, inline as client-side script.
- **Transform1.xsl**, an XSLT style sheet. This style sheet will be used to transform the contents of the sample file (books.xml), and to present it in the browser as HTML.

Try It! To run this example, copy the HTML and XSLT samples below, and save them under the specified file names. Save the books.xml file in the same folder. Open books.html to view the output.

**HTML File (books.html)**

```html
<HTML>
<HEAD>
<TITLE>Test for XSLT output to SAX XML writer application</TITLE>
</HEAD>
<BODY>
<DIV ID="result">Document is loading...</DIV>
<SCRIPT>

    //Create the XSLTemplate object (xslt).
    var xslt = new ActiveXObject("Msxml2.XSLTemplate.5.0");

```

//Create and load the stylesheet (transform1.xsl) as a DOMDocument.
var xslDoc = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
var xslProc;
xslDoc.async = false;
xslDoc.load("transform1.xsl");

//Connect the XSLTemplate object to stylesheet DOMDocument.
xsl.stylesheet = xslDoc;

//Create XSLT processor using stylesheet for XSL template.
xslProc = xslt.createProcessor();

//Create and load sample XML file (books.xml) as a DOMDocument.
var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmlDoc.async = false;
xmlDoc.load("books.xml");

//Create SAX writer.
var xmlWriter = new ActiveXObject("Msxml2.MXXMWriter.5.0");

//Assign XML sample file as input of the transform() method.
xslProc.input = xmlDoc;

//Use a SAX writer as the output of the transform() method.
xslProc.output = xmlWriter;

//Do transformation on the sample XML file.
xslProc.transform();

//Use SAX writer output to generate inner HTML for page.
result.innerHTML = xmlWriter.output;

</SCRIPT>
</BODY>
</HTML>

XML stylesheet File (transform1.xsl)

<?xml version='1.0'?
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="catalog">
<HTML>
 <HEAD>
  <TITLE>Books catalog - listing for Books.xml</TITLE>
 </HEAD>
 <BODY><FONT FACE="Verdana">
 <H3>My Books Catalog</H3>

<table>
<thead>
<tr>
<th>ID:</th>
<th>Author:</th>
<th>Title:</th>
<th>Genre:</th>
<th>Price:</th>
<th>Published on:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also

[Increasing Performance by Using the XSLTemplate Object](#) | [output Property (IXSLProcessor)](#) | [JumpStart for Creating a SAX2 Application](#)
Using XSLT within ASP

You can perform XSLT on the server to filter or customize XML, or to change the XML Schema. You can also deploy content in XML, and transform it to HTML on demand for clients.

The code for performing XSLT within an Active Server Pages (ASP) file is similar to that used on the client. For more information, see Transforming XML Data Islands in an HTML Page.

First, load the XML source document and XSLT file. Then, call transformNode to link to the XSLT processor.

```<%@ LANGUAGE = JScript %> <%
    // Set the source and style sheet locations here.
    var sourceFile = Server.MapPath("simple.xml");
    var styleFile = Server.MapPath("simple.xsl");

    // Load the XML.
    var source = Server.CreateObject("Msxml2.DOMDocument.5.0");
    source.async = false;
    source.load(sourceFile);

    // Load the XSLT.
    var style = Server.CreateObject("Msxml2.DOMDocument.5.0");
    style.async = false;
    style.load(styleFile);
    Response.Write(source.transformNode(style));
%>```

The Server.MapPath method resolves a relative URL to a full path. The Server.CreateObject method instantiates a new XML DOM Document object. The results of the transformation are sent to the client by the Response.Write method.

The transformNodeToObject method allows you to write a transformation directly to the IStream interface of the Response object. To use this feature, change the last line in the preceding example to the following:

```source.transformNodeToObject(style, Response);```
In most situations, this method performs better on the server than the `transformNode` method. This is especially true for long documents, which require a significant memory allocation to hold the complete transformation results.

For an example of a `transformNodeToObject` call, see the `simple.asp` file in XSLT Sample: Breakfast Menu.
Getting Transformation Results as a DOM Document

The `transformNodeToObject` method provides a way to generate a new DOM document instead of XML text. The following code snippet uses data islands to load the XML source document and the style sheet. An empty data island is created to hold the results of the transformation.

```html
<HTML>
  <SCRIPT FOR="window" EVENT="onload">
    data.transformNodeToObject(ss.XMLDocument, resultTree.XMLDocument);
  </SCRIPT>

  <XML id="data" src="portfolio.xml"></XML>
  <XML id="ss" src="portfolio.xsl"></XML>
  <XML id="resultTree"></XML>
</HTML>
```

Like the `transformNode` method, the `transformNodeToObject` method is applied to a node in the source tree, and accepts a style sheet in the form of a DOM tree. A second parameter contains another DOM Document node that will hold the result tree of the transformation.

Properties, that are set on the result tree object, such as `validateOnParse` and `async`, will apply during the transformation.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Performing Error Handling with XSLT

An XSLT file can generate two types of errors: parse errors and run-time errors. There are ways to check for these errors, and to report them.

The following is a simple HTML file that uses XML data islands to load an XML file and an XSLT style sheet, and to display the results.

```html
<HTML>
    <XML id="source" src="simple.xml"></XML>
    <XML id="style" src="simple.xsl"></XML>
    <SCRIPT event="onload" for="window">
        showResult.innerHTML = source.transformNode(style.XMLDocument);
    </SCRIPT>
    <BODY>
        <DIV id="showResult"></DIV>
    </BODY>
</HTML>
```

The XML data islands load the XML data from the `simple.xml` file and the XSLT style sheet from the `simple.xsl` file, respectively. The call to the `transformNode` method invokes the XSLT processor, and the resulting HTML is inserted into the page.

If nothing appears, first check the `parseError` object to see if there was a parsing error in the XML file or style sheet. Parsing errors on data islands do not generate error messages in the browser. If an error occurred, you can retrieve the cause or put an error message into the page.

The following example checks for parsing errors. If errors occur, a function is called to format the error information in a way that is appropriate for insertion into the HTML document.

```html
<HTML>
    <HEAD>
        <TITLE>Simple Error Message Demo</TITLE>
    </HEAD>
```
In addition to parsing errors, the preceding example also detects XSLT run-time errors that occur inside calls to the `transformNode` method. Run-time errors are normally reported to the user through a Microsoft® Internet Explorer 5.0 or later error dialog box. However, it is often useful to suppress this dialog box. The `try` and `catch` functions in Microsoft JScript® (compatible with the ECMA 262 language specification) allow the error to be suppressed and handled consistently with parsing errors.

The actual formatting of the error information in the previous example is performed by two functions: `reportParseError` and `reportRuntimeError`. These functions are located in a separate script file, `reportErrors.js`. The main HTML page references this script file.

**JScript File (reportErrors.js)**

```javascript
// Parse error formatting function
function reportParseError(error)
```
These two functions return HTML fragments that can be inserted into the HTML page in place of the correct transformation result. This alerts you to a problem.

Reporting an error using this mechanism is also useful in an Active Server Page (ASP). Here is the equivalent code for processing an XML file on the server.

```html
<%@ LANGUAGE = JScript %>
<%  
  // Parse error formatting function.  
  function reportParseError(error)  
  {  
    var s = "";  
    for (var i=1; i<error.linepos; i++) {  
      s += " ";  
    }  
    r = "<font face=Verdana size=2><font size=4>XML Error loading " + 
          error.url + "</font>" +  
      "<P><B>" + error.reason +  
      "</B></P></font>";  
    if (error.line > 0)  
      r += "<font size=3><XMP>" +  
      "at line " + error.line + ", character " + error.linepos +  
      "\n" + error.srcText +  
      "\n" + s + "^" +  
      "</XMP></font>";  
    return r;  
  }  

  // Runtime error formatting function.  
  function reportRuntimeError(exception)  
  {  
    return "<font face=Verdana size=2><font size=4>XSL Runtime Error</font>" +  
      "<P><B>" + exception.description + "</B></P></font>";  
  }

  // Runtime error formatting function.  
  function reportRuntimeError(exception)  
  {  
    return "<font face=Verdana size=2><font size=4>XSL Runtime Error</font>" +  
      "<P><B>" + exception.description + "</B></P></font>";  
  }
```

"</XMP></font>";
    return r;
}
// Run-time error formatting function.
function reportRuntimeError(exception)
{
    return "<font face=Verdana size=2><font size=4>XSL Runtime Error
    "<P><B>" + exception.description + "</B></P></font>";
}
// Set the source and style sheet locations here.
var sourceFile = Server.MapPath("simple.xml");
var styleFile = Server.MapPath("simple.xsl");

// Load the XML.
var source = Server.CreateObject("Msxml2.DOMDocument.5.0");
source.async = false;
source.load(sourceFile);
// Load the XSL.
var style = Server.CreateObject("Msxml2.DOMDocument.5.0");
style.async = false;
style.load(styleFile);
if (source.parseError.errorCode != 0)
    result = reportParseError(source.parseError);
else
    {
        if (style.parseError.errorCode != 0)
            result = reportParseError(style.parseError);
        else
            {
                try {
                    result = source.transformNode(style);
                }
                catch (exception) {
                    result = reportRuntimeError(exception);
                }
            } 
    }
Response.Write(result);
%>

The code here is essentially the same as that for the client side, with the same error formatting functions and processing. The Server.CreateObject and Response.Write methods are used in place of the client-side functions provided by data islands and the innerHTML method.

See Also
Implementing Error Handling with XSLT and DOM
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developers Guide
Transforming XML Data Islands in an HTML Page

When building an XSLT solution that uses Internet Explorer, you might want to use data islands as a convenient way to access XML data from an HTML page. Because XSLT is itself an XML grammar, data islands can be used to load style sheets as well. The `simple-islands.htm` example uses two data islands to load both the source XML data and the XSLT style sheet. The `transformNode` method is used on these two data islands to generate HTML code that can then be inserted into the page.

When the `simple-islands.htm` page, including the data islands, is completely loaded, the `onload` event fires and the `transformNode` method is executed on the XML source data. The result of this call is XML text, which in this example is well-formed HTML text. You can feed this text directly into the `<xslTarget>` element as in:

```javascript
xslTarget.innerHTML = source.transformNode(style.XMLDocument)
```

The object returned from a data island is a DHTML DOM object (in this case it is the DHTML object associated with the `<xml>` element), not an XML DOM object. The `style.XMLDocument` property ensures that a style object is passed to the `transformNode` method as an XML DOM object.

There is no need to write

```javascript
source.XMLDocument.transformNode(style.XMLDocument),
```

because `transformNode` is an XML DOM method, not a DHTML method. When an XML DOM method is called on a DHTML object corresponding to the XML data island element, the script engine automatically resolves it to the XML DOM Document node. In general, the `XMLDocument` property is not required if another DOM property or method is being applied.

You can also write

```javascript
source.XMLDocument.transformNode(style.documentElement)
```
and apply the style sheet to the document element instead of the root. However, the style sheet must reflect this. For example, the document element must be handled by a rule other than the root template `xsl:template match="/"`, which will not be
invoked unless the style sheet is executed against the document root. It is recommended that you process your style sheets off the document root so that they will also work for directly browsing XML. For more information about the differences between the document root and the document element, see How the DOM Defines the Context for XPath Expressions.

For more information, see Deploying XSLT in Internet Explorer.

Example

For an example of an HTML page with XML Data Islands, see simple-islands.htm in XSLT Sample: Breakfast Menu.

See Also

XML Data Islands | Performing Error Handling with XSLT | Getting Transformation Results as a DOM Document
XSLT Sample: Breakfast Menu

The Breakfast Menu sample demonstrates XML data islands and includes the following files.

- simple-islands.htm
- simple.xml
- simple.xsl
- simple.asp

To test the sample, copy the files to a folder on your local drive. Then do one of the following (beginners might want to try all three methods):

- Open simple.xml using Internet Explorer by double clicking the file from Explorer.
- Open simple-islands.htm using Internet Explorer by double clicking the file from Explorer.
- Open simple.asp page by using Internet Explorer with a URL such as "http://localhost/breakfast/simple.asp".

The last option assumes that IIS is running on your machine and that you have created a breakfast virtual directory and placed all the files in it.

HTML File (simple-islands.htm)

```html
<HTML>
  <HEAD>
    <TITLE>Simple demo of Microsoft XSL Processor</TITLE>
  </HEAD>

  <XML id="source" src="simple.xml"></XML>
  <XML id="style" src="simple.xsl"></XML>

  <SCRIPT FOR="window" EVENT="onload">
    xslTarget.innerHTML = source.transformNode(style.XMLDocument);
  </SCRIPT>

  <BODY>
    <P STYLE="font-size:10pt; font-family:Verdana; color:gray">
      This demo shows the use of data islands for
```
loading XML source and XSLT style sheets, and for inserting the transformed result into the Web page.

</DIV></DIV></BODY></HTML>

XML File (simple.xml)

<?xml version='1.0'?>
<?xml:stylesheet type="text/xsl" href="simple.xsl" ?>
<breakfast-menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup.</description>
    <calories>650</calories>
  </food>
  
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream.</description>
    <calories>900</calories>
  </food>
  
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream.</description>
    <calories>900</calories>
  </food>
  
  <food>
    <name>French Toast</name>
    <price>$4.50</price>
    <description>Thick slices made from our homemade sourdough bread.</description>
    <calories>600</calories>
  </food>
  
  <food>
    <name>Homestyle Breakfast</name>
    <price>$6.95</price>
    <description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns.</description>
    <calories>950</calories>
  </food>
</breakfast-menu>
XSLT File (simple.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <HTML>
            <BODY STYLE="font-family:Arial, helvetica, sans-serif; font-size:12pt; background-color:#EEEEEE">
                <xsl:for-each select="breakfast-menu/food">
                    <DIV STYLE="background-color:teal; color:white; padding:4px">
                        <SPAN STYLE="font-weight:bold; color:white"><xsl:value-of select="name"/></SPAN>-
                        <xsl:value-of select="price"/>
                    </DIV>
                    <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt">
                        <xsl:value-of select="description"/>
                        <SPAN STYLE="font-style:italic">
                            (<xsl:value-of select="calories"/> calories per serving)
                        </SPAN>
                    </DIV>
                </xsl:for-each>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```

ASP File (simple.asp)

```asp
<%@ LANGUAGE = JScript %>
<%
    // Set the source and style sheet locations here.
    var sourceFile = Server.MapPath("simple.xml");
    var styleFile = Server.MapPath("simple.xsl");

    // Load the XML.
    var source = Server.CreateObject("Msxml2.DOMDocument.5.0");
    source.async = false;
    source.load(sourceFile);

    // Load the XSLT.
    var style = Server.CreateObject("Msxml2.DOMDocument.5.0");
    style.async = false;
    style.load(styleFile);
    Response.Write(source.transformNode(style));
%>
```
Using XPath Expressions to Select Nodes

XML Path Language (XPath) is a language used to create expressions that can address parts of an XML document. XPath expressions can also manipulate strings, numbers, and Booleans, and can select a set of nodes in the document. XPath is used by both XSLT and XML Pointer Language (XPointer), and models an XML document as a tree of nodes of different types, including element, attribute, and text. XPath expressions identify these nodes in the XML document based on their types, names, and values, and on the relationship of a node to other nodes in the document. An XPath expression yields one of the following basic objects.

- node set
- Boolean
- number
- string

For example, suppose you want to find `<author>` elements that have a period attribute with the value "classical", and that are contained in the `<authors>` element at the document root. This query can be expressed with the following XPath expression.

```
/authors/author[@period='classical']
```

These expressions are a core part of XSLT transformations, which associate the expression results with templates to create a new XML document. Expressions using the XPath syntax can also be performed directly against the XML Document Object Model (DOM).

An important type of XPath expression is a location path, which is an expression that selects a set of nodes relative to the context node. The result of evaluating a location path is a node-set containing the selected nodes.

For more information about XPath, see the [XPath Developer's Guide](#).
See Also

XPath Syntax | Template-Driven Transformations | Defining Match Patterns in <xsl:template>

Other Resources

XML Path Language (XPath) Version 1.0
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Development Guide
Using XSLT Variables and Parameters

XSLT Variables

In XSLT, you can use `<xsl:variable>` to define an XSLT variable. A variable in XSLT functions like a named constant in traditional programming languages, such as C/C++. You can use a variable to store values that might be needed repeatedly. A variable is also useful for caching context-sensitive values or a temporary tree during the transformation.

Once defined, an XSLT variable cannot be changed until it falls out of its scope. To reference a defined variable, prefix the $ sign to the value of the `name` attribute of the `<xsl:variable>` element.

An XSLT variable is global if it is declared as an immediate child element of the `<xsl:stylesheet>` element. A global variable can be used anywhere in the stylesheet. A variable is local if it is declared within a template rule. The scope of a local variable is only within the context in which it is defined.
**XSLT Parameters**

You can use `<xsl:param>` to define an XSLT parameter. A parameter in XSLT functions like a variable in traditional programming languages, such as C/C++.

To reference a defined parameter, prefix the $ sign to the value of the `name` attribute of the `<xsl:param>` element.

An XSLT parameter is global if it is declared as an immediate child element of the `<xsl:stylesheet>` element. A global parameter can be used anywhere in the style sheet. A parameter is local if it is declared within a template rule. The scope of a local parameter is only within the context in which it is defined.

An XSLT parameter is a parameterized XSLT variable. Once defined, it cannot be changed until it falls out of its scope. However, a parameter can take a value passed in from outside its scope. For example, a global parameter can take a value passed in from a script in an HTML or ASP page. A local parameter can take a value passed in from the calling template rule.

The topics in this section refer to the **weatherlog** example, which will be added throughout the section to illustrate the use of variables and parameters. The initial sample files appear at the beginning of the section. The finished sample files are provided at the end.

This section contains the following topics.

- [Initial XSLT Example Using Variables](#)
- [Using Variables to Store Values that Do Not Change](#)
- [Using Parameters to Store Values that Change](#)
- [Finished XSLT Example Using Variables](#)
Initial XSLT Example Using Variables

The following example will be referred to, and added to, throughout this section. The Finished XSLT Example Using Variables appears at the end of the section.

Example

XML File (weatherlog.xml)

The following XML file contains weather information for various locations, over a period of time. The locations are initially assigned some generic names ("location1", "location2", etc.). Later, we will use XSLT variables to replace these names with more specific locations, such as "Airport" or "Midtown".

```xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="weatherlogv1.xsl"?>
<weather temp_scale="F">
     <day date="2000-11-01">
         <locale place="location1">
             <temp>
                 <low>47</low>
                 <high>62</high>
             </temp>
             <wind_dir>NNE</wind_dir>
             <wind_speed units="mph">5</wind_speed>
             <barom units="mb">1013.2</barom>
             <rel_humid>.74</rel_humid>
             <precip units="in">0</precip>
         </locale>
         <locale place="location2">
             <temp>
                 <low>50</low>
                 <high>64</high>
             </temp>
             <wind_dir>NNE</wind_dir>
             <wind_speed units="mph">4</wind_speed>
             <barom units="mb">1015.7</barom>
             <rel_humid>.70</rel_humid>
             <precip units="in">0</precip>
         </locale>
     </day>
</weather>
```
XSLT File (weatherlogv1.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">
    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Weather Readings</TITLE>
            </HEAD>
            <locale place="location3">
                <temp>
                    <low>48</low>
                    <high>61</high>
                </temp>
                <wind_dir>NNE</wind_dir>
                <wind_speed units="mph">8</wind_speed>
                <barom units="mb">1010.0</barom>
                <rel_humid>.68</rel_humid>
                <precip units="in">.1</precip>
            </locale>
            <day date="2000-11-02">
                <locale place="location1">
                    <temp>
                        <low>43</low>
                        <high>58</high>
                    </temp>
                    <wind_dir>NE</wind_dir>
                    <wind_speed units="mph">1</wind_speed>
                    <barom units="mb">1009.1</barom>
                    <rel_humid>.80</rel_humid>
                    <precip units="in">2.3</precip>
                </locale>
                <locale place="location3">
                    <temp>
                        <low>40</low>
                        <high>61</high>
                    </temp>
                    <wind_dir>N</wind_dir>
                    <wind_speed units="mph">12</wind_speed>
                    <barom units="mb">1011.0</barom>
                    <rel_humid>.73</rel_humid>
                    <precip units="in">2.0</precip>
                </locale>
            </day>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
Formatted Output

The resulting table shows the date and the locations where readings were taken.

Weather Readings

As of 2000-11-01

location1

location2

location3

As of 2000-11-02

location1

location3

Processor Output

Line breaks have been added for clarity.
See Also

Finished XSLT Example Using Variables
Using Variables to Store Values that Do Not Change

In XSLT, variables are the same as constants in programming and scripting languages. Variables identify generically named objects, from which a specific value can be retrieved.

Unlike programming and scripting constants, the value of an XSLT variable can be set at run time by some condition in the XML source document or in the style sheet. Because these conditions can change from one transformation to the next, these objects are known as variables rather than constants.

The `<xsl:variable>` element is used to set the value of a variable. The following topics use the `weatherlog.xml` example to illustrate utilities of XSLT variables.

- Using Variables as Convenience Constants
- Using Variables to Cache XPath Expressions

See Also

Initial XSLT Example Using Variables
Using Variables as Convenience Constants

Suppose we want to replace the generic location names in the XSLT style sheet in Initial XSLT Example Using Variables (weatherlogv1.xsl) with more descriptive names, so that "location1", "location2", and "location3" are mapped to "Midtown", "Northeast", and "Airport", respectively, for the place attribute of the locale elements. We can do this with the help of an XSLT variable, placename. The changes affect the template rule for the weather element. The revised template rule is shown below. Changes are in bold.

```xml
<xsl:template match="weather">
  <H1>Weather Readings</H1>
  <xsl:for-each select="day">
    <H2>As of <xsl:value-of select="@date"/></H2>
    <xsl:for-each select="locale">
      <xsl:variable name="placename">
        <xsl:choose>
          <xsl:when test="@place='location1'">Midtown</xsl:when>
          <xsl:when test="@place='location2'">Northeast</xsl:when>
          <xsl:when test="@place='location3'">Airport</xsl:when>
          <xsl:otherwise>[Unknown Locale]</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <H3><xsl:value-of select="$placename"/></H3>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

For each <locale> element, we have declared a placename variable, and assigned it a value based on the value of its place attribute. This variable value ($placename) is then used as the output of the new location name.

**Formatted Output**

With the revised the XSLT style sheet, the result is as follows:
Weather Readings

As of 2000-11-01

Midtown
Northeast
Airport

As of 2000-11-02

Midtown
Airport

The variable declaration above can be converted to a named template, and placed outside the template rule. This makes its value accessible anywhere in the style sheet. For more information, see Using Parameters to Store Values that Change.

See Also

Initial XSLT Example Using Variables | Finished XSLT Example Using Variables
Using Variables to Cache XPath Expressions

Because XPath expressions can be long, writing them manually can be cumbersome and prone to typographical errors, especially when you plan to use them multiple times. To avoid this problem, you can write an XPath expression once, and cache it in an XSLT variable.

Suppose we want to display the average high temperature for each day listed in the.weatherlog.xml file in Initial XSLT Example Using Variables. The following XPath expression will do just that.

format-number(sum(locale/temp/high) div count(locale), '#0.00')

We could then use this expression in an XSLT element directly, as follows:

<xsl:value-of select="format-number(sum(locale/temp/high) div count(locale), '#0.00')" />

But if we wanted to use this expression in other parts of the XSLT style sheet, we would have to type this rather complicated expression again. A more efficient way is to cache this expression in an XSLT variable:

<xsl:variable name="average_temp"
    select="format-number(
        sum(locale/temp/high) div count(locale),
        '#0.00')"
/> 

Then call the expression by referencing the variable:

<xsl:value-of select="$average_temp"/>

The following is a revised XSLT style sheet:

<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns="http://www.w3.org/TR/REC-html40">
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>Weather Readings</TITLE>
      </HEAD>
      <BODY>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </xsl:template>

  <!-- Override built-in template rule for text nodes. -->
  <xsl:template match="text()"/>

  <xsl:template match="weather">
    <H1>Weather Readings</H1>
    <xsl:for-each select="day">
      <xsl:variable name="average_temp" select="format-number(
        sum(locale/temp/high)
        div count(locale),
        '#0.00')"/>
      <H2>As of <xsl:value-of select="@date"/></H2>
      <P>Average Temperature:
        <xsl:value-of select="$average_temp"/>°F</P>
    </xsl:for-each>
    <xsl:for-each select="locale">
      <xsl:variable name="placename">
        <xsl:choose>
          <xsl:when test="@place='location1'">Midtown</xsl:when>
          <xsl:when test="@place='location2'">Northeast</xsl:when>
          <xsl:when test="@place='location3'">Airport</xsl:when>
          <xsl:otherwise>[Unknown Locale]</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <H3><xsl:value-of select="$placename"/></H3>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>

Note  The above template includes a reference to an entity, °F, which displays in Microsoft Internet Explorer as the degrees symbol, "°".

For more information, see the Finished XSLT Example Using Variables.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Using Parameters to Store Values that Change

The use of XSLT parameters is similar to that of XSLT variables. The main difference is that the value of a parameter can change during the run of the style sheet, or be passed in from an outside environment. The value of a variable, once set, cannot be changed. The following table describes two kinds of parameters you can use in style sheets.

<table>
<thead>
<tr>
<th>Parameter type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local parameters</td>
<td>Used for passing data from one template rule to another. For example, you can pass a parameter to a named template, based on some condition in a given template rule. This enables you to control the way the named template behaves.</td>
</tr>
<tr>
<td>Global parameters</td>
<td>Used for passing values into a style sheet. This enables the style sheet to perform particular steps, depending on a value supplied by the user. In Internet Explorer, parameters are passed to a style sheet using script.</td>
</tr>
</tbody>
</table>

This section contains the following topics.

- [Passing Data between Template Rules Using Local Parameters](#)
- [Passing Values into XSLT Using Global Parameters](#)

See Also

[Initial XSLT Example Using Variables](#) | [Finished XSLT Example Using Variables](#)
Passing Data between Template Rules Using Local Parameters

To pass a parameter from one template rule to another, use the `<xsl:with-param>` element as a child of an `<xsl:apply-templates>` or `<xsl:call-template>` element. For example:

```xml
<xsl:call-template name="some_named_template">
    <xsl:with-param name="param1" select="@some_attrib"/>
</xsl:call-template>
```

When the XSLT processor encounters the `<xsl:call-template>` element in a template rule, the template named `some_named_template` is instantiated. If this template uses the parameter `$param1`, a new value (the `some_attrib` attribute value of the current context node) is passed to that named template.

In the previous example of using XSLT variables, the definition of a variable had to be placed inside the template rule that used it. Because the scope of the variable is limited, its value is inaccessible to other template rules, which might make use of it. To make the value accessible, you can use a named template. The value of the `place` attribute you want to translate can then be passed into this template.

This approach differs from the variable-based approach in `Using Variables as Convenience Constants`, in the following ways.

- The `<xsl:choose>` block is now enclosed in a named template rule. The name of this rule is `placename`.
- There is an `<xsl:param>` element. This element assigns a string value, "[Unknown Locale]", to the `$location` parameter as a default. This default is used if a call to this template fails to provide a value.
- Instead of referencing the `@place` attribute, you can reference the `$location` parameter. If the call to the `placename` template rule supplies a value for the `$location` parameter (by means of an `<xsl:with-param>` element), the `<xsl:choose>` block uses that value.
- The call to the `$placename` variable becomes a call to a named template,
using the <xsl:with-param> element to pass a parameter.

**Note** The value of the place attribute is assigned to the $location variable in the <xsl:with-param> element. If that attribute is empty, or has any value other than "location1", "location2", or "location3", the value returned from the named template is "[Unknown Locale]".

**Example**

This example demonstrates passing data between template rules using local parameters.

**XML File**

Use `weatherlog.xml` in *Initial XSLT Example Using Variables*. Make sure that the `href` attribute references the `weatherlogv1.xsl` style sheet.

**XSLT File**

The following file is an adaptation of `weatherlogv1.xsl` file in *Initial XSLT Example Using Variables*.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/REC-html40">

<xsl:template match="/">
   <HTML>
      <HEAD>
         <TITLE>Weather Readings</TITLE>
      </HEAD>
      <BODY>
      <xsl:apply-templates/>
      </BODY>
   </HTML>
</xsl:template>

<!-- Override built-in template rule for text nodes. -->
<xsl:template match="text()"/>

<xsl:template name="placename">
   <xsl:param name="location" select="'[Unknown Locale]'"/>
   <xsl:choose>
```
Weather Readings

As of 2000-11-01
Midtown
Northeast
Airport

As of 2000-11-02
Midtown
Airport

See Also
Finished XSLT Example Using Variables
Passing Values into XSLT Using Global Parameters

To pass a value into an XSLT style sheet at run time, you must perform the following steps.

1. Declare a global XSLT parameter in the style sheet.
2. Call the addParameter method on the XSLProcessor object, from a script, to assign a value to the parameter.

Step 1 is straightforward: Declare `<xsl:param>` as an immediate child element of the `<xsl:stylesheet>` element. For example:

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="bgColor">black</xsl:param>
  <xsl:template match="/">
    bgColor = <xsl:value-of select="$bgColor"/>
  </xsl:template>
</xsl:stylesheet>
```

The example above defines a global parameter, `bgColor`. The value of this parameter will be displayed as a string.

Step 2 is more complicated. We need to create an XSLTemplate instance to cache the compiled style sheet, and an XSLProcessor object that will be used to pass a value (red) to the global parameter (`bgColor`) declared in the style sheet. The following is an example of Step 2.

```html
<HTML>
<HEAD>
  <TITLE>bgColor in HTML</TITLE>
```
```javascript
var xslt, xml, xslTemp, xslProc;
function load()
{
    xml = new ActiveXObject("Msxml2.DOMDocument.5.0");
    xml.async=false;
    xml.load("bgColor.xsl");

    xslt = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
    xslt.async=false;
    xslt.load("bgColor.xsl");

    xslTemp = new ActiveXObject("Msxml2.XSLTemplate.5.0");
    xslTemp.stylesheet = xslt;

    xslProc = xslTemp.createProcessor();
    xslProc.input = xml;
    xslProc.addParameter("bgColor", "red");

    try {
        xslProc.transform;
        result.innerHTML = xslProc.output;
    } catch(e) {
        result.innerHTML = e.description;
    }
}
</SCRIPT>

<BODY ONLOAD="load()">
    <DIV ID="result"></DIV>
</BODY>
</HTML>

See Also

Increasing Performance by Using the XSLTemplate Object | addParameter Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Finished XSLT Example Using Variables

The following XSLT file is the result of creating the template rules described in the previous topics in this section.

Example

This example uses variables and passes parameters between template rules.

XML File (weatherlog.xml)

Use weatherlog.xml, in Initial XSLT Example Using Variables. Change the href attribute to reference weatherlog.xsl.

XSLT File (weatherlog.xsl)

```xml
<?xml version="1.0" ?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY degrees "&#176;">
]><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/REC-html40">
  <!-- Global parameter for selecting a local. -->
  <xsl:param name="place_param" select="'[All]'"/>

  <!-- Template rule for document root: basic Web page setup. -->
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>Weather Readings</TITLE>
        <STYLE type="text/css">
          h1 {background-color: gold;
           font-family: Tahoma,Verdana,Arial,sans-serif;
           font-size: 24pt}
          h2 {background-color: yellow;
           font-family: Tahoma,Verdana,Arial,sans-serif;
           font-size: 22pt}
        </STYLE>
      </HEAD>
      <BODY>
        <P>Weather Readings...</P>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```
Override built-in template rule for text nodes
(suppress any text unless explicitly requested).

Named template rule "placename" returns informative name of
a locale, given known generic values of "location1," and so on.

Template rule for root <weather> element.

Set up major page heading, depending on value of global
parameter "place_param".
<!-- Process <day> children of <weather> element. -->
<xsl:for-each select="day">
  <!-- Variable "average_temp" holds formatted version of average high temperature, across all locales for a given day. -->
  <xsl:variable name="average_temp"
    select="format-number(sum(locale/temp/high)/count(locale), '##0.00')"/>
  <h2>As of <xsl:value-of select="@date"/></h2>
  <xsl:choose>
    <!-- If global parameter "place_param" is the default "[All]", process all <locale> children of the <day> element. -->
    <xsl:when test="$place_param = '[All]'">
      <xsl:apply-templates select="locale"/>
    </xsl:when>
    <!-- ...otherwise, filter the <locale> children of the <day> element, selecting only those matching the requested locale. -->
    <xsl:otherwise>
      <xsl:apply-templates select="locale[@place=$place_param"]"/>
    </xsl:otherwise>
  </xsl:choose>
  <!-- If processing all locales (per "place_param" global parameter), add a block displaying the average high temperature across all locales. -->
  <xsl:if test="$place_param = '[All]'">
    <div class="locale">
      <table align="center" width="400">
        <tr>
          <td>Average High Temperature for <xsl:value-of select="@date"/></td>
          <td><xsl:value-of select="$average_temp"/><degrees; <xsl:text> </xsl:text><xsl:value-of select="../@temp_scale"/></td>
        </tr>
      </table>
    </div>
  </xsl:if>
</xsl:for-each>

<!-- Template rule for <locale> elements: set up a label for the locale (using "informative" form of location name returned by the "placename" named template, followed by a table of this locale's weather readings. -->
<xsl:template match="locale">
  <div class="locale">
    <!-- Template rule for <locale> elements: set up a label for the locale (using "informative" form of location name returned by the "placename" named template, followed by a table of this locale's weather readings. -->
  </div>
<h3><xsl:call-template name="placename">
    <xsl:with-param name="location" select="@place"/>
</xsl:call-template></h3>
<table align="center">
    <tr>
        <th>Temp Range</th>
        <th>Wind</th>
        <th>Barometer</th>
        <th>Humidity</th>
        <th>Precipitation</th>
    </tr>
    <tr>
        <td align="center">
            <xsl:value-of select="concat(temp/low, '-', temp/high)"/>
            &deg; <xsl:text>
            </xsl:text>
            <xsl:value-of select="/weather/@temp_scale"/>
        </td>
        <td align="center">
            <xsl:value-of select="concat(wind_speed, ' ', wind_speed/@units)"/>
            <xsl:value-of select="concat(' from ', wind_dir)"/>
        </td>
        <td align="center">
            <xsl:value-of select="concat(barom, ' ', barom/@units)"/>
        </td>
        <td align="right">
            <xsl:value-of select="concat(string(rel_humid * 100), '%')"/>
        </td>
        <td align="right">
            <xsl:value-of select="concat(precip, ' ', precip/@units)"/>
        </td>
    </tr>
</table>

Formatted Output
Weather Readings for All Available Locations

As of 2000-11-01

<table>
<thead>
<tr>
<th>Location</th>
<th>Temp Range</th>
<th>Wind</th>
<th>Barometer</th>
<th>Humidity</th>
<th>Precipitation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Midtown</strong></td>
<td>47-52°F</td>
<td>5 mph from NNE</td>
<td>1013.2mb</td>
<td>74%</td>
<td>0 in</td>
</tr>
<tr>
<td><strong>Northeast</strong></td>
<td>50-54°F</td>
<td>4 mph from NNE</td>
<td>1015.7mb</td>
<td>70%</td>
<td>0 in</td>
</tr>
<tr>
<td><strong>Airport</strong></td>
<td>48-51°F</td>
<td>8 mph from NNE</td>
<td>1010.0mb</td>
<td>68%</td>
<td>.1 in</td>
</tr>
</tbody>
</table>

Average High Temperature for 2000-11-01, All Locales: **62.33°F**

As of 2000-11-02

<table>
<thead>
<tr>
<th>Location</th>
<th>Temp Range</th>
<th>Wind</th>
<th>Barometer</th>
<th>Humidity</th>
<th>Precipitation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Midtown</strong></td>
<td>43-59°F</td>
<td>1 mph from NE</td>
<td>1009.1mb</td>
<td>80%</td>
<td>2.3 in</td>
</tr>
<tr>
<td><strong>Airport</strong></td>
<td>40-61°F</td>
<td>12 mph from N</td>
<td>1011.0mb</td>
<td>73%</td>
<td>2.0 in</td>
</tr>
</tbody>
</table>

Average High Temperature for 2000-11-02, All Locales: **59.50°F**
Sorting with XSLT

Using XSLT to sort data in an XML document is fairly straightforward. For each sort key you want to use for a given element, add an `<xml:sort>` element as a child of either the `<xsl:for-each>` element or the `<xsl:apply-templates>` element.

The topics in this section refer to the `prodsort` example. The XSLT file and the XML file for this example are at the end of the section.

This section contains the following topics.

- [Basic Sorting in XSLT](#)
- [Sorting on Calculated Variables](#)
- [Sample XML Data File for XSLT Sorting](#)
- [Sample XSLT File for Sorting](#)

**See Also**

- `<xsl:sort>` Element
Basic Sorting in XSLT

The following topics discuss basic sorting, such as sorting rows in a table.

- Sorting in Document Order by Default
- Sorting by a Single Column
- Sorting by Two Columns
- Sorting Data by Numeric Value
Sorting in Document Order by Default

In Internet Explorer, the default way to display a straightforward table is to display content in the order in which it appears in the source document. This default sort order is called the document order.

For a short document, it is not difficult to locate a particular item, even if the document is not rigidly ordered. But for longer documents, it is important for items to be in some logical sequence for easy reference.

To introduce basic sorting, the following is a complete example that does not sort rows.

Example

This XSLT style sheet processes the prodsort.xml file. The output is a simple table with rows for each product in the file, and columns for data such as the product category and product number.

XML File (prodsort.xml)

Use the prodsort.xml in Sample XML Data File for XSLT Sorting, and change the href attribute to reference prodsort0.xsl.

XSLT File (prodsort0.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Wooden Rings and More!</TITLE>
                <STYLE type="text/css">
                    th {background-color: silver; font-family: Tahoma,Verdana,Arial,sans-serif}
<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>AX5608</td>
<td>FooBar/1.5</td>
<td>Processes foo objects using standard FB API</td>
<td>149.99 USD</td>
</tr>
<tr>
<td>Hardware</td>
<td>CB3241</td>
<td>Trihex/3.0</td>
<td>Burns multiple trihexas from single master (requires Wooden Rings 1.6 or greater)</td>
<td>178.49 EUR</td>
</tr>
<tr>
<td>Software</td>
<td>WD3241</td>
<td>Widgetry/1.0</td>
<td>Time-stamps virtual Widgets</td>
<td>46.80 GBP</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9843</td>
<td>Wooden Rings</td>
<td>Enhances WRWB 4.0 with &quot;peel'n'play&quot; capability</td>
<td>129.95 USD</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9840</td>
<td>Workbench/4.5</td>
<td>Applies Widgetry Magick timestamps to Wooden Rings 1.6 or greater</td>
<td>109.95 USD</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR1016</td>
<td>Rings/1.6</td>
<td>Our best-selling oaken toroids</td>
<td>27.99 USD</td>
</tr>
<tr>
<td>Middleware</td>
<td>WD</td>
<td>Virtual Widgets/3.5</td>
<td>Logs critical events and properties over any object's lifetime</td>
<td>109.99 GBP</td>
</tr>
</tbody>
</table>

Processor Output

A portion of the output is shown here.

```html
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<TITLE>Wooden Rings and More!</TITLE>
<STYLE type="text/css">
  th {background-color: silver; 
      font-family: Tahoma, Verdana, Arial, sans-serif}
  td {background-color: white; 
      font-family: Tahoma, Verdana, Arial, sans-serif}
</STYLE>
</HEAD>
<BODY>
<table width="75%">
<tr>
  <th>Category</th>
  <th>Prod ID</th>
  <th>Name/Version</th>
  <th>Description</th>
  <th>Price/Units</th>
</table>
```
<table>
<thead>
<tr>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>AX5608</td>
</tr>
<tr>
<td>FooBar/1.5</td>
<td>Processes foo objects using standard FB API</td>
</tr>
<tr>
<td>149.99 (USD)</td>
<td></td>
</tr>
</tbody>
</table>

...
Sorting by a Single Column

In this topic, we add a single \texttt{<xsl:sort>} element to the above unsorted example.

Example

Each \texttt{<product>} element is processed by a separate product template rule. This rule is invoked as a result of the \texttt{<xsl:apply-templates>} element in the \texttt{products} template rule. Suppose you want to sort your products by category. To do so, add an \texttt{<xsl:sort>} element as a child of the \texttt{<xsl:apply-templates>} element in the \texttt{products} template rule.

\textbf{Note} The \texttt{<xsl:apply-templates>} element, which is typically empty, must be changed to a non-empty start tag/end tag pair. Also note that the value of the \texttt{<xsl:sort>} element's select attribute is relative to the context node established by the \texttt{<xsl:apply-templates>} element. It is not relative to the context node established by the \texttt{<xsl:template>} element's match attribute. Therefore, \texttt{<xsl:sort select="product/categ"/>} will not work correctly.

XML File (prodsort.xml)

Use \texttt{prodsort.xml}, in \texttt{Sample XML Data File for XSLT Sorting}. Change the \texttt{href} attribute to reference \texttt{prodsort1.xsl}.

XSLT File (prodsort1.xsl)

\begin{verbatim}
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <HTML>
        <HEAD>
            <TITLE>Wooden Rings and More!</TITLE>
            <STYLE type="text/css">
                th {background-color: silver; 
                font-family: Tahoma,Verdana,Arial,sans-serif}
            </STYLE>
        
        </HEAD>
        
        
    </HTML>
    
</xsl:template>
</xsl:stylesheet>
\end{verbatim}
<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
</table>

**Formatted Output**
<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craftware</td>
<td>WR9843</td>
<td>Wooden Rings Workbench/4.5</td>
<td>Enhances WRWB 4.0 with &quot;peel'n'play&quot; capability.</td>
<td>129.95 (USD)</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9840</td>
<td>Wooden Rings Workbench/4.0</td>
<td>Applies Widgetry Magick timestamps to Wooden Rings 1.6 or greater.</td>
<td>109.95 (USD)</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR1016</td>
<td>Wooden Rings/1.6</td>
<td>Our best-selling oaken toroids.</td>
<td>27.99 (USD)</td>
</tr>
<tr>
<td>Hardware</td>
<td>CB3241</td>
<td>TrixelMaker/3.0</td>
<td>Burns multiple trixels from single master (requires Wooden Rings 1.6 or greater)</td>
<td>178.49 (EU)</td>
</tr>
<tr>
<td>Middleware</td>
<td>WD</td>
<td>Virtual Widgets/3.5</td>
<td>Logs critical events and properties over any object's lifetime.</td>
<td>109.99 (GBP)</td>
</tr>
<tr>
<td>Software</td>
<td>AX5608</td>
<td>FooBar/1.5</td>
<td>Processes foo objects using standard FB API.</td>
<td>149.99 (USD)</td>
</tr>
<tr>
<td>Software</td>
<td>WD3241</td>
<td>Widgetry Magick/1.0</td>
<td>Time-stamps Virtual Widgets.</td>
<td>45.80 (GBP)</td>
</tr>
</tbody>
</table>
Sorting by Two Columns

The above example makes it much easier to locate products of a particular category. However, within each category, products are still listed in the order in which they appear in the original document. To make finding products even easier, you might want to sort product names within each category. To do so, you can add a second sort level to the first, by adding a second `<xsl:sort>` element.

Example

In the formatted output, note that within the Craftware category, the products are now listed by alphabetical order.

XML File (prodsort.xml)

Use `prodsort.xml`, in Sample XML Data File for XSLT Sorting. Change the `href` attribute to reference `prodsort2.xsl`.

XSLT File (prodsort2.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/"
    >
    <HTML>
      <HEAD>
        <TITLE>Wooden Rings and More!</TITLE>
        <STYLE type="text/css">
          th {background-color: silver; 
              font-family: Tahoma,Verdana,Arial,sans-serif} 
          td {background-color: white; 
              font-family: Tahoma,Verdana,Arial,sans-serif} 
        </STYLE>
      </HEAD>
      <BODY>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```
<xsl:template match="products">
  <TABLE width="75%">
    <tr>
      <th>Category</th>
      <th>Prod ID</th>
      <th>Name/Version</th>
      <th>Description</th>
      <th>Price/Units</th>
    </tr>
    <xsl:apply-templates>
      <xsl:sort select="categ"/>
      <xsl:sort select="name"/>
    </xsl:apply-templates>
  </TABLE>
</xsl:template>

<xsl:template match="product">
  <tr>
    <td valign="top"><xsl:value-of select="categ"/></td>
    <td valign="top"><xsl:value-of select="@prodID"/></td>
    <td valign="top"><xsl:value-of select="concat(name, '/', version)"/></td>
    <td valign="top"><xsl:value-of select="descr"/></td>
    <td valign="top" align="center"><xsl:value-of select="concat(price, ' (', price/@curr, ')')"/></td>
  </tr>
</xsl:template>

<xsl:stylesheet>

Formatted Output
<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craftware</td>
<td>WR.1016</td>
<td>Wooden Rings/1.6</td>
<td>Our best-selling oaken toroids</td>
<td>27.99 (USD)</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR.9643</td>
<td>Wooden Rings Workbench/4.5</td>
<td>Enhances WRWB 4.0 with &quot;peel'n'play&quot; capability</td>
<td>129.95 (USD)</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR.9640</td>
<td>Wooden Rings Workbench/4.0</td>
<td>Applies Widgetry Magick timestamps to Wooden Rings 1.6 or greater</td>
<td>109.95 (USD)</td>
</tr>
<tr>
<td>Hardware</td>
<td>CB.3241</td>
<td>TrixelMaker/3.0</td>
<td>Burns multiple trixels from single master (requires Wooden Rings 1.6 or greater)</td>
<td>178.49 (EU)</td>
</tr>
<tr>
<td>Middleware</td>
<td>WD</td>
<td>Virtual Widgets/3.5</td>
<td>Logs critical events and properties over any object's lifetime</td>
<td>109.99 (GBP)</td>
</tr>
<tr>
<td>Software</td>
<td>AX.5608</td>
<td>FooBar/1.5</td>
<td>Processes foo objects using standard FB API</td>
<td>149.99 (USD)</td>
</tr>
<tr>
<td>Software</td>
<td>WD.3241</td>
<td>Widgetry Magick/1.0</td>
<td>Time-stamps Virtual Widgets</td>
<td>46.80 (GBP)</td>
</tr>
</tbody>
</table>
Sorting Data by Numeric Value

When sorting data, MSXML can treat the sort key as text, or as a number. You can specify how the sort key should be treated in either of the following ways.

```
<xsl:sort select="name" data-type="text"/>
```

or

```
<xsl:sort select="quantity" data-type="number"/>
```

If you do not specify a data type, MSXML treats the sort key as text. If you want the sort key to be treated as a number, you must to set the data-type attribute to "number", explicitly. The reason for specifying a data type other than the default "text" is that non-text values might sort improperly if treated as text. For example, the following are sorted correctly as text, but incorrectly as numbers:

```
1004
109
12
```

The default text sort order compares the values by looking at each character in each value from left to right: 100 is less than 109, so a string beginning with 100 sorts ahead of one beginning with 109. And the leading 10 in both 1004 and 109 sorts ahead of the 12. However, we almost certainly want these values to be sorted as numbers:

```
12
109
1004
```

In Sorting by Two Columns, the sorted table based on prodsort.xml displays products sorted first by categories and then, within each category, by product name. Notice, though, that there are two versions of the Wooden Rings
Workbench product, and that the corresponding two rows of the table are still in document order.

The next example adds a third sort level to the existing two.

**Example**

When you compare the formatted output of the following example to the previous version of the table, note that Wooden Rings Workbench versions 4.0 and 4.5 have switched places.

The version number can be used as a sort key, even though it does not display as a separate column. In fact, any document content can be used as a sort key, even if it is not displayed at all.

**XML File (prodsort.xml)**

Use `prodsort.xml`, in Sample XML Data File for XSLT Sorting. Change the `href` attribute to reference `prodsort3.xsl`.

**XSLT File (prodsort3.xsl)**

This is the final version of this series of XSLT files.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Wooden Rings and More!</TITLE>
                <STYLE type="text/css">
                    th { background-color: silver; }
                    td { background-color: white; }
                </STYLE>
            </HEAD>
            <BODY>
                <xsl:apply-templates/>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
<xsl:template match="products">
  <TABLE width="75%">
    <tr>
      <th>Category</th>
      <th>Prod ID</th>
      <th>Name/Version</th>
      <th>Description</th>
      <th>Price/Units</th>
    </tr>
    <xsl:apply-templates>
      <xsl:sort select="categ"/>
      <xsl:sort select="name"/>
      <xsl:sort select="version" data-type="number"/>
    </xsl:apply-templates>
  </TABLE>
</xsl:template>

<xsl:template match="product">
  <tr>
    <td valign="top"><xsl:value-of select="categ"/></td>
    <td valign="top"><xsl:value-of select="@prodID"/></td>
    <td valign="top"><xsl:value-of select="concat(name, '/', version)"/></td>
    <td valign="top"><xsl:value-of select="descr"/></td>
    <td valign="top" align="center"><xsl:value-of select="concat(price, ' (', price/@curr, ')')"/></td>
  </tr>
</xsl:template>

</xsl:stylesheet>

**Formatted Output**

```
<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
</table>
```
<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price/Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craftware</td>
<td>WR1016</td>
<td>Wooden Rings/1.6</td>
<td>Our best-selling oaken toroids</td>
<td>27.99 (USD)</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9840</td>
<td>Wooden Rings Workbench/4.0</td>
<td>Applies Widgetry Magick timestamps to Wooden Rings 1.6 or greater</td>
<td>109.95 (USD)</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9843</td>
<td>Wooden Rings Workbench/4.5</td>
<td>Enhances WRWB 4.0 with &quot;peel'n'play&quot; capability</td>
<td>129.95 (USD)</td>
</tr>
<tr>
<td>Hardware</td>
<td>CB3241</td>
<td>TrixelMaker/3.0</td>
<td>Burns multiple trixels from single master (requires Wooden Rings 1.6 or greater)</td>
<td>178.49 (EU)</td>
</tr>
<tr>
<td>Middleware</td>
<td>WD</td>
<td>Virtual Widgets/3.5</td>
<td>Logs critical events and properties over any object's lifetime</td>
<td>109.99 (GBP)</td>
</tr>
<tr>
<td>Software</td>
<td>AX5608</td>
<td>FooBar/1.5</td>
<td>Processes foo objects using standard FB API</td>
<td>149.99 (USD)</td>
</tr>
<tr>
<td>Software</td>
<td>WD3241</td>
<td>Widgetry Magick/1.0</td>
<td>Time-stamps Virtual Widgets</td>
<td>46.80 (GBP)</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Sorting on Calculated Variables

The sorting performed by the `<xsl:sort>` elements above, in `prodsort.xsl`, would be suitable for an application like a catalog, in which a given product needs to be located easily.

Suppose, though, that the corporation's chief financial officer asks to see a list of all products, arranged in order by retail price. The first problem this presents is that our data contains a mixture prices in various currency denominations: US dollars (`curr="USD"`), Euros (`curr="EU"`), and British pounds Sterling (`curr="GBP"`). This might seem like an easy problem to fix. You could define a variable, such as `usd_equiv`, which would hold the retail price of the product, converted to its US dollar equivalent. Then, you might think, you could sort on the `usd_equiv` variable, instead of sorting directly on the value of the `<price>` element.

The problem is that you cannot sort directly on a variable. The sort key used as the value of the `<xsl:sort>` element's `select` attribute must be an XPath expression of some kind.

You can, however, sort on a calculated value, using the following general steps:

1. Clone the source tree into a variable, which will then hold a result tree fragment. For more information, see *Introduction to Result Tree Fragments*.
2. Create a new element to hold the calculated result, and append this new element to the cloned source tree created in step 1.
3. Select and sort the result tree fragment created by steps 1 and 2. Do not select and sort the original source tree. Note that this will require the result tree fragment to be converted to a node-set, using the `msxsl:node-set()` function. For more information, see *Using msxsl:node-set() to Process Result-Tree Fragments*.

The remainder of this topic works through each of these steps for the `prodsort` example. The complete solution to the problem stated above, the `prodsort_curr.xsl` style sheet, is available at [Sample XSLT File for Sorting](Sample%20XSLT%20File%20for%20Sorting).

The solution to this problem requires the following steps.
• **Step 1: Cloning the Source Tree**
• **Step 2: Creating the New Element**
• **Step 3: Sorting the Source Tree Clone**
Step 1: Cloning the Source Tree

The following XSLT file contains template rules to process the `prodsort.xml` file. The first template rule handles the root `<products>` element. The first step in that template rule creates a variable, `prods_with_usd`. This variable contains a result tree fragment. This fragment is generated by the second template rule, which contains a copy of each `<product>` element, its attributes, and its children. Note the use of the `mode` attribute, which causes this second template rule to be fired only when the `<apply-templates>` element has a matching `mode`.

After making the changes shown below, the `prods_with_usd` variable simply contains a root node, the seven `<product>` elements, and their children. If we stopped here, there'd be no difference in the data available to be sorted.

**XML File (prodsort.xml)**

Use `prodsort.xml`, in [Sample XML Data File for XSLT Sorting](#). Change the `href` attribute to reference `prodsortcvar1.xsl`.

**XSLT File (prodsortcvar1.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Wooden Rings and More!</TITLE>
                <STYLE type="text/css">
                    th {background-color: silver;
                        font-family: Tahoma, Verdana, Arial, sans-serif}
                    td {background-color: white;
                        font-family: Tahoma, Verdana, Arial, sans-serif}
                </STYLE>
            </HEAD>
            <BODY>
                <xsl:apply-templates/>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
<xsl:template match="products">
  <!-- Create a variable to hold the generated result tree fragment. -->
  <xsl:variable name="prods_with_usd">
    <xsl:apply-templates select="product" mode="calc_usd"/>
  </xsl:variable>
  <TABLE width="75%">
    <tr>
      <th>Category</th>
      <th>Prod ID</th>
      <th>Name/Version</th>
      <th>Description</th>
      <th>Price (Currency)</th>
      <th>Price (USD)</th>
    </tr>
  </TABLE>
</xsl:template>

<xsl:template match="product" mode="calc_usd">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:copy-of select="*"/>
  </xsl:copy>
</xsl:template>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Step 2: Creating the New Element

Step 2 adds code to the second template rule described in Step 1: Cloning the Source Tree—that is, the rule fired as a result of the mode="calc_usd" attribute. The new code creates a new child for each <product> element. This new child, <usd_equiv>, contains the US dollars equivalent of the product's price in local currency. The prodsortcvar2.xsl example, below, shows this template rule, with changes in bold.

Note  The element created by the <xsl:element> here could also have been created as a literal result element. That is, the <xsl:element name="usd_equiv"/> start tag could be replaced with <usd_equiv>, and the </xsl:element> end tag could be replaced with </usd_equiv>.

The changes below create a variable, prods_with_usd. This variable contains a result tree fragment consisting of each product's data (copied from the source tree), and a new element for each product, <usd_equiv>. All that's left is to apply the sort to this slightly altered source tree clone, and place its results in table rows (one per product).

XML File (prodsort.xml)

Use prodsort.xml, in Sample XML Data File for XSLT Sorting. Change the href attribute to reference prodsortcvar2.xsl.

XSLT File (prodsortcvar2.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <HTML>
        <HEAD>
            <TITLE>Wooden Rings and More!</TITLE>
            <STYLE type="text/css">
                th {background-color: silver;
                    font-family: Tahoma,Verdana,Arial,sans-serif}
                td {background-color: white;
                    font-family: Tahoma,Verdana,Arial,sans-serif}
            </STYLE>
        </HEAD>
        <!-- More content -->
    </HTML>
</xsl:template>
</xsl:stylesheet>
```
<xsl:template match="products">
    <!-- Create a variable to hold the generated result tree fragment -->
    <xsl:variable name="prods_with_usd">
        <xsl:apply-templates select="product" mode="calc_usd"/>
    </xsl:variable>
    <TABLE width="75%">
        <tr>
            <th>Category</th>
            <th>Prod ID</th>
            <th>Name/Version</th>
            <th>Description</th>
            <th>Price (Currency)</th>
            <th>Price (USD)</th>
        </tr>
        <xsl:apply-templates select="$prods_with_usd"/>
    </TABLE>
</xsl:template>

<xsl:template match="product" mode="calc_usd">
    <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:copy-of select="*"/>
        <!-- Add a calculated <usd_equiv> child to each <product>. -->
        <xsl:element name="usd_equiv">
            <xsl:choose>
                <!-- Note: Exchange rates current as of Oct. 2000 -->
                <xsl:when test="price/@curr='USD'">
                    <xsl:value-of select="format-number(price, '#,##0.00')"/>
                </xsl:when>
                <xsl:when test="price/@curr='GBP'">
                    <xsl:value-of select="format-number(price * 1.47, '#,##0.00')"/>
                </xsl:when>
                <xsl:when test="price/@curr='EU'">
                    <xsl:value-of select="format-number(price * 0.86, '#,##0.00')"/>
                </xsl:when>
                <xsl:otherwise>Unknown Currency</xsl:otherwise>
            </xsl:choose>
        </xsl:element>
    </xsl:copy>
</xsl:template>
Step 3: Sorting the Source Tree Clone

To sort the modified clone of the source tree, convert it to a node-set, using the `msxsl:node-set()` function. Then pass it, in its sorted form, to a template rule that handles it the same way the original source tree itself would be handled.

To do this, add some new lines to the template rule for the root `<products>` element. This is illustrated in the example below.

**Note** To use the `msxsl:node-set()` function, the style sheet's `<xml:stylesheet>` element must include the following namespace declaration:

```xml
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
```

**Example**

In the formatted output at the end of this topic, note the position of the Virtual Widgets product. If the products had been sorted by the `<price>` element alone, the position of this product would be considerably different.

Also note the new column in the table, which lists the value of the new `<usd_equiv>` element.

For this technique to work, the style sheet must not include a namespace declaration for the HTML 4.0 namespace, which you will typically include when transforming XML to HTML that is compatible with Internet Explorer. This is because the generated `<usd_equiv>` element, which includes no namespace prefix, is in the default namespace. But because `<usd_equiv>` is not an HTML element, it will not appear in the result tree.

**XML File (prodsort.xml)**

Use `prodsort.xml`, in Sample XML Data File for XSLT Sorting. Change the `href` attribute to reference `prodsortcvar.xsl`.

**XSLT File (prodsortcvar.xsl)**
The product template rule with a mode attribute and a new `<xsl:apply-templates>` element invokes another product template rule. This rule is used for `<product>` elements that do not have a mode attribute. These two template rules are shown below, and the `prodsortcvar.xsl` file is now complete. For the entire completed `prodsortcvar.xsl` file, see Sample XSLT File for Sorting.

```xml
<xsl:template match="products">
   <!-- Create a variable to hold the generated result tree fragment -->
   <xsl:variable name="prods_with_usd">
      <xsl:apply-templates select="product" mode="calc_usd"/>
   </xsl:variable>
   <TABLE width="75%">
      <tr>
         <th>Category</th>
         <th>Prod ID</th>
         <th>Name/Version</th>
         <th>Description</th>
         <th>Price (Currency)</th>
         <th>Price (USD)</th>
      </tr>
      <!-- Fill remainder of table with source tree "clone". -->
      <xsl:apply-templates select="msxsl:node-set($prods_with_usd)"
         <xsl:sort select="usd_equiv" data-type="number"/>
      </xsl:apply-templates>
   </TABLE>
</xsl:template>

<xsl:template match="product">
   <tr>
      <td valign="top"><xsl:value-of select="categ"/></td>
      <td valign="top"><xsl:value-of select="@prodID"/></td>
      <td valign="top"><xsl:value-of select="concat(name, '/', version)"/></td>
      <td valign="top"><xsl:value-of select="descr"/></td>
      <td valign="top" align="center"><xsl:value-of select="concat(price, ' (', price/@curr, ')')"/></td>
      <td valign="top" align="right"><xsl:value-of select="usd_equiv"/></td>
   </tr>
</xsl:template>
</xsl:stylesheet>
```

**Formatted Output**
### Processor Output

A portion of the output is shown here.

```html
<HTML xmlns:msxsl="urn:schemas-microsoft-com:xslt">
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<TITLE>Wooden Rings and More!</TITLE>
<STYLE type="text/css">
    th {background-color: silver;
        font-family: Tahoma,Verdana,Arial,sans-serif}
    td {background-color: white;
        font-family: Tahoma,Verdana,Arial,sans-serif}
</STYLE>
</HEAD>
<BODY>
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Prod ID</th>
<th>Name/Version</th>
<th>Description</th>
<th>Price (Currency)</th>
<th>Price (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craftware</td>
<td>WR1016</td>
<td>Wooden Rings/1.6</td>
<td>Our best-selling oaken toroids</td>
<td>27.99 (USD)</td>
<td>27.99</td>
</tr>
<tr>
<td>Software</td>
<td>WD3241</td>
<td>Widgetry Magick/1.0</td>
<td>Time-stamps Virtual Widgets</td>
<td>46.80 (GBP)</td>
<td>68.92</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9840</td>
<td>Wooden Rings Workbench/4.0</td>
<td>Applies Widgetry Magick timestamps to Wooden Rings 1.6 or greater</td>
<td>109.95 (USD)</td>
<td>109.95</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR9843</td>
<td>Wooden Rings Workbench/4.5</td>
<td>Enhances WRWB 4.0 with &quot;peel'n'play&quot; capability</td>
<td>129.95 (USD)</td>
<td>129.95</td>
</tr>
<tr>
<td>Software</td>
<td>AX5608</td>
<td>FooBar/1.5</td>
<td>Processes foo objects using standard FB API</td>
<td>149.99 (USD)</td>
<td>149.99</td>
</tr>
<tr>
<td>Hardware</td>
<td>CB3241</td>
<td>TrixelMaker/3.0</td>
<td>Burns multiple trixels from single master (requires Wooden Rings 1.6 or greater)</td>
<td>178.49 (EU)</td>
<td>154.28</td>
</tr>
<tr>
<td>Middleware</td>
<td>WD</td>
<td>Virtual Widgets/3.5</td>
<td>Logs critical events and properties over any object's lifetime</td>
<td>109.99 (GBP)</td>
<td>161.99</td>
</tr>
<tr>
<td>Category</td>
<td>Prod ID</td>
<td>Name/Version</td>
<td>Description</td>
<td>Price (Currency)</td>
<td>Price (USD)</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>----------------------</td>
<td>--------------------------------------------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Craftware</td>
<td>WR1016</td>
<td>Wooden Rings/1.6</td>
<td>Our best-selling oaken toroids</td>
<td>27.99 (USD)</td>
<td>27.99</td>
</tr>
<tr>
<td>Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Sample XML Data File for XSLT Sorting

The following XML file contains data about a vendor's products. It is used throughout the Sorting with XSLT section.

XML File (prodsort.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="prodsort.xsl" ?>
<products>
  <product prodID="AX5608">
    <name>FooBar</name>
    <version>1.5</version>
    <descr>Processes foo objects using standard FB API</descr>
    <categ>Software</categ>
    <price curr="USD">149.99</price>
    <units>each</units>
  </product>
  <product prodID="CB3241">
    <name>TrixelMaker</name>
    <version>3.0</version>
    <descr>Burns multiple trixels from single master (requires Wooden Rings 1.6 or greater)</descr>
    <categ>Hardware</categ>
    <price curr="EU">178.49</price>
    <units>each</units>
  </product>
  <product prodID="WD3241">
    <name>Widgetry Magick</name>
    <version>1.0</version>
    <descr>Time-stamps Virtual Widgets</descr>
    <categ>Software</categ>
    <price curr="GBP">46.80</price>
    <units>each</units>
  </product>
  <product prodID="WR9843">
    <name>Wooden Rings Workbench</name>
    <version>4.5</version>
    <descr>Enhances WRWB 4.0 with "peel'n'play" capability</descr>
    <categ>Craftware</categ>
    <price curr="USD">129.95</price>
    <units>each</units>
  </product>
</products>
```
<product prodID="WR9840">
  <name>Wooden Rings Workbench</name>
  <version>4.0</version>
  <descr>Applies Widgetry Magick timestamps to Wooden Rings 1.6 or greater</descr>
  <categ>Craftware</categ>
  <price curr="USD">109.95</price>
  <units>each</units>
</product>

<product prodID="WR1016">
  <name>Wooden Rings</name>
  <version>1.6</version>
  <descr>Our best-selling oaken toroids</descr>
  <categ>Craftware</categ>
  <price curr="USD">27.99</price>
  <units>dozen</units>
</product>

<product prodID="WD">
  <name>Virtual Widgets</name>
  <version>3.5</version>
  <descr>Logs critical events and properties over any object's lifetime</descr>
  <categ>Middleware</categ>
  <price curr="GBP">109.99</price>
  <units>each</units>
</product>

</products>
Sample XSLT File for Sorting

This is the complete final file for the topics in the Sorting on Calculated Variables section. Other topics in the Sorting with XSLT section use different versions of the prodsortx.xsl file.

XSLT File (prodsortcvar.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">

<xsl:template match="/">
    <HTML>
        <HEAD>
            <TITLE>Wooden Rings and More!</TITLE>
            <STYLE type="text/css">
                th {background-color: silver;
                font-family: Tahoma,Verdana,Arial,sans-serif}
                td {background-color: white;
                font-family: Tahoma,Verdana,Arial,sans-serif}
            </STYLE>
        </HEAD>
        <BODY>
            <xsl:apply-templates/>
        </BODY>
    </HTML>
</xsl:template>

<xsl:template match="products">

<!-- Create a variable to hold the generated result tree fragment -->
<xsl:variable name="prods_with_usd">
    <xsl:apply-templates select="product" mode="calc_usd"/>
</xsl:variable>

<TABLE width="75%">
    <tr>
        <th>Category</th>
        <th>Prod ID</th>
        <th>Name/Version</th>
        <th>Description</th>
        <th>Price (Currency)</th>
        <th>Price (USD)</th>
    </tr>
</xsl:template>

<!-- Fill remainder of table with source tree "clone". -->
```
<xsl:apply-templates select="msxsl:node-set($prods_with_usd)/product">
  <xsl:sort select="usd_equiv" data-type="number"/>
</xsl:apply-templates>
</TABLE>
</xsl:template>

<xsl:template match="product" mode="calc_usd">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:copy-of select="*"/>
    <!-- Add a calculated <usd_equiv> child to each <product>.
    -->
    <xsl:element name="usd_equiv">
      <xsl:choose>
        <!-- Note: Exchange rates current as of Oct. 2000. -->
        <xsl:when test="price/@curr='USD'">
          <xsl:value-of select="format-number(price, '#,##0.00')"/>
        </xsl:when>
        <xsl:when test="price/@curr='GBP'">
          <xsl:value-of select="format-number(price * 1.47275, '#,##0.00')"/>
        </xsl:when>
        <xsl:when test="price/@curr='EU'">
          <xsl:value-of select="format-number(price * 0.864379, '#,##0.00')"/>
        </xsl:when>
        <xsl:otherwise>Unknown Currency</xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:copy>
</xsl:template>

<xsl:template match="product">
  <tr>
    <td align="top">
      <xsl:value-of select="categ"/>
    </td>
    <td align="top">
      <xsl:value-of select="@prodID"/>
    </td>
    <td align="top">
      <xsl:value-of select="concat(name, '/', version)"/>
    </td>
    <td align="top">
      <xsl:value-of select="descr"/>
    </td>
    <td align="center">
      <xsl:value-of select="concat(price, ' (', price/@curr, ')')"/>
    </td>
    <td align="right">
      <xsl:value-of select="usd_equiv"/>
    </td>
  </tr>
</xsl:template>
Working with Namespaces and Prefix Aliases

This section contains the following topics.

- Introduction to Namespaces in XSLT
- Declaring Namespaces for XSLT
- Changing Namespace Aliases Programmatically

See Also

<xsl:namespace-alias> Element
Introduction to Namespaces in XSLT

A namespace defines a class (or set) of names of elements and attributes designed for use in the context of a specific technology. For example, the elements and attributes for XHTML, such as `<HTML>`, `<BODY>`, `<HREF>`, and so on, share the XHTML namespace. The elements defined in the XSLT Version 1.0 specification, such as `<xsl:stylesheet>`, `<xsl:template>`, `<xsl:if>`, and so on, share the XSLT 1.0 namespace, `xs1`.

A namespace is identified by a URI (for example, `http://www.w3.org/1999/XSL/Transform`), and by one or more namespace prefixes (for example, `xs1`) that serve as the short-hand of the URI. In general, an element belonging to a particular namespace is written in the format of `namespace-prefix:element-name`. For example, `<xs1:sort>`.

Namespaces help to avoid collisions of names of different classes. For example, `<my:if>` and `<xs1:if>` differentiate the `<if>` element of the xs1 namespace from that of the my namespace.

**Note**  Technically, attribute names without a prefix are assumed to be in no namespace in particular. To ensure that an attribute comes from a specified namespace, use the namespace prefix when specifying the attribute.

For more information about how to associate a particular prefix with a particular namespace, see the next section, Declaring Namespaces for XSLT.
Declaring Namespaces for XSLT

To declare a namespace, set the xmlns attribute on an element. For example:

```xml
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/REC-html40">

The <stylesheet> element declares an XSLT style sheet that will reference elements defined in the XSLT 1.0 and HTML namespaces. The XSLT elements will use an xsl prefix for the names. The HTML elements, on the other hand, will have no prefixes in their names. The namespace declarations apply to the element and all its children.

Declaring a namespace means choosing a namespace prefix, and binding the prefix to the appropriate namespace URI. When declaring namespaces, remember the following:

- The choice of a namespace prefix is unimportant to the XSLT processor, although it should be meaningful for human readers (for documentation purposes). The main constraint is that it must be a valid XML name. A valid XML name begins with a letter or underscore, and has no embedded spaces. Therefore, an XSLT style sheet does not have to use the conventional xsl: prefix. It could use something like xslt:, transform:, or XSLT_Stylesheet_Prefix:, instead.
- The choice of the URI, however, is critical. For XSLT style sheets, you can associate any prefix with the XSLT namespace URI, http://www.w3.org/1999/XSL/Transform. But any element or attribute whose name begins with that prefix will be treated as an instruction by the XSLT processor. If you change even a single character of this namespace URI, however, the XSLT processor will assume that an element or attribute whose name starts with the given prefix is something other than an XSLT instruction.

**Note** Not all URIs take the form of a traditional Uniform Resource Locator (URL), such as http://www.mycompany.com. In the case of the MSXML processor, in particular, Microsoft extensions to the XSLT standard, such as the msxsl:node-set() function, are identified
by a prefix associated with the URI urn:schemas-microsoft-com:xslt. The following namespace declaration is an example.

```xml
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
```

This URI form is called a Uniform Resource Name (URN). Again, whether the URI takes the form of a locator or a name, the specific URI cannot be changed if you want to take advantage of a feature supported by a particular processor.

**See Also**

[Declaring Namespaces](#)
Example of Declaring Namespaces in XSLT

Example

This example produces a table that lists each element in the style sheet, in document order, and provides some information about the parent of each element to make it easier to locate a specific instance of an element. The table also shows the namespaces associated with each element—that is, the namespace the element itself belongs to, and all the namespaces in scope for that element.

To view this example, view the XSLT file in Internet Explorer.

XML File

No XML file is required, because the XSLT file calls itself.

If you want to see the XSLT file applied to a separate XML file, you can use the Sample XML File (books.xml). Below the line `<?xml version='1.0'?>`, add the following line:

```xml
<?xml-stylesheet type="text/xsl" href="ns_decl.xsl"?>
```

XSLT File (ns_decl.xsl)

This file invokes itself, so you can view it directly in Internet Explorer. You could also use this style sheet with any XML document. Code related to namespaces is highlighted.

```xml
<?xml-stylesheet type="text/xsl" href="ns_decl.xsl"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40" >

<xsl:template match="text()"/>

<xsl:template match="/">
    <HTML>
```
<table>
<thead>
<tr>
<th>Child</th>
<th>Parent</th>
<th>Child #</th>
<th>Namespace URI</th>
<th>Namespace(s) in Scope</th>
</tr>
</thead>
</table>

Formatted Output
The following output is for the elements used to generate the value of the `root_or_name` variable in the XSLT file. Notice that the namespace prefix was changed from `xsl` to `transform`, but the namespace URI remains unchanged. So `<transform:variable>` is equivalent to `<xsl:variable>`. This applies to all its children, as well.

<table>
<thead>
<tr>
<th>Child Name</th>
<th>Parent</th>
<th>Child #</th>
<th>Namespace URI</th>
<th>Namespace(s) in Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xsl:stylesheet</code></td>
<td>--ROOT--</td>
<td>2 of 2</td>
<td><a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a></td>
<td><a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a></td>
</tr>
<tr>
<td><code>html</code></td>
<td><code>xsl:template</code></td>
<td>1 of 1</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
</tr>
<tr>
<td><code>body</code></td>
<td><code>html</code></td>
<td>1 of 1</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
</tr>
<tr>
<td><code>table</code></td>
<td><code>body</code></td>
<td>1 of 1</td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
<td><a href="http://www.w3.org/TR/REC-html40">http://www.w3.org/TR/REC-html40</a></td>
</tr>
</tbody>
</table>

**Processor Output**
A portion of the XSLT processor output is shown here, with line breaks added for clarity.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<HTML xmlns="http://www.w3.org/TR/REC-html40">
<BODY>
<TABLE border="1">
<tr style="background-color: silver">
<th>Child Name</th>
<th>Parent</th>
<th>Child #</th>
<th>Namespace URI</th>
<th>Namespace(s) in Scope</th>
</tr>
<tr>
<td valign="top">xsl:stylesheet</td>
<td valign="top">--ROOT--</td>
<td valign="top">2 of 2</td>
<td valign="top">http://www.w3.org/1999/XSL/Transform</td>
<td>http://www.w3.org/XML/1998/namespace
 http://www.w3.org/1999/XSL/Transform
 http://www.w3.org/TR/REC-html40</td>
</tr>
<tr>
<td valign="top">xsl:template</td>
<td valign="top">xsl:stylesheet</td>
<td valign="top">1 of 3</td>
<td valign="top">http://www.w3.org/1999/XSL/Transform</td>
</tr>
</TABLE>
</BODY></HTML>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Changing Namespace Aliases Programmatically

Suppose you want to use an XSLT style sheet to produce another XSLT style sheet. In such an application, you will have to deal with two kinds of XSLT elements. One set of elements consists of the XSLT instructions that tell the XSLT processor to perform the transformation. The other set consists of the XSLT elements that will be output. These elements are the literal result elements. During the transformation, you will have to use different namespaces for these two types of elements.

The following is a simple example.

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml"/>
    <xsl:template match="text()">
        <xsl:value-of select="."/>
    </xsl:template>
    <xsl:template match="/">Template for root mapped.</xsl:template>
</xsl:stylesheet>
```

We cannot put this inside a template rule, like this:

```xml
<xsl:template match="/">
    <!-- We can't put the above literal results elements here. -->
</xsl:template>
```

An `<xsl:stylesheet>` element cannot be a child of any XSLT element. The solution to this problem is to treat the two sets of XSLT elements as belonging to different namespaces during the transformation. Use the regular XSLT namespace for the XSLT instruction elements, and use a temporary namespace for the literal result elements. At the end of the transformation, change the output elements from the temporary namespace to the regular XSLT namespace.
An XSLT Style Sheet Can Generate another XSLT Style Sheet

The following style sheet generates another style sheet.

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="simple.xsl"/>
<!-- Declare two namespaces: "xsl" is for XSLT instructions, and "out" is for literal result elements, temporarily. -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:out="transform.xsl">
    <xsl:output method="xml" indent="yes"/>
    <!-- The following instruction switches the result element's namespace from "out" to "xsl". -->
    <xsl:namespace-alias stylesheet-prefix="out" result-prefix="xsl"/>
    <xsl:template match="/">
        <out:stylesheet>
            <out:output method="xml"/>
            <out:template match="text()">
                <out:value-of select="."/>
            </out:template>
            <out:template match="/">Template for root mapped.</out:templa
        </out:stylesheet>
    </xsl:template>
</xsl:stylesheet>
```

The use of the temporary namespace (xmlns:out="temporary.xsl") prevents the XSLT processor from processing the output XSLT elements. It is important that both the namespace prefix and the URI differ for the result and instruction elements. The <xsl:namespace-alias> element re-maps the namespace from out to xsl (including the corresponding URIs) on the result elements.

Output XSLT Style Sheet

The following is the output of this XSLT transformation.

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```
<xsl:stylesheet>
  <xsl:output method="xml"/>
  <xsl:template match="text()">
    <xsl:value-of select="."/>
  </xsl:template>
  <xsl:template match="/">Template for root mapped</xsl:template>
</xsl:stylesheet>
Using msxsl:node-set() to Process Result-Tree Fragments

A result tree fragment is a part of a result tree that is not a node set. The following is an example.

The `<color hue="red">red</color>` color is `<value brightness="bright">bright</value>`.

To process a result tree fragment, you must convert it to a node set. MSXML provides the `msxsl:node-set()` extension function to do the conversion.

- The following topics illustrate the use of the `msxsl:node-set()` extension function.
  - Introduction to Result Tree Fragments
  - Declaring the msxsl: Namespace Prefix
  - Processing a Tree Fragment as a Tree by Using msxsl:node-set()
  - Calculating a Sum of Sums Using msxsl:node-set()

See Also

Using Functions in XPath Expressions
Introduction to Result Tree Fragments

The XPath specification defines four basic data types that are accessible in XPath location steps, function references, and so on:

- strings
- numbers
- Booleans
- node-sets

To these basic XPath data types, the XSLT specification adds a fifth: result tree fragments.

A result tree fragment is a part of the result tree of an XSLT transformation. That is, it is a template to be instantiated in the result tree—specifically, a template instantiated by a non-empty `<xsl:variable>` or `<xsl:param>` element. The result tree fragment can be well-formed, but does not have to be. It only has to be well-balanced—that is, each start tag must include an end tag, the tags must be nested properly, and so on.

A data set differs from a result tree fragment in that the data set has a root element containing other elements. This root element can the document's own root element, or a child of the document's root element.

The following code fragment from an XSLT style sheet is an example of how an `<xsl:variable>` element can create a non-well-formed, but well-balanced, result tree fragment.

```xml
<xsl:variable name="tree">
   <em>Emphasized and <b>bold</b></em> words
</xsl:variable>
```

The result tree instantiated by this variable will be the following:

```xml
<em>Emphasized and <b>bold</b></em> words
```
That is, it consists of:

- A complete \textit{element}, containing some text and a \textbf{element}, which itself includes some text.
- Some trailing text: the string "words".

If you define a variable or parameter which is not empty, you will often want to then attempt some operation on the value of the variable or parameter. If you try to do so using built-in XPath and XSLT functions, it will probably not produce the desired result. This is because those functions are intended to be used with one of the four basic data types, not with result tree fragments.

**XSLT File (rtf.xsl)**

Suppose you want to walk the tree established by the $tree variable in the fragment above. The following XSLT file attempts to do this. The file contains a template rule to be applied to the XML result tree fragment, \textit{Emphasized and \textbf{bold} words}.

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="rtf.xsl" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:variable name="tree">
\textit{Emphasized and \textbf{bold} words}
</xsl:variable>

<xsl:template match="/">
  <table border="1" cellpadding="2" cellspacing="0">
    <tr>
      <th>Node name</th>
      <th>Node type</th>
    </tr>
    <!-- Show document root and each element in $tree variable. -->
    <xsl:for-each select="$tree/descendant-or-self::*|$tree">
      <tr>
        <td>
          <xsl:choose>
            <xsl:when test="name()='"">
              (DOC ROOT)
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="name()"/>
            </xsl:otherwise>
          </xsl:choose>
        </td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>
```
Result

You might apply this style sheet to itself, as follows:

Try
{
    xslt.transformNode(xslt);
}
catch (e)
    alert(e.description);
}

This results in the following runtime error:

    Expression must evaluate to a node-set. -->$tree<--/...

Here xslt refers to a DOM instance of the XSLT style sheet. This example might seem artificial, but you will almost certainly encounter similar problems in real-world situations, such as sorting and grouping.

To get around this common problem, you can use msxsl:node-set() to convert the result tree fragment into a node set.
Declaring the msxsl: Namespace Prefix

Before using the msxsl:node-set() function in an XSLT style sheet, you must declare the msxml: namespace prefix. Most often this is done in the root <xml:stylesheet> element of the style sheet, as shown below:

XSLT File (msxsldecl.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">
    <xsl:template match="text()"/>
</xsl:stylesheet>
```

As with other namespace declarations, the prefix (such as msxsl:) is not important. Only the namespace URI (that is, the value of the xmlns attribute) is important. To use the Microsoft implementation of the node-set() extension function, that value must be "urn:schemas-microsoft-com:xslt". For example, you could use either of the following two namespace declarations, as long as all references to the node-set() function appear as ms:node-set() or xyz:node-set(), respectively.

- xmlns:ms="urn:schemas-microsoft-com:xslt"
- xmlns:xyz="urn:schemas-microsoft-com:xslt"

Throughout this topic, we have used the msxsl: prefix.

**Note** The Uniform Resource Identifier (URI) for using Microsoft extensions to core W3C technologies takes the form of a Uniform Resource Name (URN), such as urn:schemas-microsoft-com:xslt. This is an accepted alternative to a Uniform Resource Locator (URL), such as http://www.w3.org/1999/XSL/Transform.

Also note that declaring the msxsl: namespace gives your style sheet access
not only to the `msxsl:node-set()` function, but also to other Microsoft extensions, such as the `<msxsl:script>` element.
Processing a Tree Fragment as a Tree by Using msxsl:node-set()

According to the XSLT specification, you can do only two things with a result tree fragment that is returned as the value of an `<xsl:variable>` or `<xsl:param>` element: copy the fragment to the result tree, or treat it as a string.

Often, however, you will want to treat the fragment as a tree or sub-tree, to enable processing by a further step (such as sorting or accumulating a sum).

To treat a result tree fragment as a tree or sub-tree

1. Declare the `msxsl:` namespace. For more information, see Declaring the `msxsl:` Namespace Prefix.
2. Establish the result tree fragment, using a variable or parameter.
3. Convert the result tree fragment to a node-set, using the `msxsl:node-set()` function.
4. Use the converted node-set as you would use any other node-set, in XPath expressions, function calls, and so on.

The following is a simple example of Step 4.

```xml
<xsl:value-of select="msxsl:node-set($emprecs)/employee"/>
```

This example treats the variable or parameter called `$emprecs` as if it were a tree of nodes. An instance of an `<employee>` child of that tree of nodes is created in the result tree.
Calculating a Sum of Sums Using msxsl:node-set()

The initial example for sum of sums uses an XML document for an invoicing application. A common objective of an invoicing application is to show, for each item purchased, the number of items purchased, the unit price, and the total price. This could be accomplished with the XSLT file shown in the initial example in this section, inv.xsl.

This section contains the following topics:

- Initial Example for Sum of Sums
- Calculating a Sum of Sums (First Attempt)
- Calculating a Sum of Sums (Second Attempt)
- Calculating a Sum of Sums (Final Version)
Initial Example for Sum of Sums

This example generates a table with a header row and one row of data for each item. In subsequent topics, we add sums to this table.

XML File (invoice.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="inv.xsl" ?>
<invoice>
  <item item_num="AX43598">
    <quantity>29</quantity>
    <unit_price>2.00</unit_price>
  </item>
  <item item_num="FH29782">
    <quantity>140</quantity>
    <unit_price>6.50</unit_price>
  </item>
</invoice>
```

XSLT File (inv.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <table border="1" cellpadding="2" cellspacing="0">
      <tr>
        <th>Item</th>
        <th>Quantity</th>
        <th>Unit Price</th>
        <th>Extended Price</th>
      </tr>
      <xsl:for-each select="invoice/item">
        <tr>
          <td>
            <xsl:value-of select="@item_num"/>
          </td>
          <td align="right">
            <xsl:value-of select="quantity"/>
          </td>
          <td align="right">
            <xsl:value-of select="format-number(unit_price,
```
### Formatted Output

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Unit Price</th>
<th>Extended Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX43598</td>
<td>29</td>
<td>2.00</td>
<td>58.00</td>
</tr>
<tr>
<td>FH29782</td>
<td>140</td>
<td>6.50</td>
<td>910.00</td>
</tr>
</tbody>
</table>
Calculating a Sum of Sums (First Attempt)

The Initial Example for Sum of Sums, inv.xsl, does not generate the sum of all extended prices. If you needed this kind of functionality, you would have to enhance your style sheet. As a first attempt at a solution, you might try something like the following. This third <tr> element is added to inv.xsl, resulting in inv2.xsl.

```xml
<tr>
    <th colspan="3">Invoice Total</th>
    <td align="right">
        <xsl:value-of select="format-number(sum(item/unit_price),',#0.00')"/>
    </td>
</tr>
```

The result of this approach is shown in the following example.

Example

XML File (invoice.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="inv2.xsl" ?>
<invoice>
    <item item_num="AX43598">
        <quantity>29</quantity>
        <unit_price>2.00</unit_price>
    </item>
    <item item_num="FH29782">
        <quantity>140</quantity>
        <unit_price>6.50</unit_price>
    </item>
</invoice>
```

XSLT File (inv2.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```
### Result

Internet Explorer (or any other compliant XSLT processor) treats this as an error. The following is the message from Internet Explorer:

*Expression does not return a DOM node. format-number(-->sum(item/quantity * item/unit_price)<-- , '#,##0.00')*
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer’s Guide
Calculating a Sum of Sums (Second Attempt)

As a next attempt, we'll declare a variable to accumulate and hold the extended price. Then we'll try to use that variable for the sum operation. The style sheet now looks like this:

```xsl
<xsl:variable name="ext_price">
    <xsl:for-each select="invoice/item">
        <accum><xsl:value-of select="quantity * unit_price"/></accum>
    </xsl:for-each>
</xsl:variable>

<xsl:template match="/">
    ...
    <tr>
        <th colspan="3">
            Total, All Items
        </th>
        <td align="right">
            <xsl:value-of select="format-number(sum($ext_price), '#,##0.00')"/>
        </td>
    </tr>
    ...
</xsl:template>
```

The result of this attempt is shown in the following example.

**Example**

**XML File (invoice.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="inv3.xsl"?>
<invoice>
    <item item_num="AX43598">
        <quantity>29</quantity>
        <unit_price>2.00</unit_price>
    </item>
    <item item_num="FH29782">
        <quantity>140</quantity>
        <unit_price>6.50</unit_price>
    </item>
</invoice>
```
XSLT File (inv3.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:variable name="ext_price">
    <xsl:for-each select="invoice/item">
        <accum><xsl:value-of select="quantity * unit_price"/></accum>
    </xsl:for-each>
</xsl:variable>

<xsl:template match="/">
    <table border="1" cellspacing="0">
        <tr>
            <th>Item</th>
            <th>Quantity</th>
            <th>Unit Price</th>
            <th>Extended Price</th>
        </tr>
        <xsl:for-each select="invoice/item">
            <tr>
                <td>
                    <xsl:value-of select="@item_num"/>
                </td>
                <td align="right">
                    <xsl:value-of select="quantity"/>
                </td>
                <td align="right">
                    <xsl:value-of select="format-number(unit_price, '#,##0.00')"/>
                </td>
                <td align="right">
                    <xsl:value-of select="format-number(quantity * unit_price, '#,##0.00')"/>
                </td>
            </tr>
        </xsl:for-each>
        <tr>
            <th colspan="3">Total, All Items</th>
            <td align="right">
                <xsl:value-of select="format-number(sum($ext_price/accum), '#,##0.00')"/>
            </td>
        </tr>
    </table>
</xsl:template>
</xsl:stylesheet>
```

Result
Now when we try to view our sample document in Internet Explorer, we get another error message:

Reference to variable or parameter 'ext_price' must evaluate to a node list.

Even though the `<accum>` element contains numeric data, the XPath `sum()` function must take a node-set as an argument. To correct the problem, see the next topic, Calculating a Sum of Sums (Final Version).
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Calculating a Sum of Sums (Final Version)

To correct the failed attempt in the previous topic, wrap the reference to the $ext_price variable in the msxsl:node-set() function call, as follows:

```xml
<tr>
    <th colspan="3">Total, All Items</th>
    <td align="right">
        <xsl:value-of select="format-number(sum(msxsl:node-set($ext_price)))"/>
    </td>
</tr>
```

You also need to declare the msxsl: namespace prefix:

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">
```

Now the table contains the original data and the total of the extended prices. The finished example is shown below.

For more information about the sum() function, see [Using Numeric Functions to Perform Math Operations in XPath](#).

**Example**

**XML File (invoice.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="invtot.xsl" ?>
<invoice>
    <item item_num="AX43598"
        quantity="29" unit_price="2.00"/>
    <item item_num="FH29782"
        quantity="140" unit_price="6.50"/>
</invoice>
```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">
    <xsl:variable name="ext_price">
        <xsl:for-each select="invoice/item">
            <accum><xsl:value-of select="quantity * unit_price"/></accum>
        </xsl:for-each>
    </xsl:variable>

    <xsl:template match="/">
        <table border="1" cellpadding="2" cellspacing="0">
            <tr>
                <th>Item</th>
                <th>Quantity</th>
                <th>Unit Price</th>
                <th>Extended Price</th>
            </tr>
            <xsl:for-each select="invoice/item">
                <tr>
                    <td>
                        <xsl:value-of select="@item_num"/>
                    </td>
                    <td align="right">
                        <xsl:value-of select="quantity"/>
                    </td>
                    <td align="right">
                        <xsl:value-of select="format-number(unit_price,"/>
                    </td>
                    <td align="right">
                        <xsl:value-of select="format-number(quantity * unit_price,"/>
                    </td>
                </tr>
            </xsl:for-each>
            <tr>
                <th colspan="3">Total, All Items</th>
                <td align="right">
                    <xsl:value-of select="format-number(sum(msxsl:node-set($ext_price),"/>
                </td>
            </tr>
        </table>
    </xsl:template>
</xsl:stylesheet>
### Formatted Output

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Unit Price</th>
<th>Extended Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX43598</td>
<td>29</td>
<td>2.00</td>
<td>58.00</td>
</tr>
<tr>
<td>FH29782</td>
<td>140</td>
<td>6.50</td>
<td>910.00</td>
</tr>
<tr>
<td><strong>Total, All Items</strong></td>
<td></td>
<td></td>
<td><strong>968.00</strong></td>
</tr>
</tbody>
</table>
Using Named Template Rules in XSLT

As the term implies, an XSLT named template contains a template that can be instantiated in the result tree, and this template is given a name for easy reference elsewhere in the style sheet. The named template is actually instantiated in the result tree only at the point of a corresponding `<xsl:call-template>` element in a regular template rule.

The following is a simple named template.

```
<xsl:template name="display_price">
  <xsl:value-of select="@price"/>
</xsl:template>
```

The following is a template rule that instantiates the named template.

```
<xsl:template match="/">
  <xsl:call-template name="display_price"/>
</xsl:template>
```

Note that the value of the `name` attribute of the named template is the same as the value of the `name` attribute in the `<xsl:call-template>` element.

The contents of the named template are inserted into the calling template rule, replacing the `<xsl:call-template>` element.

Named templates have two primary uses:

- As containers for frequently-used templates.
- As templates whose contents and structures depend on some variable set of conditions in the invoking template rules, using parameters.

A given named template might fulfill one purpose or the other, or both.

This section begins with the `region_qtr` example. This example provides an XML file that contains regional sales data for a fictitious company, and an XSLT file to display the data. The subsequent topics in this section add to these initial
files, resulting in a finished example at the end of the section.

This section contains the following topics.

- Initial Example for Named Templates
- Using Named Templates for Frequently Used Templates
- Passing Parameters Using Named Templates
- Finished Example for Named Templates

See Also

<xsl:template> Element | <xsl:param> Element | <xsl:with-param> Element | Using XSLT Variables and Parameters | XSLT Utilities and Samples You Can Download
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Initial Example for Named Templates

This example creates a standard simple table. The XSLT file does not yet use named templates. The subsequent topics in this section add named templates, resulting in the Finished Example for Named Templates.

Example

This example contains an XML file, and an XSLT file to transform it. The result is a simple series of headings and tables for the individual regions. This table shows data for each sales region, but not for all regions together.

XML File (region_qtr.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="region_qtr.xsl" ?>
<sales quarter="2001-01-01">
  <region>
    <name>Northeast</name>
    <manager>Kim Abercrombie</manager>
    <units>9881</units>
    <sales_amt curr">150680.89</sales_amt>
  </region>
  <region>
    <name>Southeast</name>
    <manager>Jeff D. Henshaw</manager>
    <units>4329</units>
    <sales_amt curr">67015.48</sales_amt>
  </region>
  <region>
    <name>Southwest</name>
    <manager>Sunil Koduri</manager>
    <units>3543</units>
    <sales_amt curr">57029.25</sales_amt>
  </region>
  <region>
    <name>Northwest</name>
    <manager>Brad Sutton</manager>
    <units>7569</units>
    <sales_amt curr">127423.90</sales_amt>
  </region>
</sales>
```
XSLT File (region_qtr.xsl)

<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns="http://www.w3.org/TR/REC-html40"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <html>
        <head>
            <title>Quarterly Sales (Quarter Beginning <xsl:value-of select="sales/@quarter"/>)</title>
            <style type="text/css">
                h1 {background-color: teal;
                    font-family: Tahoma,Verdana,Arial,sans-serif;
                    font-size: 24pt}
                h2 {background-color: yellow;
                    font-family: Tahoma,Verdana,Arial,sans-serif;
                    font-size: 18pt}
                h3 {background-color: white;
                    font-family: Tahoma,Verdana,Arial,sans-serif;
                    font-size: 14pt;
                    font-weight: bold}
                th {background-color: silver;
                    font-family: Tahoma,Verdana,Arial,sans-serif}
                td {background-color: white;
                    font-family: Tahoma,Verdana,Arial,sans-serif}
                .page {width: 75%}
            </style>
        </head>
        <body>
            <div class="page">
                <xsl:apply-templates/>
            </div>
        </body>
    </html>
</xsl:template>

<xsl:template match="sales">
    <h1>Quarterly Sales by Region</h1>
    <h2>Quarter Beginning <xsl:value-of select="@quarter"/></h2>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="region">
    <h3>
        <xsl:value-of select="name"/> Region
        (Manager: <xsl:value-of select="manager"/>)
    </h3>
</xsl:template>

Formatted Output
## Quarterly Sales by Region

### Quarter Beginning 2001-01-01

### Northeast Region (Manager: Kim Abercrombie)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>9,881</td>
<td>150,680.89</td>
<td>15.250</td>
</tr>
</tbody>
</table>

### Southeast Region (Manager: Jeff D. Henshaw)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,329</td>
<td>67,015.48</td>
<td>15.481</td>
</tr>
</tbody>
</table>

### Southwest Region (Manager: Sunil Koduri)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,543</td>
<td>57,029.25</td>
<td>16.096</td>
</tr>
</tbody>
</table>

### Northwest Region (Manager: Brad Sutton)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,569</td>
<td>127,423.90</td>
<td>16.835</td>
</tr>
</tbody>
</table>
Using Named Templates for Frequently Used Templates

This topic demonstrates the use of a named template as a boilerplate (that is, a frequently used template that is designed to be reusable).

We will enhance the region_qtr.xsl style sheet in Initial Example for Named Templates to display company-wide results, as well as results for the individual regions. The display of the overall results will be similar to the display of individual results: a level-3 heading, followed by a table. The final result is region_qtr_param.xsl.

The following is the relevant portion of the initial region_qtr.xsl style sheet, which displays data for each sales region, but not for all regions together.

```xml
<xsl:template match="sales">
  <h1>Quarterly Sales by Region</h1>
  <h2>Quarter Beginning <xsl:value-of select="@quarter"/></h2>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="region">
  <h3>
    <xsl:value-of select="name"/> Region
    (Manager: <xsl:value-of select="manager"/>)
  </h3>
  <table width="100%">
    <tr>
      <th width="33%" align="right">#
        Units</th>
      <th width="33%" align="right">Qtr Sales Amount</th>
      <th align="right">Amt/Unit</th>
    </tr>
    <tr>
      <td width="33%" align="right">
        <xsl:value-of select="format-number(units, '"#,##0')"
      </td>
      <td width="33%" align="right">
        <xsl:value-of select="format-number(sales_amt, '"#,##0.00')"
      </td>
      <td align="right">
        <xsl:value-of select="format-number(sales_amt div units, '"#,##0.000')"
      </td>
    </tr>
  </table>
</xsl:template>
```
The `<xsl:template match="sales">` template rule contains an `<xsl:apply-templates>` statement, which invokes `<xsl:template match="region">` to select and process the four `<region>` elements in `region_qtr.xml`.

First, add the company-wide table to the template rule for the `<sales>` element. This information should follow the `<apply-templates>` element, like this (the changes are in bold):

```xml
<xsl:template match="sales">
  <h1>Quarterly Sales by Region</h1>
  <h2>Quarter Beginning <xsl:value-of select="@quarter"/></h2>
  <xsl:apply-templates/>
  <h3>Company-Wide Results</h3>
  <table width="100%">
    <tr>
      <th width="33%"># Units</th>
      <th width="33%">Qtr Sales Amount</th>
      <th>Amt/Unit</th>
    </tr>
    <tr>
      [cells for company-wide data]
    </tr>
  </table>
</xsl:template>
```

Note that the structure of the first company-wide table row is identical to the corresponding row for the regional tables. This is an ideal situation for a reusable named template—that is, for a portion of a template, or a complete template, that you want to re-use multiple times in your style sheet.

To create such a named template, copy the reusable portion into an `<xsl:template>` rule with a name attribute, as follows:

```xml
<xsl:template name="table_hdgs">
  <tr>
    <th width="33%"># Units</th>
    <th width="33%">Qtr Sales Amount</th>
    <th>Amt/Unit</th>
  </tr>
</xsl:template>
```
Note The portion to be extracted to a named template must be well-formed. In this case, even though the identical portions of the two template rules include the <table> start tag and the <tr> start tag that follows the first row, the following would not be correct:

```xml
<xsl:template name="common_table"> <table>
<tr>
<th width="33%"># Units</th>
<th width="33%">Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
<tr>
</table>
</xsl:template>
```

This named template would be rejected by the XSLT processor, because it is not well-formed.

The second step is to replace, in the style sheet, each occurrence of the extracted code with an <xsl:call-template> element. The name attribute of this element must match the name attribute of the named template itself.

When the second step is complete, the template rules for the <sales> and <region> elements look like this:

```xml
<xsl:template match="sales">
  <h1>Quarterly Sales by Region</h1>
  <h2>Quarter Beginning <xsl:value-of select="@quarter"/></h2>
  <xsl:apply-templates/>
  <h3>Company-Wide Results</h3>
  <table width="100%">
    <xsl:call-template name="common_table"/>
    <tr>
      [cells for company-wide data]
    </tr>
  </table>
</xsl:template>

<xsl:template match="region">
  <h3>
    Region (Manager: <xsl:value-of select="manager"/>)
  </h3>
  <table width="100%">
    <xsl:call-template name="common_table"/>
    <tr>
      <td width="33%" align="right">
```
This makes each template rule shorter and easier to understand. It also simplifies maintenance. If you want to make a change, such as adding a new column or changing the formatting, the change only has to be made once. This change is made to the named template, instead of once for every occurrence of the boilerplate template.

The next topic, Passing Parameters Using Named Templates, continues to build this example.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Passing Parameters Using Named Templates

The following might be a first attempt at a template rule to produce a company-wide quarterly sales report.

```xml
<xsl:template match="sales">
  <h1>Quarterly Sales by Region</h1>
  <h2>Quarter Beginning <xsl:value-of select="@quarter"/></h2>
  <xsl:apply-templates/>
  <h3>Company-Wide Results</h3>
  <table width="100%">
    <xsl:call-template name="common_table"/>
    <tr>
      <td width="33%" align="right">
        <xsl:value-of select="format-number(sum(region/units), '#,##0')"/>
      </td>
      <td width="33%" align="right">
        <xsl:value-of select="format-number(sum(region/sales_amt), '#,##0.00')"/>
      </td>
      <td align="right">
        <xsl:value-of select="format-number(sum(region/sales_amt) div sum(region/units), '#,##0.000')"/>
      </td>
    </tr>
  </table>
</xsl:template>
```

Notice that this looks very similar to a template rule for an individual sales region. The only differences between this template rule and the one for the `<region>` element are the data values being plugged into the table. Even the surrounding `<table>` element is the same, and the level-3 headings that precede the tables differ only in their contents.

Fortunately, XSLT supplies parameters. Parameters enable you to pass values into a named template to be inserted as needed into a common format.

Parameters are passed from the calling template rule to the named template by `<xsl:with-param>` elements in the template rule, and by `<xsl:param>` elements in the named template. Each `<xsl:with-param>` element is a child of the `<xsl:call-template>` element. Each `<xsl:param>` element is a child of the
named template's <xsl:template> element.

For example, a named template to be passed a parameter might look like this:

```xml
<xsl:template name="param_template">
  <xsl:param name="passed_param"/>
  <h1><xsl:value-of select="$passed_param"/></h1>
</xsl:template>
```

The following might be a template rule to pass data to this named template.

```xml
<xsl:template match="/">
  <xsl:call-template name="param_template">
    <xsl:with-param name="passed_param">
      Our Company Name
    </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

This template rule passes the string "Our Company Name" to the param_template named template by way of the passed_param parameter. The passed value is then inserted into an <h1> element by the named template, and instantiated in the result tree of the template rule.

The <xsl:with-param> and <xsl:param> elements also have an optional select attribute. In the calling template rule, the value of this attribute is an XPath expression relative to the context node at that point. For more information about context nodes, see Determining the Context Node. In the named template, the select attribute of the <xsl:param> element supplies a default value if a calling template rule fails to provide one.

Therefore, we can now change our named template as follows. In particular, note the use of passed parameters substituted into the common table structure.

```xml
<xsl:template name="common_table">
  <xsl:param name="table_hdg" select="'Default Heading'"/>
  <xsl:param name="units_param" select="'Default Units'"/>
  <xsl:param name="amt_param" select="'Default Amount'"/>
  <h3><xsl:value-of select="$table_hdg"/></h3>
  <table width="100%">
    <tr>
      <th width="33%"># Units</th>
      <th width="33%">Qtr Sales Amount</th>
      <th>Amt/Unit</th>
    </tr>
  </table>
</xsl:template>
```
The template rules for the <sales> and <region> elements are now shorter, as follows:

<xsl:template match="sales">
  <h1>Quarterly Sales by Region</h1>
  <h2>Quarter Beginning <xsl:value-of select="@quarter"/></h2>
  <xsl:apply-templates/>
  <xsl:call-template name="common_table">
    <xsl:with-param name="table_hdg" select="'Company-Wide Results'"/>
    <xsl:with-param name="units_param" select="sum(region/units)"/>
    <xsl:with-param name="amt_param" select="sum(region/sales_amt)"/>
  </xsl:call-template>
</xsl:template>

<xsl:template match="region">
  <xsl:call-template name="common_table">
    <xsl:with-param name="table_hdg">
      Region
      (Manager: <xsl:value-of select="manager"/>)
    </xsl:with-param>
    <xsl:with-param name="units_param" select="units"/>
    <xsl:with-param name="amt_param" select="sales_amt"/>
  </xsl:call-template>
</xsl:template>

Note  When passing string values to named templates, be sure to enclose the string value in internal quotation marks. For example, in the select attribute of the <xsl:with-param name="table_hdg" select="'Company-Wide Results'"/> code in the <sales> template rule above, "Company-Wide Results" (without internal quotation marks) would be incorrect. If you omit the internal quotation marks, the XPath expression is evaluated as if it refers to a child of the context node at that point.
Although this example simply passes parameters for display, note that you can also use parameters to control optional processing in the named template. For example, we could use parameters to insert some extra spacing and a horizontal rule into the named template, to be displayed immediately before the company-wide results. The following code does this with an `<xsl:if>` element with a $table_hdg parameter.

```xml
<xsl:if test="$table_hdg='Company-Wide Results'">
  <p />
  <hr />
</xsl:if>
```

The resulting finished example is shown in the next topic, Finished Example for Named Templates.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Finished Example for Named Templates

This finished example is based on the Initial Example for Named Templates. The other topics in this section have added named templates to this initial example.

Example

The formatted output at the end of this topic shows the completed company-wide results, as well as some of the regional results.

XML File (region_qtr.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="region_qtr_param.xsl" ?>
<sales quarter="2001-01-01">
  <region>
    <name>Northeast</name>
    <manager>Kim Abercrombie</manager>
    <units>9881</units>
    <sales_amt curr="">150680.89</sales_amt>
  </region>
  <region>
    <name>Southeast</name>
    <manager>Jeff D. Henshaw</manager>
    <units>4329</units>
    <sales_amt curr="">67015.48</sales_amt>
  </region>
  <region>
    <name>Southwest</name>
    <manager>Sunil Koduri</manager>
    <units>3543</units>
    <sales_amt curr="">57029.25</sales_amt>
  </region>
  <region>
    <name>Northwest</name>
    <manager>Brad Sutton</manager>
    <units>7569</units>
    <sales_amt curr="">127423.90</sales_amt>
  </region>
</sales>
```
This file, region_qtr_param.xsl, is an expanded version of region_qtr.xsl. This is the result of the above topics.

```xml
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/REC-html40">

<!-- Template rule for document root: -->
<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>Quarterly Sales (Quarter Beginning <xsl:value-of select="sales/@quarter"/>)</TITLE>
    </HEAD>
    <BODY>
      <DIV class="page">
        <xsl:apply-templates/>
      </DIV>
    </BODY>
  </HTML>

<!-- Named template for constructing sales results headings and tables: -->
<xsl:template name="common_table">
  <xsl:param name="table_hdg" select="'Default Heading'"/>
  <xsl:param name="units_param" select="'Default Units'"/>
  <xsl:param name="amt_param" select="'Default Amount'"/>
</xsl:template>
```
<xsl:if test="$table_hdg='Company-Wide Results'">
  <p/>
  <hr/>
</xsl:if>

<h3><xsl:value-of select="$table_hdg"/></h3>
<table width="100%">
  <tr>
    <th width="33%"># Units</th>
    <th width="33%">Qtr Sales Amount</th>
    <th>Amt/Unit</th>
  </tr>
  <tr>
    <td width="33%" align="right">
      <xsl:value-of select="format-number($units_param, '#,##0')"/>
    </td>
    <td width="33%" align="right">
      <xsl:value-of select="format-number($amt_param, '#,##0.0')"/>
    </td>
    <td align="right">
      <xsl:value-of select="format-number($amt_param div $units_param, '#,##0.000')"/>
    </td>
  </tr>
</table>
Formatted Output

The following is a portion of the result.

## Quarterly Sales by Region

### Quarter Beginning 2001-01-01

#### Northeast Region (Manager: Kim Abercrombie)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>9,881</td>
<td>150,680.89</td>
<td>15.250</td>
</tr>
</tbody>
</table>

#### Southeast Region (Manager: Jeff D. Henshaw)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,329</td>
<td>67,015.48</td>
<td>15.481</td>
</tr>
</tbody>
</table>

#### Southwest Region (Manager: Sunil Koduri)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,543</td>
<td>57,029.25</td>
<td>16.096</td>
</tr>
</tbody>
</table>

#### Northwest Region (Manager: Brad Sutton)

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,569</td>
<td>127,423.90</td>
<td>16.835</td>
</tr>
</tbody>
</table>

---

### Company-Wide Results

<table>
<thead>
<tr>
<th># Units</th>
<th>Qtr Sales Amount</th>
<th>Amt/Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>25,322</td>
<td>402,149.52</td>
<td>15.881</td>
</tr>
</tbody>
</table>
Inserting Automatic Numbering Using XSLT

XSLT provides several ways for you to number portions of your output, for example in a numbered list of items. This section illustrates numbering in XSLT. We will use a two-player, two-hand card game as an example, which consists of an XML file, cardgame.xml, and an XSLT style sheet that displays the card game as an HTML document.

This section contains the following topics.

- Inserting Automatic Numbering Using position()
- Inserting Automatic Numbering Using <xsl:number>

See Also

position Function | <xsl:number> Element | <xsl:decimal-format> Element | format-number Function | Context and Navigation in XPath
Inserting Automatic Numbering Using position()

This topic contains the initial sample files for this section, cardgame.xml and cardgame_orig.xsl. The XML file represents the playing cards dealt in a card game with two players. Each player receives five cards at first, and in the second hand of the game they can exchange up to three cards for new ones.

Two hands are dealt. During the first hand, Jack receives the Queen of Clubs, Ace of Spades, and so on, while Jill receives the 5 of Diamonds, 5 of Spades, and so on. In the second hand, Jack trades in three cards (look at the replace attributes to see which ones), and Jill only one.

The hand each player is dealt includes one or more <tell> elements. (A "tell" in card games is a player's reaction—often unconscious—to the cards they have received.)

Note the use of the position() function. This XPath function returns the number of the current node within whatever node-set is being processed. Because the node-set established by the template rule (match="hand") contains all <hand> elements in the document, the output will be numbered from 1 to 2, in document order.

Example

XML File (cardgame.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="cardgame_orig.xsl"?>
<cardgame>
  <hand>
    <player name="Jack">
      <card dealtID="card01">
        <rank>Q</rank><suit>Clubs</suit>
      </card>
      <card dealtID="card02">
        <rank>A</rank><suit>Spades</suit>
      </card>
    </player>
    <player name="Jill">
      <card dealtID="card03">
        <rank>2</rank><suit>Clubs</suit>
      </card>
      <card dealtID="card04">
        <rank>4</rank><suit>Spades</suit>
      </card>
      <card dealtID="card05">
        <rank>5</rank><suit>Spades</suit>
      </card>
      <card dealtID="card06">
        <rank>6</rank><suit>Clubs</suit>
      </card>
      <card dealtID="card07">
        <rank>7</rank><suit>Diamonds</suit>
      </card>
    </player>
  </hand>
</cardgame>
```
<card dealtID="card03">
  <rank>10</rank><suit>Hearts</suit>
</card>
twitches nervously</tell>
<card dealtID="card04">
  <rank>8</rank><suit>Clubs</suit>
</card>
<card dealtID="card05">
  <rank>4</rank><suit>Spades</suit>
</card>
squints</tell>
</player>

<player name="Jill">
<card dealtID="card06">
  <rank>5</rank><suit>Diamonds</suit>
</card>
<card dealtID="card07">
  <rank>5</rank><suit>Spades</suit>
</card>
<card dealtID="card08">
  <rank>6</rank><suit>Clubs</suit>
</card>
<card dealtID="card09">
  <rank>6</rank><suit>Spades</suit>
</card>
<card dealtID="card10">
  <rank>10</rank><suit>Diamonds</suit>
</card>
grins</tell>
</player>

</hand>

<player name="Jack">
<card dealtID="card11" replace="card03">
  <rank>4</rank><suit>Diamonds</suit>
</card>
mutters, "Uh-oh..."</tell>
<card dealtID="card12" replace="card04">
  <rank>K</rank><suit>Spades</suit>
</card>
<card dealtID="card13" replace="card05">
  <rank>7</rank><suit>Spades</suit>
</card>
tugs at left ear</tell>
</player>

<player name="Jill">
<card dealtID="card14" replace="card10">
  <rank>K</rank><suit>Diamonds</suit>
</card>
sips her drink</tell>
XSLT File (cardgame_orig.xsl)

The first version of the XSLT style sheet simply numbers the hands, using a template rule for the <hand> element:

```xml
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns="http://www.w3.org/TR/REC-html40"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Set up the HTML page and table styles. -->
<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>Card Game</TITLE>
      <STYLE type="text/css">
        h1 {background-color: gold;
            font-family: Tahoma,Verdana,Arial,sans-serif;
            font-size: 24pt}
        h2 {background-color: yellow;
            font-family: Tahoma,Verdana,Arial,sans-serif;
            font-size: 18pt}
        h3 {background-color: white;
            font-family: Tahoma,Verdana,Arial,sans-serif;
            font-size: 14pt;
            font-weight: bold}
        th {background-color: silver;
            font-family: Tahoma,Verdana,Arial,sans-serif}
        td {background-color: white;
            font-family: Tahoma,Verdana,Arial,sans-serif}
        .page {width: 75%}
        .tell {font-weight: bold}
      </STYLE>
    </HEAD>
    <BODY>
      <DIV class="page">
        <xsl:apply-templates/>
      </DIV>
    </BODY>
  </HTML>
</xsl:template>

<!-- Suppress <tell> elements. -->
<xsl:template match="tell"/>
```
Formatted Output

Hand #1
QClubsASpades10Hearts8Clubs4Spades5Diamonds5Spades6Clubs6Spades10Diamonds

Hand #2
4DiamondsKSpades7SpadesKDiamonds
Inserting Automatic Numbering Using `<xsl:number>`

The `<xsl:number>` element is useful for generating hierarchical strings of numbers that indicate the depth to which a particular portion of a document is nested. For example, a hypothetical document outline might appear as follows:

I. Major Heading #1

   I.A. First sub-heading under Major Heading #1

      I.A.1. ...

      I.A.2. ...

   I.B. Second sub-heading under Major Heading #1

II. Major heading #2

   ...

The `cardgame.xml` file in Inserting Automatic Numbering Using `position()` is a hierarchy that lends itself to this kind of numbering. Each hand dealt is a level. Each hand level contains a level representing each player. For each player, there up to five cards dealt. The hands themselves have already been numbered, using the `position()` function. Now we'll use `<xsl:number>` to number the players and their cards.

The following is the general form of the start tag for an `<xsl:number>` element.

```xml
<xsl:number
   level="depthofnumbering"
   count="pattern"
   from="pattern"
   value="expression"
   format="formatspec"
   lang="langcode"
   letter-value="alphaortrad"
```
All of the attributes are optional. The ones we'll be concerned with in this section (and the ones you will probably use most often) are level, count, and format.

Without any attributes at all, `<xsl:number>` behaves the same way as the `position()` function does, as demonstrated earlier.

Now we'll add the following template rule to our existing style sheet, `cardgame_orig.xsl`. This style sheet is available in Inserting Automatic Numbering Using `position()`.

```xml
<xsl:template match="player">
  <h2>
    Player <xsl:number/>:
    <xsl:value-of select="@name"/>
  </h2>
  <xsl:apply-templates/>
</xsl:template>
```

The new template rule displays the each player's position within each hand, as in the level-2 headings shown here:
Suppose, however, that we want to number the players more carefully, using a combination of the hand number and the position of the <player> element within that hand. We can use <xsl:number> with level="multiple", level="single", or level="any".

This section continues with the following topics:

- Using level="multiple" for Numbering
- Using level="single" for Numbering
- Using level="any" for Numbering
- Example of Automatic Numbering with <xsl:number>
Using level="multiple" for Numbering

In this topic, we add the level, format, and count attributes to the <xsl:number> element of our existing style sheet, cardgame_orig.xsl. The original file is in Inserting Automatic Numbering Using position(). The following snippet shows the new attributes and their values in bold.

```xml
<xsl:template match="player">
    <h2>
        Player
        <xsl:number level="multiple" format="1.1" count="hand | player"/>
        <xsl:value-of select="@name"/>
    </h2>
    <xsl:apply-templates/>
</xsl:template>
```

The level attribute with the value "multiple" indicates that this will be a hierarchical number. The format attribute indicates that the hierarchy will be represented as a string of period-delimited integers (such as 1.1 or 1.2, and so on). The count attribute includes a list of elements along the ancestor-or-self:: axis, separated by a pipe (|). These elements are to be used in determining the number to be displayed.

Output from this version of the style sheet now appears as follows:
Note that the player numbers now include the hand number, in the following format.

`handnumber,playernumber`

The next step is to add the `<xsl:template match="card">` template rule to process the individual cards dealt in each hand. Again, we'll use `<xsl:number>` to number them according to their sequence within each hand dealt, and within each `<player>` element to which the `<card>` elements belong. This time, however, we'll change the formatting card numbers within each hand/player hierarchy, so that each card will be represented with two digits instead of one. For example, the following card number would represent the fourth card dealt to player #1 in hand #2.

`2.1.04`

We'll also add some code to generate a `<table>` element and table headings for each group of cards. Our two template rules from `cardgame_orig.xsl` now look like this:
<xsl:template match="player">
  <h2>
    Player
    <xsl:number level="multiple" format="1.1" count="hand | player"/>
    :<xsl:value-of select="@name"/>
  </h2>
</xsl:template>

<table width="75%">
  <tr>
    <th>Card #</th>
    <th>Rank/Suit</th>
    <th>Replaces</th>
  </tr>
  <xsl:apply-templates/>
</table>

<xsl:template match="card">
  <tr>
    <td align="center" width="30%">
      <xsl:number level="multiple" format="1.1.01" count="hand | player | card"/>
    </td>
    <td width="40%">
      <xsl:value-of select="rank"/>
      <xsl:text> of </xsl:text>
      <xsl:value-of select="suit"/>
    </td>
    <td>
      <xsl:choose>
        <xsl:when test="@replace">
          <xsl:value-of select="preceding::card[@dealtID=current()/@replace]/rank"/>
          <xsl:value-of select="preceding::card[@dealtID=current()/@replace]/suit"/>
        </xsl:when>
        <xsl:otherwise>--</xsl:otherwise>
      </xsl:choose>
    </td>
  </tr>
</xsl:template>

Note  The values inserted into the last column (headed "Replaces") use a complex pair of XPath expressions. For each <card> element, these expressions obtain the rank and suit of the preceding <card> element whose dealtID attribute matches the replace attribute of the current <card>
element. If there is no such preceding \texttt{<card>}, this table cell contains a pair of hyphens (--).

A portion of the results when viewed in Internet Explorer looks like this:

**Player 1.2: Jill**

<table>
<thead>
<tr>
<th>Card #</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.01</td>
<td>5 of Diamonds</td>
<td>--</td>
</tr>
<tr>
<td>1.2.02</td>
<td>5 of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.2.03</td>
<td>6 of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.2.04</td>
<td>6 of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.2.05</td>
<td>10 of Diamonds</td>
<td>--</td>
</tr>
</tbody>
</table>

**Hand #2**

**Player 2.1: Jack**

<table>
<thead>
<tr>
<th>Card #</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.01</td>
<td>4 of Diamonds</td>
<td>10 of Hearts</td>
</tr>
<tr>
<td>2.1.02</td>
<td>K of Spades</td>
<td>3 of Clubs</td>
</tr>
<tr>
<td>2.1.03</td>
<td>7 of Spades</td>
<td>4 of Spades</td>
</tr>
</tbody>
</table>

**Player 2.2: Jill**
Using level="single" for Numbering

The <xsl:number> examples in the previous topic use the level attribute with a value of "multiple" to assign a series of hierarchical number strings such as 1.1.01, 1.1.02, and so on.

The level attribute can also take the values "single" and "any". These are used to sequentially number elements, either at the same level in the document tree, or across all levels, respectively.

This topic demonstrates the effects of these two values by creating a non-empty template rule for the <tell> elements in the sample XML document, cardgame.xml. This file is in Inserting Automatic Numbering Using position().

The following is the template rule with level="single".

```xml
<xsl:template match="tell">
    <tr>
        <td colspan="3" align="center" class="tell">
            (Reaction #<xsl:number level="single" count="tell"/>: \\
             <xsl:value-of select="../@name"/>)
        </td>
    </tr>
</xsl:template>
```

The value "single" tells the XSLT processor to generate a sequential number that cycles through the elements whose names match the value of the count attribute.

The formatted output below shows the results. Note the numbering assigned to the reactions of Jack and Jill as the cards are dealt.
Hand #1

Player 1.1: Jack

<table>
<thead>
<tr>
<th>Card #</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.01</td>
<td>Q of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.1.02</td>
<td>A of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.1.03</td>
<td>10 of Hearts</td>
<td>--</td>
</tr>
</tbody>
</table>

(Reaction #1: Jack twitches nervously)

<table>
<thead>
<tr>
<th>Card #</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.04</td>
<td>8 of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.1.05</td>
<td>4 of Spades</td>
<td>--</td>
</tr>
</tbody>
</table>

(Reaction #2: Jack squints)

Player 1.2: Jill

<table>
<thead>
<tr>
<th>Card #</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.01</td>
<td>5 of Diamonds</td>
<td>--</td>
</tr>
<tr>
<td>1.2.02</td>
<td>5 of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.2.03</td>
<td>6 of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.2.04</td>
<td>6 of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.2.05</td>
<td>10 of Diamonds</td>
<td>--</td>
</tr>
</tbody>
</table>

(Reaction #1: Jill grins)

The numbering generated from the `<te11>` elements begins again with each player, so Jack's reactions are numbered 1 and 2, and Jill's reactions are numbered starting at 1 again.
Using level="any" for Numbering

This topic uses the level attribute with the value "any". The following template rule is based on the initial cardgame example, in Inserting Automatic Numbering Using position().

```xml
<xsl:template match="tell">
  <tr>
    <td colspan="3" align="center" class="tell">
      (Reaction #<xsl:number level="any" count="tell"/>: <xsl:value-of select="/../@name"/> <xsl:text/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="."/>)
    </td>
  </tr>
</xsl:template>
```

The value "any" instructs the XSLT processor to number the indicated element sequentially, no matter what level it is at in the document hierarchy. Jack's reactions during the first hand are still numbered 1 and 2, while Jill's first reaction is now #3 in the game as a whole. The results are shown in Example of Automatic Numbering with <xsl:number>.
Example of Automatic Numbering with <xsl:number> 

Example

This is the final example produced by the topics about automatic numbering with <xsl:number>.

XML File (cardgame.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="cardgame_xslnum.xsl"?>
<cardgame>
  <hand>
    <player name="Jack">
      <card dealtID="card01">
        <rank>Q</rank><suit>Clubs</suit>
      </card>
      <card dealtID="card02">
        <rank>A</rank><suit>Spades</suit>
      </card>
      <card dealtID="card03">
        <rank>10</rank><suit>Hearts</suit>
      </card>
      <tell>twitches nervously</tell>
      <card dealtID="card04">
        <rank>8</rank><suit>Clubs</suit>
      </card>
      <card dealtID="card05">
        <rank>4</rank><suit>Spades</suit>
      </card>
      <tell>squints</tell>
    </player>
    <player name="Jill">
      <card dealtID="card06">
        <rank>5</rank><suit>Diamonds</suit>
      </card>
      <card dealtID="card07">
        <rank>5</rank><suit>Spades</suit>
      </card>
      <card dealtID="card08">
        <rank>6</rank><suit>Clubs</suit>
      </card>
    </player>
  </hand>
</cardgame>
```
XSLT File (cardgame_xslnum.xsl)

This is the final version of the XSLT file for the topics about automatic numbering with <xsl:number>.

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/TR/REC-html40">

<!-- Template rule for the document root: -->
<xsl:template match="/"
 <HTML>
 <HEAD>
 <TITLE>Card Game</TITLE>
**Hand #1**

**Player 1.1: Jack**

<table>
<thead>
<tr>
<th>Hand.Player.Card</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.01</td>
<td>Q of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.1.02</td>
<td>A of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.1.03</td>
<td>10 of Hearts</td>
<td>--</td>
</tr>
</tbody>
</table>

(Reaction #1: Jack twitches nervously)

<table>
<thead>
<tr>
<th>Hand.Player.Card</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.04</td>
<td>8 of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.1.05</td>
<td>4 of Spades</td>
<td>--</td>
</tr>
</tbody>
</table>

(Reaction #2: Jack squints)

**Player 1.2: Jill**

<table>
<thead>
<tr>
<th>Hand.Player.Card</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.01</td>
<td>5 of Diamonds</td>
<td>--</td>
</tr>
<tr>
<td>1.2.02</td>
<td>5 of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.2.03</td>
<td>6 of Clubs</td>
<td>--</td>
</tr>
<tr>
<td>1.2.04</td>
<td>6 of Spades</td>
<td>--</td>
</tr>
<tr>
<td>1.2.05</td>
<td>10 of Diamonds</td>
<td>--</td>
</tr>
</tbody>
</table>

(Reaction #3: Jill grins)
Hand #2

Player 2.1: Jack

<table>
<thead>
<tr>
<th>Hand.Player.Card</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.01</td>
<td>4 of</td>
<td>10 of</td>
</tr>
<tr>
<td></td>
<td>Diamonds</td>
<td>Hearts</td>
</tr>
</tbody>
</table>

(Reaction #4: Jack mutters, "Uh-oh..."

| 2.1.02           | K of Spades| 8 of Clubs|
| 2.1.03           | 7 of Spades| 4 of Spades|

(Reaction #5: Jack tugs at left ear)

Player 2.2: Jill

<table>
<thead>
<tr>
<th>Hand.Player.Card</th>
<th>Rank/Suit</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.01</td>
<td>K of</td>
<td>10 of</td>
</tr>
<tr>
<td></td>
<td>Diamonds</td>
<td>Diamonds</td>
</tr>
</tbody>
</table>

(Reaction #6: Jill sips her drink)
Importing and Including Style Sheets

XSLT provides the `<xsl:import>` and `<xsl:include>` elements for incorporating the content of one style sheet within another.

This section contains the following topics.

- Initial Example for Importing Style Sheets
- Using `<xsl:include>` to Insert an XSLT File
- Using `<xsl:import>` to Insert an XSLT File with Lower Precedence
- Deciding Whether to Import or Include an XSLT File
- Avoiding Conflicting Rules by Using `<xsl:apply-Imports>`

Note Using external parsed entities with DTDs

Because XSLT style sheets are XML documents themselves, you can also use XML's own simple facility for incorporating one XML document within another. This facility, the use of external parsed entities, is discussed in the XML Developer's Guide. The advantage of using such entities is that they are not constrained by any of the XSLT-specific rules governing the use of the `<xsl:include>` and `<xsl:import>` elements. They are only constrained by the rules of well-formed XML documents. The drawback of using such entities is that they require a Document Type Definition (DTD), which is not always practical or even possible.
Initial Example for Importing Style Sheets

Scootney Publishing has various lines of books, which cover a broad range of subjects: general fiction; literary fiction; genre fiction (such as science fiction and romance); software and other technical reference books; biography; and home-repair. One of the company's most successful ventures has been placing the first chapter or two of each of its books on the company's Web site, to encourage actual book sales.

The content of each book is represented as XML, and each line of books has its own XSLT style sheet. This allows literary novels to be presented one way, reference books another, and so on. However, Scootney's latest title—a novel, *Synchronized Jamming*—is a weird amalgam of fictional, technical, and other content, marked up using combined elements of several editorial lines.

Example

XML File (jamming.xml)

Here's a brief excerpt from the first chapter of *Synchronized Jamming*. The book's structure is a mix of common elements of fiction, literal HTML code, and musical annotation.

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="jamming.xsl"?>
<novel copyright="2001" isbn="555000000X">
  <title>Synchronized Jamming</title>
  <author>Kari Hensien</author>
  <chapter number="1">
    <title>The Bleeding Edge</title>
    <epigraph><text>Any sufficiently advanced technology is indistinguishable from noise.</text><author>Canopy Rose</author><source>The Fusion Manifesto</source></epigraph>
    <para>Politics. Technology. Rock'n'roll. All you needed to know about the underground band known as Canopy Rose was right there in the first couple grafts of their so-called "Fusion Manifesto":</para>
    <markup_code><![CDATA[
```
Listen. You know as well as we do...
... the sound of one hand jammin':

Welcome to the block party. Welcome to the revolution.

XSLT File (jamming.xsl)

The Scootney production department developed the initial version of the XSLT style sheet for this file to process all the fiction-related markup in *Synchronized Jamming*. It was derived from their standard *fiction.xsl* style sheet for converting novel excerpts to HTML.

This initial *jamming.xsl* style sheet provides template rules for all the common fictional content encountered in this excerpt. Other such content types will be added to the style sheet in subsequent topics in this section.

```xml
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/REC-html40">
  <xsl:template match="novel">
    <HTML>
      <HEAD>
        <TITLE>Excerpt: 
          "<xsl:value-of select="title"/>",
        by 
        <xsl:value-of select="author"/>
        </TITLE>
        <STYLE>
          * {font-family: Arial, Helvetica, Univers, sans-serif}
          pre {font-family: Courier New, Courier, monospace}
        </STYLE>
      </HEAD>
      <BODY>
```
Formatted Output

The initial XSLT file displays the content of the fiction-related elements. It does not, however, display content from computer-related and music-related elements.

Excerpt:

**Synchronized Jamming**

*by Kari Hensien*

Copyright 2001 by Kari Hensien

Scoorney Publishing / ISBN 55555555555

---

**Chapter 1:**

**The Bleeding Edge**

Politics. Technology. Rock'n'roll. All you needed to know about the underground band known as Canopy Rose was right there in the first couple graf's of their so-called "Fusion Manifesto":

Any sufficiently advanced technology is indistinguishable from noise.

-- Canopy Rose (The Fusion Manifesto)

<html> <head><title>Fused</title></head> <body> <p>Listen. You know as well as we do...<p> <p>... the sound of one hand jammin':</p> <p>AC</p> <p>Welcome to the block party. Welcome to the revolution.</p> </body> </html>

The following topic, Using `<xsl:include>` to Insert an XSLT File, alters the XSLT file to make it more appropriate for the unusual content of the XML file.
Using `<xsl:include>` to Insert an XSLT File

Many programming languages allow you to logically include the contents of one program (or portion of a program) within the scope of another, without doing so physically. For example, in the C programming language, you can use the `#include` directive for this purpose. Web sites developed using Microsoft FrontPage can take advantage of a similar feature, the Include Page component, which causes HTML code from one document to be embedded within another.

The XSLT counterpart of this feature is the `<xsl:include>` element. This is a top-level element within the style sheet—that is, it must be a child of the style sheet's root `<xsl:stylesheet>` element. The syntax is as follows:

```
<xsl:include href="uri"/>
```

In this syntax, `uri` is the Uniform Resource Identifier (URI) of the style sheet to be embedded at that point.

The referenced style sheet must be a valid style sheet itself. That is, it must include a root `<xsl:stylesheet>` element. With the exception of the start and end tags, all the root element's content is logically inserted into the including style sheet. The effect is the same as if the template rules, named templates, variable declarations, and so on were physically included at that point.

Although `<xsl:include>` must be a top-level element, it can be placed anywhere among the style sheet's other top-level elements.

The examples in this topic work with `jamming.xml` and `jamming.xsl`, in Initial Example for Importing Style Sheets.

The `markup_code.xsl` file (shown later in this topic) is included into the main XSLT file, `jamming.xsl` with the help of `<xsl:include>` To incorporate the template rule for processing the `<markup_code>` elements in `jamming.xml`, the production department of the fiction group added an `<xsl:include>` element to `jamming.xsl`. 
This saves a few lines of code, and ensures that the <markup_code> elements in *Synchronized Jamming* are treated the same as those in a real computer reference text. A few more lines of code are saved by the fact that Scootney's "ScootMusic" line of books had already established a style sheet for <chord> elements, *chord.xsl*. This style sheet is shown later in this topic as an imported XSLT file.

A reference to *chord.xsl* was added to *jamming.xsl*, as follows:

```xml
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/REC-html40">
  <xsl:include href="markup_code.xsl"/>
  <xsl:include href="chord.xsl"/>
  <xsl:template match="novel">
    ...
  </xsl:template>
</xsl:stylesheet>
```

As a result, the first chapter of *Synchronized Jamming* is displayed in a manner consistent with all three publishing lines, as shown in the formatted output at the end of this topic.

To simplify *jamming.xsl*, its designers could have included not only *markup_code.xsl* and *chord.xsl*, but the entire *fiction.xsl* style sheet from which *jamming.xsl* is otherwise derived—and then removed all of the specific template rules which *jamming.xsl* provides. The resulting style sheet would look as follows:

**Alternative XSLT File (jamming_importonly.xsl)**

**Note**  This file is not used in the full example.
Example

This example builds on the example from Initial Example for Importing Style Sheets.

While designing the style sheet for Synchronized Jamming, the fiction branch of Scootney Publishing discovered that a style sheet, markup_code.xsl, had already been established in the computer-reference division for the <markup_code> element. They also learned that the computer-reference division had broken out all their separate styles into separate style sheet modules, such as chord.xsl. These modules could be included in other style sheets as needed.

XML File (jamming.xml)

Use jamming.xml, in Initial Example for Importing Style Sheets. Change the href attribute to reference jamming2.xsl.

Included XSLT File (markup_code.xsl)

<xlsl:stylesheet version="1.0"
   xmlns:xlsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/REC-html40">

  <xlsl:template match="markup_code">
    <div style="width:75%; margin-left:20; padding:10">
      <pre><xlsl:value-of select="."/></pre>
    </div>
  </xlsl:template>
</xlsl:stylesheet>

Included or Imported XSLT File (chord.xsl)

<xlsl:stylesheet version="1.0"
   xmlns:xlsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/REC-html40">
<xsl:template match="chord">
  <div style="width:75%; margin-left:20; padding:10">
    <table border="0">
      <tr>
        <th style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Chord Notes
        </th>
      </tr>
      <tr>
        <xsl:for-each select="note">
          <td><b><xsl:value-of select="."/></b></td>
        </xsl:for-each>
      </tr>
      <tr>
        <td style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Duration:
        </td>
      </tr>
      <tr>
        <td style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Voice:
        </td>
      </tr>
      <tr>
        <td style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Vibrato:
        </td>
      </tr>
    </table>
  </div>
</xsl:template>

**XSLT File (jamming2.xsl)**

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">
  <xsl:include href="markup_code.xsl"/>
  <xsl:include href="chord.xsl"/>
</xsl:stylesheet>
```
Formatted Output
Chapter 1: The Bleeding Edge

Politics. Technology. Rock’n’roll. All you needed to know about the underground band known as Canopy Rose was right there in the first couple grafts of their so-called "Fusion Manifesto":

Any sufficiently advanced technology is indistinguishable from noise
-- Canopy Rose (The Fusion Manifesto)

<p>Listen. You know as well as we do...</p>
<p>... the sound of one hand jammin':</p>

Chord Notes
A    C
Duration: 4.5 seconds
Voice: steel
Vibrato: on

<p>Welcome to the block party. Welcome to the revolution.</p>
Using `<xsl:import>` to Insert an XSLT File with Lower Precedence

In addition to the `<xsl:include>` mechanism discussed in the previous topic, the XSLT standard provides a more advanced import feature: the `<xsl:import>` element.

The `<xsl:import>` element resembles `<xsl:include>` in many respects. Both are empty, top-level elements, and they have the same syntax:

```xml
<xsl:import href="uri"/>
```

In this syntax, `uri` is the Uniform Resource Identifier (URI) of another style sheet whose contents are to be logically included within the main style sheet.
Deciding Whether to Import or Include an XSLT File

All `<xsl:import>` elements must appear as the first top-level elements in the importing style sheet. By contrast, `<xsl:include>` elements can appear anywhere among other top-level elements in the including style sheet.

The main difference between `<xsl:import>` and `<xsl:include>` is that the template rules and other top-level elements embedded with `<xsl:import>` are processed according to the explicit rules of import precedence. This enables you to control the ways in which conflicting or overlapping styles are applied, in a predictable fashion. Any template rules imported into a style sheet have a lower import precedence than those that physically reside in that style sheet. Template rules included in a style sheet have the same import precedence as those in the including style sheet.

Consider a style sheet, `main.xsl`. Suppose you want to logically incorporate the contents of another style sheet, `sub_1.xsl`, into `main.xsl`. Furthermore, you want to incorporate `sub_2.xsl` in `sub_1.xsl`. The `main.xsl` style sheet has a template rule for processing an `<a>` element. The `sub_1.xsl` style sheet has a template rule for processing a `<b>` element. The `sub_2.xsl` style sheet has template rules for both an `<a>` element and a `<b>` element.

Whether you use `<xsl:include>` or `<xsl:import>`, the general logical structure of the three style sheets might be represented as follows:

```
main.xsl

  sub_1.xsl:
    template rule for <b>

  sub_2.xsl:
    template rule for <a>
```
template rule for <b>

main.xsl's own template rule for <a>

If main.xsl includes both sub_1.xsl and sub_2.xsl, the <a> and <b> elements are processed according to the default rules for resolving conflicts. According to the XSLT standard, conflicting template rules are an error. Therefore, some XSLT processors might treat this situation as an error.

Alternatively, the standard allows the processor to accept the last such template rule in the style sheet, ignoring the others. Microsoft's XSLT processor (like most) follows this alternative processing route. Therefore:

- The template rule for <b> in sub_2.xsl will override that in sub_1.xsl, because the one in sub_2.xsl appears later in main.xsl.
- The template rule for <a> in main.xsl itself will override that in sub_2.xsl, because the one in main.xsl appears later.

Two rules determine the import precedence of a conflicting declaration:

1. Any conflicting declaration in an imported style sheet has a lower import precedence than one in the importing style sheet.
2. If one style sheet imports several others, the declarations of the first imported style sheet have lower precedence than those of the second imported style sheet, which have lower precedence than those of the third, and so on.

According to these rules, if the connections among the three style sheets are all <xsl:import> elements, the following statements are true:

- The template rule for <b> which appears in sub_1.xsl will override that in sub_2.xsl, because sub_1.xsl imports sub_2.xsl, and therefore the declarations in sub_1.xsl have a higher import precedence. This is the opposite of what happens if you use <xsl:include> in the same situation.
- The template rule for <a> which appears in main.xsl itself will override that in sub_2.xsl, because main.xsl imports sub_2.xsl. This result is identical to the result when using <xsl:include>—but for a different, explicit reason.
Avoiding Conflicting Rules by Using `<xsl:apply-imports>`

The previous topic, [Deciding Whether to Import or Include an XSLT File](#), discussed some advantages of using imported (vs. included) style sheets. This topic discusses another important advantage of imported style sheets.

The declarations of an included style sheet are either applied or not applied, depending on whether there are conflicting declarations in the including style sheet, and on the sequence in which the conflicting declarations appear.

When you import a style sheet, however, you have another option. You can supplement the imported declarations with others. To do so, the importing style sheet needs an appropriate `<xsl:import>` element, as always. However, an `<xsl:apply-imports>` element also appears at the appropriate point in the template rule. This template rule would otherwise simply override its counterpart in the imported style sheet.

The syntax of the empty `<xsl:apply-imports>` element is straightforward. It does not even include any attributes:

```xml
<xsl:apply-imports/>
```

To see how this can be used, consider the files discussed earlier in this topic:

- `jamming.xml`, in [Initial Example for Importing Style Sheets](#)
- `jamming.xsl`, in [Using `<xsl:include>` to Insert an XSLT File](#)
- `chord.xsl`, in [Using `<xsl:include>` to Insert an XSLT File](#)

The above version of `jamming.xsl` contains template rules for processing fiction-related elements in the *Synchronized Jamming* book. It also includes two style sheets, via `<xsl:include>`. These included style sheets handle other elements, such as markup code and musical annotations.

To illustrate the advantage of supplemental imported declarations, we'll begin by changing the following line in `jamming.xsl`. 

```xml
""
The new line is as follows:

```xml
<xsl:import href="chord.xsl"/>
```

**Note** Because any `<xsl:import>` elements must be the first top-level elements in a style sheet, we also have to move this line to a position above the `<xsl:include href="markup_code.xsl"/>` element in `jamming.xsl`.

So far, this has no effect on the output as viewed in Internet Explorer. No conflict is introduced, because `jamming.xsl` itself has no template rules for processing `jamming.xml`'s `<chord>` element, its attributes, or its child elements.

However, the production designers of *Synchronized Jamming* do face a conflict. The conflict is not between template rules, but between design standards. According to these design standards, any table embedded in a Web page must be surrounded by a double box, to make it stand out from plain text. The production designers have three choices for resolving the design conflict:

1. Do nothing. They can accept ScootMusic's styling of the `<chord>` element as a borderless table, and therefore discard, for this book only, their own design standards for tables.
2. Physically copy and paste the contents of `chord.xsl`'s template rule into `jamming.xsl`. The drawback here is that if ScootMusic ever changes `chord.xsl`, *Synchronized Jamming*'s designers will have to re-edit `jamming.xsl` to maintain consistency.
3. Use an `<xsl:apply-imports>` element to customize the table's appearance. This way, they can incorporate `chord.xsl`'s template rule for the `<chord>` branch of the source tree, and double-border it as well.

For option #3, add a template rule for the `<chord>` element to `jamming.xsl`, as follows:

```xml
<xsl:template match="chord">
  <div style="border-style:double; width:50%">
    <xsl:apply-imports/>
  </div>
</xsl:template>
```

The XSLT processor creates the bordered `<div>` element in the result tree at the
point of the source tree's <chord> element. Then, at the point of the <xsl:apply-imports>, the processor examines all imported style sheets to see if they also provide a template rule for the <chord> element. If so, the imported template rule is fired within the resulting <div> element. The effect is as though the only template rule for the <chord> element was the following (the portion from chord.xsl is in bold):

```
<xsl:template match="chord">
  <div style="border-style:double; width:50%">
    <div style="width:75%; margin-left:20; padding:10">
      <table border="0">
        [and so on]
      </table>
    </div>
  </div>
</div>
```

**Formatted Output**

When viewed in Internet Explorer, the relevant portion of the page now appears as follows:

![Chord Notes](image)

<table>
<thead>
<tr>
<th>Chord Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>Duration: 4.5 seconds</td>
</tr>
<tr>
<td>Voice: steel</td>
</tr>
<tr>
<td>Vibrato: on</td>
</tr>
</tbody>
</table>
Finished Example for Importing and Including

Example

XML File (jamming.xml)

The following is a brief excerpt from the first chapter of *Synchronized Jamming*. The structure of the book is a mixture of common elements of fiction, literal HTML code, and musical annotation.

```xml
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="jamming.xsl"?>
<novel copyright="2001" isbn="555000000X">
  <title>Synchronized Jamming</title>
  <author>Kari Hensien</author>
  <chapter number="1">
    <title>The Bleeding Edge</title>
    <epigraph><text>Any sufficiently advanced technology is indistinguishable from noise.</text><author>Canopy Rose</author><source>The Fusion Manifesto</source></epigraph>
    <para>Politics. Technology. Rock'n'roll. All you needed to know about the underground band known as Canopy Rose was right there in the first couple grafs of their so-called "Fusion Manifesto":</para>
    <markup_code><![CDATA[
<html>
<head><title>Fused</title></head>
<body>
<p>Listen. You know as well as we do...</p>
<p>... the sound of one hand jammin':</p>
]]></markup_code>
    <chord duration="4.5" units="seconds" voice="steel" vibrato="on">
      <note>A</note>
      <note>C</note>
    </chord>
    <markup_code><![CDATA[
<p>Welcome to the block party. Welcome to the revolution.</p>
]]></markup_code>
  </chapter>
</novel>
```
XSLT File (jamming.xsl)

This is the final version of the file.

```xml
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/TR/REC-html40">
  <xsl:import href="chord.xsl"/>
  <xsl:include href="markup_code.xsl"/>
  <xsl:template match="novel">
    <html>
      <head>
        <title>Excerpt: 
          "<xsl:value-of select="title"/>", 
        by 
        <xsl:value-of select="author"/>
        </title>
        <style>
          * {font-family: Arial, Helvetica, Univers, sans-serif}
          pre {font-family: Courier New, Courier, monospace}
        </style>
      </head>
      <body>
        <h2>Excerpt:<br/>
        <em><xsl:value-of select="title"/></em></h2>
        <h3>by <xsl:value-of select="author"/></h3>
        <h6>Copyright <xsl:value-of select="@copyright"/>
        by <xsl:value-of select="author"/></h6>
        <hr size="2" width="75%" align="left"/>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="novel/title">
    </xsl:template>
  <xsl:template match="novel/author">
    </xsl:template>
  <xsl:template match="chapter">
    <div style="width:75%">
      <h2>Chapter <xsl:value-of select="@number"/>: <br/>
      <xsl:value-of select="title"/></h2>
      <xsl:apply-templates/>
    </div>
  </xsl:template>
</xsl:stylesheet>
```
Imported XSLT File (chord.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">
```

```
<xsl:template match="chord">
  <div style="width:75%; margin-left:20; padding:10">
    <table border="0">
      <tr>
        <th style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Chord Notes
        </th>
      </tr>
      <tr>
        <xsl:for-each select="note">
          <td><b><xsl:value-of select="."/></b></td>
        </xsl:for-each>
      </tr>
      <tr>
        <td style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
        </td>
      </tr>
      <tr>
        <td style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Voice: <xsl:value-of select="@voice"/>
        </td>
      </tr>
      <tr>
        <td style="background-color: yellow">
          <xsl:attribute name="colspan"><xsl:value-of select="count(note)"/></xsl:attribute>
          Vibrato: <xsl:value-of select="@vibrato"/>
        </td>
      </tr>
    </table>
  </div>
</xsl:template>

Included XSLT File (markup_code.xsl)

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">
  <xsl:template match="markup_code">
    <div style="width:75%; margin-left:20; padding:10">
```
```
Excerpt:
*Synchronized Jamming*

by Kari Hensien

Copyright 2001 by Kari Hensien

Seoulheey Publishing / ISBN 555000000X

---

Chapter 1:
The Bleeding Edge

Politics. Technology. Rock'n'roll. All you needed to know about the underground band known as Canopy Rose was right there in the first couple grafts of their so-called "Fusion Manifesto":

---

Any sufficiently advanced technology is indistinguishable from noise.
---

-- Canopy Rose (The Fusion Manifesto)

---

<p>Listen. You know as well as we do...</p>
<p>... the sound of one hand jammin':</p>

---

Chord Notes

A, C

Duration: 4.5 seconds
Voice: steel
Vibrato: on

---

<p>Welcome to the block party. Welcome to the revolution.</p>
Using XSLT Keys to Increase Performance

Using XSLT to process a large XML document with many data points can involve many performance issues, especially when specific nodes within the document are to be processed repeatedly within a single transformation.

For example, a document representing a publishing company's book catalog might contain data about hundreds or thousands of books. In a given transformation, many of these book titles might be referenced a dozen times, not only in the descriptions of the books, but also in an index of all titles, a list of featured titles, and so on. Searching or cross-referencing the document many times for a given title or a title that matches a specific condition can result in unacceptable performance.

XSLT solves this problem by supporting the use of keys to uniquely identify a node or class of nodes. Each key is used by the XSLT processor to build an index, which the processor can reference when a particular value is needed. This index enables the processor to retrieve the keyed node without searching for it. Using keys can also greatly simplify the coding of XPath expressions.

- [Deciding whether to Use IDs or Keys](#)
- [Using the `<xsl:key>` Element](#)
- [Using the `key()` Function](#)
- [Using XSLT Keys to Group Data](#)
- [Sample XML Data File for XSLT Keys](booksvariant.xml)
- [Sample XSLT File for XSLT Keys](lib_cat.xsl)
Deciding whether to Use IDs or Keys

The XML specification itself provides a simple means to uniquely identify an element, using an ID-type attribute. A conforming XML processor, such as the MSXML processor, is expected to build an index of any elements with IDs, enabling them to be cross-referenced. They might be cross-referenced, for example, by other elements with a corresponding IDREF-type attribute.

Why, then, does the XSLT standard provide keys for essentially the same purpose? There are several drawbacks to using IDs alone:

- You must use an attribute value as the element's ID. That is, you can't use the element's content as an ID, or the content of some other element related to the element you want to identify. For example, if an author's name is a desired index value, but is held in an `<author_name>` element rather than as an attribute value, there is no way to index the name using XML alone.
- A given element can have no more than one ID-type attribute. This means, for example, that you cannot hold both a book's ISBN and its publication date in ID-type attributes.
- Each ID value must be unique in a given document. If `pub_date` is an ID-type attribute, this effectively prevents two books from sharing the same publication date.
- In order to use ID attributes at all, they must be declared as such in a Document Type Definition (DTD). If the document is merely well-formed and therefore does not use a DTD, none of its attributes can be IDs.
- All IDs in a document constitute a single set of IDs. You can't establish multiple sets of IDs that are different but related. For example, if you've designed a document type that uses ISBNs to uniquely identify books, and author numbers to uniquely identify authors, there's no way to treat these two kinds of IDs distinctly.
- The name of an XML ID must be a legitimate XML name. For example, it cannot contain spaces, and it must start with a letter. This forces you to use somewhat artificial values, such as "isbn0123456789" and "AFarewellToArms" as the unique identifier, rather than "0123456789" and "A Farewell to Arms," respectively.

The use of keys enables an XSLT style sheet to circumvent any or all of these
restrictions.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Using the `<xsl:key>` Element

The `<xsl:key>` element is a top-level element—that is, a child of the `<xsl:stylesheet>` element. The purpose of `<xsl:key>` is to declare the name of a key. The node corresponding to the key can then be retrieved using the `key()` function (see Using the `key()` Function). You can include as many `<xsl:key>` elements in your style sheets as needed.

The syntax for the `<xsl:key>` element is as follows:

```xml
<xsl:key name="name" match="node" use="expression" />
```

Note that `<xsl:key>` is always an empty element.

All the three attributes are required. Their values are as follows:

- **name**: This is a text string that assigns a name to the key. This name will be used by the `key()` function to retrieve the corresponding node from the source tree.
- **node**: This is the name of the node to which to assign the key. The value cannot contain a reference to a variable name.
- **expression**: Here, use an XPath expression that will be the value of the key. Note that this can be a string instead of one of the more common XPath expressions, which consist of one or more location steps. It can also be a call to the XSLT `generate-id` Function, which will assign a unique value to the key for this node even if no other unique value is available. The `expression` attribute, like the `node` attribute, cannot contain a reference to a variable name.

There are no constraints on the number of keys that can be assigned to a given node. Likewise, any number of nodes can be assigned keys with the same name and/or value.

**Note** The XSLT standard sets few limits on the use of keys. However, a given XSLT application using keys might be limited by the hardware environment (particularly memory) in which the application is running. Also, the performance gains made possible by the use of keys require a
An example of the `<xsl:key>` element is:

```xml
<xsl:key name="isbn_key" match="book" use="@isbn"/>
```

This declares a key named `isbn_key`. This key will be assigned to each `<book>` element, and will have as its value the value of that `<book>` element's `isbn` attribute. For example, the book *Jambing on the Trixels* will be assigned an `isbn_key` whose value is "991100-001"; *For Love of a Toothpick* will have `isbn_key="991100-002"`; and so on.

Here is another example:

```xml
<xsl:key name="categ_key" match="book" use="category"/>
```

This tells the XSLT processor to build a key for the `<category>` child of each `<book>` element.

Though a key is most commonly assigned to an element based on the value of a child text node or element, or on the value of an attribute, the `use` attribute can readily take any other XPath expression relative to the element. For example, you could assign a key as follows:

```xml
<xsl:key name="book_for_categ" match="category"
         use="../title"/>
```

This assigns a key to each `<category>` element. The value of this key is the corresponding `<book>` element's `<title>` child.

**Files Used for This Topic**

- [Sample XML Data File for XSLT Keys](#)
- [Sample XSLT File for XSLT Keys](#)
Using the key() Function

The other side of the `<xsl:key>` element is provided by the built-in XSLT `key()` function. The purpose of this function is to retrieve the node or nodes keyed to a particular name and value.

The syntax of the `key()` function is as follows:

```
key(name, value)
```

The `key()` function returns a node-set containing all nodes which match the key identified by the arguments. The following arguments are passed to it.

- **name**: A string providing the name of the key to use. It corresponds to the name of one or more keys defined by `<xsl:key>` elements.
- **value**: The value of the desired key. Technically, this argument can be of any data type. However, if it is a node-set, the function will return all nodes with the same value. Note that this argument can be a variable reference.

If the value of `name` does not match the name of a key defined by a corresponding `<xsl:key>` element, the `key()` function will return an empty node-set.

The following table shows some brief examples of the node-sets returned by calls to the `key()` function:

<table>
<thead>
<tr>
<th>Key declaration</th>
<th>key() function call</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:key name=&quot;isbn_key&quot; match=&quot;book&quot; use=&quot;@isbn&quot;/&gt;</code></td>
<td><code>key(&quot;isbn_key&quot;, &quot;991100-102&quot;)</code></td>
<td>The <code>&lt;book&gt;</code> element that corresponds to the title <code>Quantum Superstring Electrodynamics for Newbies</code></td>
</tr>
<tr>
<td><code>&lt;xsl:key name=&quot;categ_key&quot; match=&quot;book&quot; use=&quot;category&quot;/&gt;</code></td>
<td><code>key(&quot;categ_key&quot;, &quot;Fiction&quot;)</code></td>
<td>The <code>&lt;book&gt;</code> element that corresponds to <code>Jambing on the Trixels</code></td>
</tr>
<tr>
<td><code>&lt;xsl:key name=&quot;categ_key&quot; match=&quot;book&quot; use=&quot;category&quot;/&gt;</code></td>
<td><code>key(&quot;categ_key&quot;, &quot;Nonfiction&quot;)</code></td>
<td>Every <code>&lt;book&gt;</code> element, except the one that</td>
</tr>
</tbody>
</table>
The two <book> elements that correspond to *Fretting Over Your Guitar* and *Quantum Superstring Electrodynamics for Newbies* correspond to *Jambing on the Trixels*.

Files Used for This Topic

- [Sample XML Data File for XSLT Keys](booksvariant.xml) (booksvariant.xml)
- [Sample XSLT File for XSLT Keys](lib_cat.xsl) (lib_cat.xsl)
Sample XML Data File for XSLT Keys

Topics in this section use the following XML document. This document is a portion of the catalog at a large metropolitan public library, developed by Scootney Publishing as part of a promotional exhibit.

XML File (booksvariant.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="lib_cat.xsl"?>
<!DOCTYPE book [ 
<!ENTITY oumlaut "&#38;#246;"/>
]> 

<book isbn="991100-001" catnum="101.3490">
  <title>Jambing on the Trixels</title>
  <author>Tristan Randall</author>
  <publisher>Scootney Publishing</publisher>
  <copyright>2000</copyright>
  <pages>496</pages>
  <category>Fiction</category>
  <description>Tristan Randall scores another action-adventure hit with this fast-paced tale of murder among the faculty teaching semantics at a small New England university.</description>
</book>

<book isbn="991100-002" catnum="432.9431">
  <title>For Love of a Toothpick</title>
  <author>J&ouml;rg Frey</author>
  <publisher>Scootney Publishing</publisher>
  <copyright>2000</copyright>
  <pages>175</pages>
  <category>Non-fiction</category>
  <description>Looking for a whimsical introduction to the mysterious world of miscellaneous dining and personal-grooming utensils? This is the book for you!</description>
</book>

<book isbn="991100-010" catnum="894.3207">
  <title>Desperation 101</title>
  <author>Jack Gode</author>
  <publisher>Scootney Publishing</publisher>
  <copyright>2000</copyright>
  <pages>262</pages>
```
<category>Non-fiction</category>
<description>Self-help guru Gode recounts his personal journey from confusion to certainty, with the help of a most unusual therapy: highway hypnosis.</description>
</book>

<book isbn="991100-008" catnum="562.1093">
<title>Fretting Over Your Guitar</title>
<author>Becki Culbert</author>
<publisher>ScootMusic (a Division of Scootney Publishing)</publisher>
<copyright>1998</copyright>
<pages>160</pages>
<category>Non-fiction</category>
<description>In part a traditional "fake book," in part a profound exploration of the relationship between musician and instrument, this little volume surprises the reader on every page.</description>
</book>

<book isbn="991100-102" catnum="450.9043">
<title>Quantum Superstring Electrodynamics for Newbies</title>
<author>Becki Culbert</author>
<publisher>Scootney Publishing</publisher>
<copyright>2001</copyright>
<pages>656</pages>
<category>Non-fiction</category>
<description>Multi-dimensional author Culbert turns her attention to Quantum Superstring Electrodynamics, clearly explaining every facet of this frighteningly complex topic in a mere 600-plus pages, many of them containing words understandable to the rankest of newcomers.</description>
</book>
Sample XSLT File for XSLT Keys

Some topics in this section use the following file.

XSLT File (lib_cat.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:(key name="categ_key" match="book" use="category"/>
<xsl:key name="author_key" match="book" use="author"/>
<xsl:template match="/">
<html>
<head>
<title>Scootney Publishing: Special Exhibit</title>
<style>
    h1 {font-family: Arial, Helvetica, Univers, sans-serif;
        font-size: 18pt; font-weight: bold;
        background-color: teal;
        width: 75%;}
    h2 {font-family: Arial, Helvetica, Univers, sans-serif;
        font-size: 14pt; font-weight: bold;
        background-color: white;
        width: 75%;}
    h3 {font-family: Arial, Helvetica, Univers, sans-serif;
        font-size: 12pt; font-weight: bold;
        background-color: gold;
        width: 75%;
        border-width: 1;
        border-style: solid}
</style>
</head>
<body>
<h1>Scootney Publishing: Special Exhibit Catalog</h1>
<xsl:apply-templates select="lib_cat" mode="grp_categ"/>
<xsl:apply-templates select="lib_cat" mode="grp_author"/>
</body>
</html>
</xsl:template>
<xsl:template match="lib_cat" mode="grp_categ">
<h2>Titles Grouped by Category</h2>
<xsl:for-each select="book[count(. | key('categ_key', category))[

```
<h3><xsl:value-of select="category" /></h3>
<xsl:for-each select="key('categ_key', category)">
  <xsl:sort select="title" />
  <table border="0" width="75%">
    <tr>
      <th width="10%" align="right">Title</th>
      <td width="90%" align="left"><xsl:value-of select="title" /></td>
    </tr>
    <tr>
      <th width="10%" align="right">Author</th>
      <td width="90%" align="left"><xsl:value-of select="author" /></td>
    </tr>
    <tr>
      <th width="10%" align="right">Copyright</th>
      <td width="90%" align="left"><xsl:value-of select="copyright" /></td>
    </tr>
    <tr>
      <th width="10%" align="right">Pages</th>
      <td width="90%" align="left"><xsl:value-of select="pages" /></td>
    </tr>
    <tr>
      <th width="10%" align="right" valign="top">Summary</th>
      <td width="90%" align="left" valign="top"><xsl:value-of select="description" /></td>
    </tr>
  </table>
</xsl:for-each>
</xsl:for-each>
</xsl:template>

<xsl:template match="lib_cat" mode="grp_author">
  <h2>Titles Grouped by Author</h2>
  <h3><xsl:value-of select="author" /></h3>
  <xsl:for-each select="key('author_key', author)">
    <xsl:sort select="title" />
    <table border="0" width="75%">
      <tr>
        <th width="10%" align="right">Title</th>
        <td width="90%" align="left"><xsl:value-of select="title" /></td>
      </tr>
      <tr>
        <th width="10%" align="right">Category</th>
        <td width="90%" align="left"><xsl:value-of select="category" /></td>
      </tr>
      <tr>
        <th width="10%" align="right">Copyright</th>
        <td width="90%" align="left"><xsl:value-of select="copyright" /></td>
      </tr>
    </table>
  </xsl:for-each>
</xsl:template>
Using XSLT Keys to Group Data

One of the most common uses of keys is for grouping data in a document. We've already discussed the `<xsl:sort>` element in Sorting with XSLT. However, with sorting alone, you must repeat each of the items establishing a category. What if you want to display the category only once, followed by a list of all items that fall under it?

The public library participating in Scootney's promotional exhibit wants to produce a single Web page that groups the titles in the exhibit in two ways. One grouping will show the titles grouped by category. The other grouping will show them grouped by author.

The style sheet developed by the library, `lib_cat2.xsl`, defines the following two keys that correspond to these two groupings.

- `<xsl:key name="categ_key" match="book" use="genre"/>
- `<xsl:key name="author_key" match="book" use="author"/>

There are also two template rules for the root `<catalog>` element. These template rules use modes to enable the element to be processed twice—once for each grouping. These template rules are invoked by `<xsl:apply-templates>` elements within the template rule for the document root:

```xml
<xsl:template match="/">
...
  <xsl:apply-templates select="catalog" mode="grp_categ"/>
  <xsl:apply-templates select="catalog" mode="grp_author"/>
...
</xsl:template>
```

The two template rules for the `<catalog>` element take two slightly different approaches to the problem of grouping, although both use keys to do so. The following topics describe these two approaches.

- Grouping Data by Examining the Keyed Node-set
- Grouping Data by Using `generate-id()`
Files Used for This Topic

- Sample XML Data File for XSLT Keys (booksvariant.xml)
- Sample XSLT File for XSLT Keys (lib_cat.xsl)
Grouping Data by Examining the Keyed Node-set

By definition, a node-set cannot contain a given node more than once. Look at the following portion of the <catalog> element's template rule, the grp_categ mode, for grouping the books by genre. See if you can tell what the references to the categ_key key are doing.

```xml
<xsl:template match="catalog" mode="grp_categ">
  <h2>Titles Grouped by Genre</h2>
  <xsl:for-each select="book[count(./key('categ_key', genre)[1])=1]">
    <h3><xsl:value-of select="genre"/></h3>
    <xsl:for-each select="key('categ_key', genre)">
      <xsl:sort select="title"/>
      ...
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

The predicate portion of the outer `<xsl:for-each>` element's select attribute looks rather complex, but it essentially consists of two separate conditions joined by the XPath union operator, a pipe or vertical-bar symbol (|). So `select="book[count(./key('categ_key', genre)[1])=1]"` translates as selecting the current `<book>` element if it is also the first item in the "genre" category as returned by the `key('categ_key', genre)[1]` function.

The union operator | returns a node-set joined by the selected `<book>` element and the first item of the "genre" category. If the size of this node-set is one (1), the selected `<book>` is the first item in the "genre" category and the template rule instantiates a level-3 heading in the result tree, displaying the name of the book's category. Otherwise, the selection is bypassed.

The second reference to the `key()` function is where the actual grouping occurs, within the category established by the outer `<xsl:for-each>`. It says, within each of the node-sets established by the `categ_key` key, to list the titles and so on of all books in the group.

Example
XML File (books.xml)

Use the Sample XML File (books.xml). Below the line `<?xml version='1.0'?>`, add the following line:

```xml
<?xml-stylesheet type="text/xsl" href="lib_cat2.xsl"?>
```

XSLT File (lib_cat2.xsl)

This file uses the `<genre>` element as the category for sorting.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:key name="categ_key" match="book" use="genre"/>
    <xsl:key name="author_key" match="book" use="author"/>

    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Scootney Publishing: Special Exhibit</TITLE>
                <STYLE>
                    h1 {font-family: Arial,Helvetica,Univers,sans-serif;
                        font-size: 18pt; font-weight: bold;
                        background-color: teal;
                        width: 75%}
                    h2 {font-family: Arial,Helvetica,Univers,sans-serif;
                        font-size: 14pt; font-weight: bold;
                        background-color: white;
                        width: 75%}
                    h3 {font-family: Arial,Helvetica,Univers,sans-serif;
                        font-size: 12pt; font-weight: bold;
                        background-color: gold;
                        width: 75%;
                        border-width: 1;
                        border-style: solid}
                </STYLE>
            </HEAD>
            <BODY>
                <h1>Scootney Publishing: Special Exhibit Catalog</h1>
                <xsl:apply-templates select="catalog" mode="grp_categ"/>
                <xsl:apply-templates select="catalog" mode="grp_author"/>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
<h2>Titles Grouped by Genre</h2>
<xsl:for-each select="book[count(.) | key('categ_key', genre)[1]]">
    <h3><xsl:value-of select="genre"/></h3>
    <xsl:for-each select="key('categ_key', genre)">
        <xsl:sort select="title"/>
        <TABLE border="0" width="75%">
            <tr>
                <th width="10%" align="right">Title</th>
                <td width="90%" align="left"><xsl:value-of select="title"/></td>
            </tr>
            <tr>
                <th width="10%" align="right">Author</th>
                <td width="90%" align="left"><xsl:value-of select="author"/></td>
            </tr>
            <tr>
                <th width="10%" align="right">Copyright</th>
                <td width="90%" align="left"><xsl:value-of select="publish_date"/></td>
            </tr>
            <tr>
                <th width="10%" align="right">Price</th>
                <td width="90%" align="left"><xsl:value-of select="price"/></td>
            </tr>
            <tr>
                <th width="10%" align="right" valign="top">Summary</th>
                <td width="90%" align="left" valign="top"><xsl:value-of select="description"/></td>
            </tr>
        </TABLE>
    </xsl:for-each>
</xsl:for-each>
</xsl:template>

<xsl:template match="catalog" mode="grp_author">
    <h2>Titles Grouped by Author</h2>
    <xsl:for-each select="book[generate-id() = generate-id(key('author_key', author))]">
        <xsl:sort select="author"/>
        <h3><xsl:value-of select="author"/></h3>
        <xsl:for-each select="key('author_key', author)">
            <xsl:sort select="title"/>
            <TABLE border="0" width="75%">
                <tr>
                    <th width="10%" align="right">Title</th>
                    <td width="90%" align="left"><xsl:value-of select="title"/></td>
                </tr>
                <tr>
                    <th width="10%" align="right">Genre</th>
                    <td width="90%" align="left"><xsl:value-of select="genre"/></td>
                </tr>
                <tr>
                    <th width="10%" align="right">Copyright</th>
                    <td width="90%" align="left"><xsl:value-of select="publish_date"/></td>
                </tr>
                <tr>
                    <th width="10%" align="right">Price</th>
                    <td width="90%" align="left"><xsl:value-of select="price"/></td>
                </tr>
            </TABLE>
        </xsl:for-each>
    </xsl:for-each>
</xsl:template>
## Formatted Output

A portion of the output is shown here.

### Scootney Publishing: Special Exhibit Catalog

#### Titles Grouped by Genre

<table>
<thead>
<tr>
<th>Computer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Microsoft .NET: The Programming Bible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>O'Brien, Tim</td>
</tr>
<tr>
<td>Copyright</td>
<td>2000-12-09</td>
</tr>
<tr>
<td>Price</td>
<td>36.95</td>
</tr>
<tr>
<td>Summary</td>
<td>Microsoft's .NET initiative is explored in detail in this deep programmer's reference.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>MSXML3: A Comprehensive Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>O'Brien, Tim</td>
</tr>
<tr>
<td>Copyright</td>
<td>2000-12-01</td>
</tr>
<tr>
<td>Price</td>
<td>36.95</td>
</tr>
<tr>
<td>Summary</td>
<td>The Microsoft MSXML3 parser is covered in detail, with attention to XML DOM interfaces, XSLT processing, SAX and more.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Visual Studio 7: A Comprehensive Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Galos, Mike</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Grouping Data by Using generate-id()

As an alternative to the complex syntax necessary to examine each node in a key's node-set, you can take advantage of the generate-id() function's ability to uniquely identify a node.

Here's an excerpt from the template rule in lib_cat2.xsl that groups books by author's name. The complete file is in Grouping Data by Examining the Keyed Node-set.

```xml
<xsl:template match="catalog" mode="grp_author">
  <h2>Titles Grouped by Author</h2>
  <xsl:for-each select="book[generate-id() = generate-id(key('author_key', author))]
    <xsl:sort select="author"/>
    <h3><xsl:value-of select="author"/></h3>
    <xsl:for-each select="key('author_key', author)">
      <xsl:sort select="title"/>
      ...
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

Here, the predicate portion of the outer <xsl:for-each> element's select attribute translates as follows: Select a <book> element only if its generated ID equals that of the first <book> element in the node-set established by the author_key key. When such a <book> is found, the template rule instantiates a level-3 heading, displaying the value of the corresponding <author> element.

The inner <xsl:for-each> translates as follows: Select all <book> elements in the node-set established by the author_key key, sort the books by a given author by their titles, and so on.

Example

XML File (books.xml)

Use the Sample XML File (books.xml). Below the line <?xml version='1.0'?>, add the following line:
XSLT File (lib_cat2.xsl)

See the listing in Grouping Data by Examining the Keyed Node-set.

Formatted Output

A portion of the output is shown here.

Titles Grouped by Author

<table>
<thead>
<tr>
<th>Corets, Eva</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
</tr>
<tr>
<td><strong>Genre</strong></td>
</tr>
<tr>
<td><strong>Copyright</strong></td>
</tr>
<tr>
<td><strong>Price</strong></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
</tr>
</tbody>
</table>

| **Title**  | Oberon's Legacy |
| **Genre**  | Fantasy |
| **Copyright** | 2001-03-10 |
| **Price**  | 5.95 |
| **Summary** | In post-apocalypse England, the mysterious agent known only as Oberon helps to create a new life for the inhabitants of London. Sequel to Maeve Ascendant. |

| **Title**  | The Sundered Grail |
| **Genre**  | Fantasy |
| **Copyright** | 2001-09-10 |

See Also

Grouping Data by Examining the Keyed Node-set | generate-id Function

Forward-compatible processing is also used when extensions have been applied to the XSLT parser. The extension namespace must be specified in the `<xsl:stylesheet>` element, using the `extension-element-prefixes` (or `xsl:extension-element-prefixes`) with each extension prefix used by the
parser separated by white space.
Creating Interactive XSLT

Interactive XSLT is XSLT that involves user interfaces. Typically, this is done on the Web. You can create such interactive XSLT applications with the help of either ASP or HTML. With ASP, XSLT is performed on the server side before the resulting HTML document or fragment is downloaded to the client. With HTML, XSLT is performed on the client side after the required XML documents and XSLT style sheets are downloaded. One of the advantages of server-side XSLT is that the application is free of browser compatibility issues. One of the advantages of client-side XSLT is that, once XML and XSLT files are downloaded, the application is relatively immune to server shutdown or network traffic jams.

This section contains the following topics.

- Using ASP with XSLT Processor and Template Objects
- Using XML Data Islands and Client-Side XSLT
- Controlling Output Format with the media-type Attribute
Using ASP with XSLT Processor and Template Objects

Performance is often a big concern for ASP applications when many clients request the services. This applies to ASP/XSLT applications as well. For example, the following ASP code can result in poor performance and unpleasant experiences for the user.

```<%@LANGUAGE="JavaScript"%>
<%Response.expires=-1;
function showSelection(){
    var xmlDoc=new ActiveXObject("Msxml2.DOMDocument.5.0");
    xmlDoc.async=false;
    xmlDoc.load(Server.mapPath(Request("xmlDoc")));

    var xsltDoc=new ActiveXObject("Msxml2.DOMDocument.5.0");
    xsltDoc.async=false;
    xsltDoc.load(Server.mapPath(Request("xsltDoc")));

    xmlDoc.transformNode(xsltDoc);
}
showSelection();%>
```

The problem with the ASP code above is that the `transformNode` method (or its variant, `transformNodeToObject`) must compile the XSLT style sheet (`xsltDoc`) every time a user makes a request. To avoid such a pitfall, use the `XSLTemplate` DOM object to cache the compiled XSLT style sheet, and use the `XSLTProcessor` DOM object to activate the transformation using the cached style sheet.

This approach includes a subtle but useful feature. The `XSLTProcessor` object allows you to pass a parameter to the style sheet, so that you can write a single XSLT style sheet to transform different XML source documents based on this input parameter. Otherwise, you might need to keep different style sheets for different source documents.

**Advantages of using XSLProcessor vs. TransformNode()**
The following are some advantages of using XSLTemplate and XSLTProcessor in an ASP/XSLT application.

- **Scalability.** Caching the XSLT style sheet helps an ASP/XSLT application to scale better. However, it is not generally recommended to cache an XML document, due to the memory constraints involved.

- **Performance.** Using XSLTemplate means that the style sheet needs to be compiled only once. This considerably reduces the amount of processing involved. The XSLTemplate object can also be cached as an MTS object. The processing of this object can then be optimized across sessions, as well as within a session.

- **Parameterization.** The XSLTProcessor object enables you to pass parameters into the style sheet.

- **Extensions.** You can also pass COM objects into XSLT documents as namespace extensions.

**Example**

The following is an example of ASP/XSLT application for a catalog of books. It can display books of a chosen genre, which is selected from a selection box. The style sheet includes both the selection box and a table that displays the requested subset of books. If no selection is made, all the books are displayed.

The example contains the following files.

- Sample XSLT File for ASP XSLT Processor Objects *(catalogFilter.xsl)*
- Sample ASP Script for XSLT Processor Objects *(catalog.asp)*
- Sample XML File *(books.xml)*

**To run this example**

- Put the files in a directory on an HTTP server. Enter a URL, such as http://www.myserver.com/catalog.asp.

**Results**

When the ASP file is first invoked with no parameters, the requesting URL is this format: "http://www.myserver.com/catalog.asp". All the records will be shown. When the drop-down list box is set to a different field (such as
"Computer"), the column for Genre is dropped and only those books in the indicated genre are shown, in alphabetical order:

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Author</th>
<th>Publication Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microsoft .NET: The Programming Bible</td>
<td>O'Brien, Tim</td>
<td>2000-12-09</td>
<td>Microsoft's .NET initiative is explored in detail in this deep programmer's reference</td>
</tr>
<tr>
<td>2</td>
<td>MSXML 3 A Comprehensive Guide</td>
<td>O'Brien, Tim</td>
<td>2000-12-01</td>
<td>The Microsoft MSXML 3 parser is covered in detail, with attention to XML DOM interfaces, XSLT processing, SAX and more</td>
</tr>
<tr>
<td>3</td>
<td>Visual Studio 7: A Comprehensive Guide</td>
<td>Gates, Mike</td>
<td>2001-04-16</td>
<td>Microsoft Visual Studio 7 is explored in depth, looking at how Visual Basic, Visual C++, C#, and ASP.NET are integrated into a comprehensive development environment</td>
</tr>
<tr>
<td>4</td>
<td>XML Developer's Guide</td>
<td>Gemmell, Matthew</td>
<td>2000-10-01</td>
<td>An in-depth look at creating applications with XML</td>
</tr>
</tbody>
</table>

In this example, the main() function can be modified to handle any data source, style sheet, and set of parameters, without changing any other routines. You can even use the names of different style sheets as HTML parameters, and pass them to be applied when need. This can be useful if you work with different types of output (HTML 4.0, Internet Explorer 5 targeted HTML, XML, WML, and so on), handle SOAP objects, or need to present the same information in various ways (summaries, charts, graphs, and so on).
Sample XSLT File for ASP XSLT Processor Objects

XSLT File (CatalogFilter.xsl)

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt"
    version="1.0">
    <xsl:output method="html"/>
    <xsl:param name="selected_genre" select='"all"'/>
    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Displayed Catalog</TITLE>
            </HEAD>
            <BODY>
                <STYLE>
                    .catalog_genre_head {background-color:darkGreen;font-size:24pt;color:white;font-family:Impact;}
                    .catalog_head {background-color:green;font-size:18pt;color:white;font-family:Impact;}
                    .catalog_row0 {background-color:lightGreen;}
                    .catalog_row1 {background-color:white;}
                    .catalog_row_end {background-color:darkGreen}
                </STYLE>
                <FORM method="post" action="catalog.asp">
                    Genre
                    <SELECT name="genre" value="{$selected_genre}" onchange="submit()">
                        <OPTION value="all">All</OPTION>
                        <OPTION value="Computer">Computer</OPTION>
                        <OPTION value="Fantasy">Fantasy</OPTION>
                        <OPTION value="Horror">Horror</OPTION>
                        <OPTION value="Romance">Romance</OPTION>
                        <OPTION value="Science Fiction">Science Fiction</OPTION>
                    </SELECT>
                </FORM>
                <BR/>
                <xsl:apply-templates select="catalog"/>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
### Catalog Table

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Author</th>
<th>Publication Date</th>
<th>Description</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Great Gatsby</td>
<td>F. Scott Fitzgerald</td>
<td>1925</td>
<td>A classic American novel</td>
<td>Romance</td>
</tr>
<tr>
<td>2</td>
<td>1984</td>
<td>George Orwell</td>
<td>1949</td>
<td>A dystopian novel</td>
<td>Science Fiction</td>
</tr>
<tr>
<td>3</td>
<td>To Kill a Mockingbird</td>
<td>Harper Lee</td>
<td>1960</td>
<td>A novel about race and class in the American South</td>
<td>Novel</td>
</tr>
<tr>
<td>4</td>
<td>Moby-Dick</td>
<td>Herman Melville</td>
<td>1851</td>
<td>A novel about whaling</td>
<td>Fiction</td>
</tr>
</tbody>
</table>

**Genre:**
- Romance
- Science Fiction
- Novel

---

**Functionality:**
- Filters books by genre
- Sorts books by title

---

**Usage:**
- Ideal for libraries or bookstores
- Easy to update with new titles

---

**Contact:**
- Library Director: info@library.com
- Bookstore Manager: sales@bookstore.com

---

**Terms:**
- All rights reserved
- Unauthorized use prohibited

---

**Privacy Policy:**
- Personal information is kept confidential
- Data is secure

---

**Disclaimer:**
- While every effort has been made to ensure accuracy, no warranty is given to the accuracy or completeness of the information in this catalog.
Sample ASP Script for XSLT Processor Objects

The following ASP script demonstrates most of the key features of the XSLTProcessor and XSLTemplate objects. It takes the genre parameter from a client-side HTML page and returns a table of all books from the book catalog in the given genre, sorted by title.

The ASP script contains the following functions:

- The loadSource() function creates a free-threaded DOM document that holds the catalog XML structure. The free-threaded nature of the document is important, because this generates a document that processes on a thread separate from the ASP document itself. This is critical for scalability if either the source or transformation XML is cached.

  **Note** In MSXML, "free-threaded" means ThreadingModel='Both', and cross-thread marshalling is supported.

- The getProcessor() function loads an XSLT file and caches it to an XSLTemplate object instantiated in this function. This XSLTemplate object supports the createProcessor method for creating an XSLProcessor object that will be used to activate the XSLT transformation.

- The transformData() function assigns the input for the processor to the source document, and sets the output to the Response object. The resulting XML structure is converted into a string and sent to the client for this particular case. Finally, the transformData() function calls the transform() method on the XSLProcessor object. This method actually performs the transformation that pumps the XML from the input device through the style sheet, and from there to the output device.

- The main() function parses the parameters a client sends to the server through the Request object. It tests to see if a genre parameter was passed in either a form or a query string, and assigns a default value of "all" to the parameter if nothing was passed. The processor.addParameter method then passes this value as a parameter to the XSLT style sheet. Note that the style sheet must have this parameter defined beforehand, even if no value is
ASP Script (catalog.asp)

```html
<%@LANGUAGE="JavaScript"%>
<%Response.expires=-1;
// catalog.asp
// Sorts books from a given catalog by title, for a given Genre.
function loadSource(sourceName){
    var xmlDoc=new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
    xmlDoc.async=false;
    xmlDoc.load(Server.mapPath(sourceName));
    return xmlDoc;
}

function getProcessor(transformName){
    if (""+Session(transformName)=="undefined"){
        var xslDoc=new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
        var xslTemplate=new ActiveXObject("Msxml2.XSLTemplate.5.0");
        xslDoc.async=false;
        xslDoc.load(Server.mapPath(transformName));
        xslTemplate.stylesheet=xslDoc;
        xslProcessor=xslTemplate.createProcessor();
        Session(transformName)=xslProcessor;
    }
    else {
        xslProcessor=Session(transformName);
    }
    return xslProcessor;
}

function transformData(srcDoc,processor){
    processor.input=srcDoc;
    processor.output=Response;
    processor.transform();
    Response.Flush();
    return true;
}

function main(){
    var srcDoc=loadSource("books.xml");
    var processor=getProcessor("catalogFilter.xsl");

    var genre=Request("genre");
    if(""+genre=="undefined"){
        genre="all";
    }
    else {
        genre=""+genre;
    }
    transformData(srcDoc,processor);
}
```
Note  The following is an alternative implementation of the
transformData() function.

function transformData(srcDoc,processor){
  processor.input=srcDoc;
  processor.transform();
  Response.Write(processor.output);
  return true;
}
Using XML Data Islands and Client-Side XSLT

XSLT running on the client can be an effective way of creating applications that require a minimal amount of interaction with a server. After the XML, XSLT, and (in some cases) HTML files are downloaded to the client, all or most operations are carried out on the client host. This usually means more responsive user interactions.

The major drawback of client-side XSLT is that you, the XSLT developer, will have to deal with potential browser incompatibility issues. Therefore, client-side XSLT is best for applications running on an intranet, where it is possible to establish a consistent environment for creating and deploying applications.

To use XML data islands with client-side XSLT, you need MXSML 3.0 or later, which supports the XSLT standards, and Internet Explorer 5.0 and later, which supports XML data islands.

The HTML file in the next topic, DataIsland.htm, shows how to use XML data islands in a client-side XSLT application. An XML data island is represented by an <XML>...</XML> tag in an HTML page. Remember that XML data islands are HTML elements, or DHTML DOM objects when used in a script. This HTML tag can have two attributes: ID and SRC. ID serves as the identifier of the DOM object, and SRC can be used to load an XML document from a specified file.

In this example, there are two XML data islands—that is, two <XML>...</XML> tags. One is identified as book_catalog and holds the XML source document. The other is identified as catalog_filter and holds the XSLT style sheet used to transform book_catalog. Here one XML document (catalog_filter) is hard-coded into the XML data island. The other (book_catalog) reads from an external XML file (book_catalog.xml). To retrieve the XML DOM object from an XML data island, use the XMLDocument property on the data island. For example, catalog_filter.XMLDocument returns the XML DOM object for the XSLT style sheet.

In the DataIsland.htm sample file, the JavaScript code for transforming the
XSLT is quite similar to the code used in the server version in *Using ASP with XSLT Processor and Template Objects*, with a few notable differences. The showGenre() function (renamed from main() in the server version) takes the genre name as a parameter, which lets it modify the contents of a DIV element (called catalog_table) when the select list box's value is changed. This function passes references to the two data islands book_catalog and catalog_filter directly, rather than passing file names. The transformData() function returns the resulting transformed data as an XML object, rather than sending the output to the Response object. This is because the Server.Response object can take an XML DOM stream as input, but the innerHTML property can only take a string.

Both loadSource() and getProcessor() have also been modified to take into account the data island source of the information, and getProcessor() now uses a local variable to store the XSLT processor object, rather than a session variable. Otherwise, the two are quite similar.

The only changes from the XSLT in *Using ASP with XSLT Processor and Template Objects* are as follows:

- The result is now contained in a <DIV> element, rather than in the full <HTML> wrapper. This has already been declared on the client.
- The style sheet has been moved out to the containing HTML document, to simplify the code.
- The onchange attribute of the <SELECT> list box has been changed from a URL call to the command showGenre(this.value). This command will update the table to reflect the selected genre.

This example brings up a number of points about working with XSLT on the client:

- The JavaScript functions discussed here are highly modular. Only the showGenre function actually contains code specific to a single style sheet and data source. By maintaining this modularity, you can create other functions that use different sources or transformations in the same page—a navigation bar, for example, or a menu—without functions that transform each component separately.
- You can use the SRC attribute on data islands to load both data and transformations from the server in response to changing conditions on the
client. For example, an XML document containing sales figures could have a default table translation, but could be converted into a chart by changing the src attribute of the transformation data island to reflect a chart.xsl document on the server. Then, the transformation could be applied using the same interface described above. Similarly, live data from a SQL Server 2000 URL call could be retrieved into a data island via the src attribute (the specific parameters determined from form elements), and could then be reformatted by the existing transformation on the client.

- Use parameters in the XSLT transformation to change the nature of the transformation. Do not try to modify the style sheet directly. The parameters of a style sheet are much like the public properties of a COM object—they represent an interface into the style sheet, which otherwise should be treated as a black box.
- The Sample HTML File for XML Data Islands follows.
Sample HTML File for XML Data Islands

XML File (book_catalog.xml)

```xml
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <title>XML Developer's Guide</title>
    <author>Gambardella, Matthew</author>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML</description>
  </book>
  <book id="bk102">
    <title>Midnight Rain</title>
    <author>Ralls, Kim</author>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies, an</description>
  </book>
  <book id="bk103">
    <title>Maeve Ascendant</title>
    <author>Corets, Eva</author>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology society in</description>
  </book>
  <book id="bk104">
    <title>Oberon's Legacy</title>
    <author>Corets, Eva</author>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-03-10</publish_date>
    <description>In post-apocalypse England, the mysterious agent</description>
  </book>
  <book id="bk105">
    <title>The Sundered Grail</title>
    <author>Corets, Eva</author>
    <genre>Fantasy</genre>
    <price>5.95</price>
</catalog>
```
<publish_date>2000-09-02</publish_date>
<description>When Carla meets Paul at an ornithology conference,
temper fly as feathers get ruffled.</description>
</book>
<book id="bk108">
<title>Creepy Crawlies</title>
<author>Knorr, Stefan</author>
<genre>Horror</genre>
<price>4.95</price>
<publish_date>2000-12-06</publish_date>
<description>An anthology of horror stories about roaches, centipedes,
and other insects.</description>
</book>
<book id="bk109">
<title>Paradox Lost</title>
<author>Kress, Peter</author>
<genre>Science Fiction</genre>
<price>6.95</price>
<publish_date>2000-11-02</publish_date>
<description>After an inadvertant trip through a Heisenberg Uncertainty Device,
James discovers the problems of being quantum.</description>
</book>
<book id="bk110">
<title>Microsoft .NET: The Programming Bible</title>
<author>O'Brien, Tim</author>
<genre>Computer</genre>
<price>36.95</price>
<publish_date>2000-12-09</publish_date>
<description>Microsoft's .NET initiative is explored in detail.</description>
</book>
<book id="bk111">
<title>MSXML3: A Comprehensive Guide</title>
<author>O'Brien, Tim</author>
<genre>Computer</genre>
<price>36.95</price>
<publish_date>2000-12-01</publish_date>
<description>The Microsoft MSXML3 parser is covered in detail,
exploring XML DOM interfaces, XSLT processing, and more.</description>
</book>
<book id="bk112">
  <title>Visual Studio 7: A Comprehensive Guide</title>
  <author>Galos, Mike</author>
  <genre>Computer</genre>
  <price>49.95</price>
  <publish_date>2001-04-16</publish_date>
  <description>Microsoft Visual Studio 7 is explored in depth, l</description>
</book>

HTML File (DataIsland.htm)

<HTML>
<HEAD>
<TITLE>Untitled</TITLE>
<STYLE>
  .catalog_genre_head {background-color:darkGreen;font-size:24
  .catalog_head {background-color:green;font-size:18pt;color:w
  .catalog_row0 {background-color:lightGreen;}
  .catalog_row1 {background-color:white;}
  .catalog_row_end {background-color:darkGreen;}
</STYLE>
</HEAD>

<BODY>
<H1>Scootney Press Book Catalog</H1>
<P>Select a genre to see all books in that genre:</P>
</DIV id="catalog_table"</DIV>

<xml id="book_catalog" src="book_catalog.xml">
</xml>

<xml id="catalog_filter">
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt"
    version="1.0">
    <xsl:output method="xml" omit-xml-declaration="yes"/>
    <xsl:param name="selected_genre" select="'all'"/>
    <xsl:template match="/"
        <div>
            <form method="post" action="catalog.asp">
                <select name="genre" value="{$selected_genre}" o
                <option value="all"/><xsl:if test="$selected_"<option value="Computer"/><xsl:if test="$sel
<option value="Fantasy"><xsl:if test="$selected_genre='Fantasy'"><xsl:attribute name="selected">Selected</xsl:attribute></xsl:if>Fantasy</option>
<option value="Horror"><xsl:if test="$selected_genre='Horror'"><xsl:attribute name="selected">Selected</xsl:attribute></xsl:if>Horror</option>
<option value="Romance"><xsl:if test="$selected_genre='Romance'"><xsl:attribute name="selected">Selected</xsl:attribute></xsl:if>Romance</option>
<option value="Science Fiction"><xsl:if test="$selected_genre='Science Fiction'"><xsl:attribute name="selected">Selected</xsl:attribute></xsl:if>Science Fiction</option>
</select>
</form>
<br/>
<xsl:apply-templates select="catalog"/>
</div>
</xsl:template>

<xsl:template match="catalog">
<table class="catalog_table">
<xsl:apply-templates select="book[($selected_genre='all') or ($selected_genre=./genre)]">
<xsl:sort select="title"/>
</xsl:apply-templates>
</table>
</xsl:template>

<xsl:template match="book">
<xsl:if test="position()=1">
<tr class="catalog_genre_head"><td colspan="6">
<xsl:choose>
<xsl:when test="$selected_genre='all'">All Genres</xsl:when>
<xsl:otherwise>Genre: <xsl:value-of select="genre"/>
</xsl:otherwise>
</xsl:choose>
</td></tr>
<tr class="catalog_head">
<td>#</td>
<td>Title</td>
<td>Author</td>
<td>Publication Date</td>
<td>Description</td>
<xsl:if test="$selected_genre='all'">
<td>Genre</td>
</xsl:if>
</tr>
<xsl:if test="position() mod 2 = 0">
<tr class="catalog_row{position() mod 2}" class="catalog_row">
<td><xsl:value-of select="position()"/></td>
<td class="catalog_cell"><xsl:value-of select="title"/></td>
<td class="catalog_cell"><xsl:value-of select="author"/></td>
<td class="catalog_cell"><xsl:value-of select="publish_date"/></td>
<td class="catalog_cell"><xsl:value-of select="description"/></td>
<xsl:if test="$selected_genre='all'">
<td class="catalog_cell"><xsl:value-of select="genre"/></td>
</xsl:if>
</tr>
</xsl:if>
</xsl:if>
function loadSource(sourceObj){
    var xmlDoc=new ActiveXObject("Msxml2.FreeThreadedDOMDocument.5.0");
    xmlDoc.async=false;
    xmlDoc.load(sourceObj.XMLDocument);
    return xmlDoc;
}

var table_proc=null;

function getProcessor(transformObj){
    if (table_proc==null){
        var xslDoc=new ActiveXObject("Msxml2.FreeThreadedDOMDocument";
        var xslTemplate=new ActiveXObject("Msxml2.XSLTemplate.5.0");
        xslDoc.async=false;
        xslDoc.load(transformObj.XMLDocument);
        xslTemplate.stylesheet=xslDoc;
        xslProcessor=xslTemplate.createProcessor();
        table_proc=xslProcessor;
    }
    else {
        xslProcessor=table_proc;
    }
    return xslProcessor;
}

function transformData(srcDoc,processor){
    var resultDoc=new ActiveXObject("MSXML.DOMDocument");
    processor.input=srcDoc;
    processor.output=resultDoc;
    processor.transform();
    return resultDoc;
}

function showGenre(genre){
    var srcDoc=loadSource(book_catalog);
    var processor=getProcessor(catalog_filter);
    processor.addParameter("selected_genre",genre);
var rsltDoc=transformData(srcDoc,processor);
catalog_table.innerHTML=rsltDoc.xml
}

showGenre("all");

</SCRIPT>

</BODY>
</HTML>
Controlling Output Format with the media-type Attribute

XSLT uses the `<xsl:output>` element to determine whether the output produced by the transformation is conformant XML (`<xsl:output method="xml"/>`), valid HTML (`<xsl:output method="html"/>`), or unverified text (`<xsl:output method="text"/>`). However, this can cause versioning problems, especially on the server, because Microsoft® Internet Information Services (IIS) automatically sets the output of ASP files to a mime-type of text/html to ensure that the browser interprets material sent to it as Web pages.

Although these types can determine the necessary headers, if any, the fact that a document is in XML format does not necessarily identify its output type. For example, a graphic file might be in Scalable Vector Graphics format, or a Wireless message might be in WML format. These files must inform ASP that they are both of type XML and have the appropriate mime-type (such as image/svg for SVG or text/vnd.wap.wml for WML).

The `<xsl:output>` command includes one additional attribute: `media-type`. This attribute is not used directly by the XSLT parser in generating output, but it can be read from the style sheet through the DOM, and used to set the `Response.ContentType` property. `ContentType` creates a header in the returning HTTP message. This header tells the browser how it should handle given mime-types. The following style sheet is an example.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" media-type="text/vnd.wap.wml"/>
<xsl:template match="/"/>
</xsl:stylesheet>
```

In this example, the `media-type` attribute has a value of "text/vnd.wap.wml". This indicates that the output from the style sheet should be a WML document. The `getMimeType()` function takes an XSLT processor object, and retrieves the
mimeType from the <xsl:output> node. The full code for this example is in catalog.asp, in Sample ASP Script for XSLT Processor Objects.

```javascript
function getMimeType(processor)
{
    var stylesheet=processor.stylesheet;
    var localTypeNode=stylesheet.selectSingleNode("//xsl:output/@media-type");
    if (''+localTypeNode=='undefined'){
        var localMethodNode=stylesheet.selectSingleNode("//xsl:output/@method");
        if (''+localMethodNode=='undefined'){
            mimeType="text/html";
        }
        else {
            localMethodName=localMethodNode.text;
            switch(localMethodName){
                case "text":mimeType="text/plain";break;
                case "html":mimeType="text/html";break;
                case "xml":mimeType="text/xml";break;
            }
        }
    }
    else {
        mimeType=localTypeNode.text;
    }
    return mimeType;
}
```

You can then set the ContentType property by assigning the result of the getMimeType function to it, as follows:

```javascript
Response.ContentType=getMimeType(processor)
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Controlling White Space with XSLT

This section contains the following topics.

- How White Space is Handled in XML Files
- How the MSXML Processor Parses White Space
- Inserting White Space Using `<xsl:text>`
- Preserving or Stripping White Space
- Example of `<xsl:preserve-space>` and `<xsl:strip-space>`
- Overriding `<xsl:preserve-space>` and `<xsl:strip-space>`
- Preserving Markup Characters by Using CDATA Sections
- Stripping White Space Using `normalize-space()`
- Controlling White Space with the DOM
- How the Browser Handles the `preserveWhiteSpace` Property
- Controlling White Space Using the Style Sheet
- Controlling White Space Using the MSXML Processor

See Also

`<xsl:strip-space>` Element | `<xsl:preserve-space>` Element
How White Space is Handled in XML Files

White space refers to characters in an XML document, such as spaces and tabs, that do not appear when the document is represented in a text editor like Notepad, or in an application like Microsoft® Internet Explorer.

<table>
<thead>
<tr>
<th>Character</th>
<th>Entity representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>&amp;32; (hexadecimal &amp;x20;)</td>
</tr>
<tr>
<td>Carriage return</td>
<td>&amp;13; (&amp;xD;)</td>
</tr>
<tr>
<td>Tab</td>
<td>&amp;9; (&amp;x9;)</td>
</tr>
<tr>
<td>Line feed (newline)</td>
<td>&amp;10; (&amp;xA;)</td>
</tr>
</tbody>
</table>

White space includes the tab, space, and line break characters. Any series of white space characters between elements is considered a single white space node. For example,

<books>
  <book>
</book>
</books>

Suppose that between the above elements is a line break, a tab, and some space characters. Together, these characters form a single white space node. If this white space node is stripped, the following results:

<books><book>

If the white space node is normalized, the following single space results:

<books>  <book>

If you want the processor to produce human-readable output such as a standard HTML listing, you should preserve the white space nodes that are contained in the XML input file and the XSLT file.

To control white space, use the following XSLT constructs:

&xA0; or &160;
A non-breaking space (&#160;, hexadecimal &#xA0;, or &nbsp; in HTML) is not considered white space, because it is not an allowable place to break to a new line.

In HTML, a non-breaking space is useful for formatting. For example, if a table cell is completely empty, the inner cell border is not drawn. If you want the border drawn even though the cell has no content, use a non-breaking space in the XSLT, as follows: `<td>&#160;</td>`

An alternative to using &#xA0; or &#160; in XSLT is to define the &nbsp; entity at the beginning of the XSLT file, as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [<!ENTITY nbsp "&#160;">]>
```

This allows you to use &nbsp; throughout the document. It will be recognized the same way as an HTML &nbsp; entity.

You can control white space output in an XML document by using the Document Object Model (DOM) or XSLT features.

Any XML document exercises some control over white space. The parser handles white space in an XML document according to the xml:space attribute and the default white space rules, before the document is processed by the XSLT processor.

The XML 1.0 specification imposes the following white space rules.

- The parser normalizes newline characters specific to an operating system into true newline characters (hex x0A, or decimal 10). This is because different operating systems represent line breaks in different ways: for example, as true newlines, carriage returns, line feed/carriage return character pairs, and so on.
- The parser normalizes the values of attributes (other than CDATA-type attributes), by replacing multiple consecutive occurrences of white space with a single space. For example, an attribute value such as "text  text" (with four intervening spaces embedded in the value is passed from the parser) as "text  text". The multiple spaces in the original value are
replaced with a single space.

- If an xml:space attribute in the source XML document or style sheet conflicts with explicit XSLT white space handling, the behavior associated with that xml:space attribute always takes precedence.

According to the XSLT specification, the XSLT processor merges adjacent text nodes into a single text node. If a text node (following any merging that occurs) consists of white space only, the containing element is compared to the list of elements in any <xsl:strip-space> elements in the style sheet. If the containing element appears in such a list, the text node with white space only is removed from the result tree.

This applies only to insignificant white space—that is, white space between, not within, elements. Use the XSLT normalize-space() function to normalize white space within elements.

White space is handled by the built-in parser of MSXML, as well as by its built-in XSLT processor.

**See Also**

How the MSXML Processor Parses White Space

When MSXML parses an XML document, all white space in the document is preserved and passed to a downstream application.

The built-in XSLT processor is not the downstream application that receives the output of the MSXML parser. This can be confusing to developers who are new to MSXML. The following diagram shows the intermediate steps that receive the output of the parser, build a Document Object Model (DOM) tree, and cache this tree in memory for ready access later.

The input document that the XSLT processor works with, therefore, is actually the DOM tree in memory. By default, the Microsoft DOM tree builder removes extraneous white space from the document passed to it by the parser. Any
content that is not in the scope of an xml:space attribute whose value is "preserve" is extraneous in this context.

To override this default and retain the original document's extraneous white space in the DOM tree, you must set the preserveWhiteSpace property to True at the time the DOM is loaded. In other words, you cannot preserve the original document's extraneous or insignificant white space using XSLT features alone. By the time the XSLT processor sees a version of the original the document, all such white space has, by default, been stripped.

The only way to set the preserveWhiteSpace property to True is to process the DOM through script or a programming language. For more information, see White Space and the DOM.
Inserting White Space Using
<xsl:text>

The <xsl:text> element sends literal text to the output stream. Because this element forces any white space information within it to be retained, it can be used to send multiple white space characters as part of a string of text. Each white space character is entered directly by keyboard, or represented as an entity.

How the XSLT processor treats white space in the document depends on the white space it receives from the Microsoft® Document Object Model (DOM) processor. Keep this in mind as you follow the examples in this section.

Example

In this example, the first pair of <xsl:value-of> elements are separated by four spaces. Because this text node contains white space only, it is stripped from the result tree by the XSLT processor. In the second pair of <xsl:value-of> elements, however, the four spaces are preserved by the <xsl:text> element. The following is the HTML generated by these sections of the style sheet.

White space stripped:&lt;-here<br>
White space preserved:    &lt;-here

Note that four spaces are retained in the second line. However, Internet Explorer, following standard browser behavior, normalizes multiple spaces in an HTML document to a single space on output.

XML File

None. The XSLT file calls itself. View the XSLT file in Internet Explorer.

XSLT File (whitespace1.xsl)

<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="whitespace1.xsl" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
Formatted Output

White space stripped: <-here
White space preserved: <-here

Processor Output

<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<title>Test for WhiteSpace1.xsl</title></head>
<body>
White space stripped:&lt;-here<br> White space preserved: &lt;
</body>
</html>

See Also

Character and Entity References
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Preserving or Stripping White Space

XSLT can distinguish nodes that contain white space intermingled with other characters. The white space is considered inseparable from the other text in the node.

For nodes that contain nothing but white space, the `<xsl:preserve-space>` and `<xsl:strip-space>` elements determine how the nodes are output.
Preserving White Space with `<xsl:preserve-space>`

The `<xsl:preserve-space>` element provides a list of those elements in the source document whose white space must be preserved on output. The `<xsl:preserve-space>` element is always an empty, top-level element—for example, a child of the `<xsl:stylesheet>` element in an XSLT style sheet.

The following is the general syntax.

```xml
<xsl:preserve-space elements="elem1 elem2...elemN" />
```

In this syntax, `elem1` through `elemN` are the names (without the enclosing `<` and `>` delimiters) of all elements whose white space must be preserved.

You can specify all of the elements in the source document with the asterisk operator ("*"), as follows:

```xml
<xsl:preserve-space elements="*" />  
```

Because all of the content of an XML document is, by default, preserved, `<xsl:preserve-space>` is only useful when you have used `<xsl:strip-space>` to override the default behavior. In such cases, use `<xsl:preserve-space>` to identify exceptions to the explicit stripping of white space.

**Note** The Microsoft® XSLT processor will process all of a document's white space only if the `preserveWhiteSpace` property has been set to True prior to loading the document into the DOM. For more information, see [How the MSXML Processor Parses White Space](https://msdn.microsoft.com/en-us/library/ff693574(v=vs.85).aspx).
Removing White Space with `<xsl:strip-space>`

The `<xsl:strip-space>` element provides a list of elements in the source document that have content which must be removed from the output tree. The `<xsl:strip-space>` element is an empty, top-level element.

The following is the general syntax.

```
<xsl:strip-space elements="elem1 elem2...elemN" />  
```

In this syntax, `elem1` through `elemN` are the names (without the enclosing `<` and `>` delimiters) of all elements whose white space must be removed.

You can specify all of the elements in the source document with the asterisk operator ("*"), as follows:

```
<xsl:strip-space elements="*" />  
```

If an element appears in both an `<xsl:strip-space>` and `<xsl:preserve-space>` list, the specification that appears last applies. Therefore, a typical sequence of these two elements in an XSLT style sheet is as follows:

```
<xsl:strip-space elements="*" />  
<xsl:preserve-space elements="elem1 elem2...elemN" />  
```

If this order were reversed, the explicit `preserve-space` settings for `elem1`, `elem2`, and so on would be overridden by the global `strip-space` setting.
Example of `<xsl:preserve-space>` and `<xsl:strip-space>`

You can use the `<xsl:preserve-space>` and `<xsl:strip-space>` elements to preserve and strip white space from XML documents.

Example

The `strippreserve.xml` document below contains `<code>` and `<block>` elements that have white space in the form of tabs, spaces, and newline characters.

The `strippreserve.xsl` style sheet below does the following:

- Preserves the extraneous white space in the `<code>` elements.
- Strips the extraneous white space from the `<block>` elements.
- Encloses the content of each `<code>` and `<block>` element in square brackets ([ ]).
- Uses the XSLT `position()` function to provide a numbered label for each `<code>` or `<block>.
- Uses the XPath `translate()` function to converts invisible white space to a visible character, one character at a time. This enables you to see the specific white characters in the `<code>` and `<block>` elements.

The results are as follows:

- Each space character appears in the result tree as a hyphen (-).
- Each newline character appears in the result tree as the letter N.
- Each carriage return appears in the result tree as the letter R.
- Each tab appears in the result tree as the letter T.
- The first `<code>` element contains a single space, denoted by the hyphen in the [ ] characters. The first `<block>` element also contains a single space in the source document. However, the `<xsl:strip-space>` setting has caused this space to be removed from the result tree.
- In the second `<code>` and `<block>` elements, no white space has been removed. If an element has any content that is not white space, white space
will not be stripped, even if `<xsl:strip-space>` is set for that element.

**Note**  This example uses only XSLT to strip and preserve white space. You can also strip or preserve white space by manipulating the Document Object Model (DOM). For more information about this approach, see [White Space and the DOM](#).

**XML File (strippreserve.xml)**

Copy this text to a file. Then, find each `tabhere` and replace it with a tab character (that is, press the TAB key).

```xml
<?xml-stylesheet type="text/xsl" href="strippreserve.xsl" ?>
<document>
  <block>
    Some text
    tabhere
    Some text
    tabhere</block>
  <code>
    Some text
    tabhere
    Some text
    tabhere</code>
</document>
```

**XSLT File (strippreserve.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Retain white space within all <code> elements -->
  <xsl:preserve-space elements="code"/>
  <!-- ... but strip it from all <block> elements -->
  <xsl:strip-space elements="block"/>

  <xsl:template match="/">
    <html>
      <head><title>Test: Stripping/Preserving White Space</title></head>
      <body>
        <h4>Code blocks:</h4>
        <!-- <pre> element forces all output characters to be same width -->
        <pre>
```
Formatted Output

Code blocks:

"Code" #1: [-]

"Code" #2: [NTNSome-textTNTNT]

Normalized blocks:

"Block" #1: []

"Block" #2: [NTNSome-texttabhereNTNT]

Processor Output

<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<title>Test: Stripping/Preserving White Space</title></head>
<body>
<h4>Code blocks:</h4>
<pre>
"Code" #1: [-]<br>
"Code" #2: [NTNSome-textTNTNT]<br></pre>
<h4>Normalized blocks:</h4>
<pre>
"Block" #1: []<br>
"Block" #2: [NTNSome-texttabhereNTNT]<br></pre>
"Block" #2: [NTNSome-textTNTNT]<br></pre></body></html>
Overriding `<xsl:preserve-space>` and `<xsl:strip-space>`

Use the `xml:space` attribute to override the effect of `<xsl:strip-space>` or `<xsl:preserve-space>`.

For an example of this, consider `strippreserve.xsl` in Example of `<xsl:preserve-space>` and `<xsl:strip-space>`. Even though the `<block>` elements in this file were specified in the `<xsl:strip-space>` element list, you could force the preservation of four spaces by including the following in that template rule.

```xml
<block><xsl:text xml:space="yes"> </xsl:text></block>
```

This example is artificial in that it forces the insertion of white space within the style sheet. It does not restore or preserve white space in the original document. To do that, you must override the default stripping of extraneous white space by setting the `preserveWhiteSpace` property to `True` at the time the document is loaded into the Document Object Model (DOM).

See Also

- [Preserving or Stripping White Space](#)
- [How the MSXML Processor Parses White Space](#)
- [White Space and the DOM](#)
Preserving Markup Characters by Using CDATA Sections

CDATA sections within an XML document can contain markup elements common to XML, without affecting whether the document is well-formed. For example, if you want to include a mathematical XML application using the < and > characters to refer to less-than and greater-than, you can use a CDATA section.

To specify that a given element in the result tree will contain a CDATA section, include the element name as a value of the cdata-section-elements attribute of the <xsl:output> element. The CDATA section created in this way will preserve white space and character entities.

Example

The XSLT processor scans the result tree, making a CDATA section of each item in the cdata-section-elements attribute.

To create the CDATA section, the elements listed as values of the cdata-section-elements attribute must be elements in the result tree, not the source tree. In the preceding examples, the Microsoft JScript® code appears as a text node within the source tree's <scriptcode> element, but the cdata-section-elements attribute refers to and affects <script> elements in the result tree.

HTML does not recognize CDATA sections. Do not use this option when generating HTML.

XML File (cdata.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="cdata.xsl" ?>
<document>
<scriptcode language="JScript">
function message(msg){
alert(msg);
}
```
XSLT File (cdata.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output cdata-section-elements="script"/>
    <xsl:template match="scriptcode">
        <document>
            <script language="JScript">
                <xsl:value-of select="."/>
            </script>
        </document>
    </xsl:template>
</xsl:stylesheet>

Formatted Output

None (blank).

Processor Output

<?xml version="1.0" encoding="UTF-16"?><document><script language="J" function message(msg){
    alert(msg);
}]]></script></document>

See Also

CDATA Sections
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Stripping White Space Using 
normalize-space()

XSLT provides a built-in function, normalize-space(), that strips leading and 
trailing white space from a string and normalizes multiple successive white 
space characters to a single space. It takes one argument: a string or node-set 
(node-sets are converted to strings as necessary). If the argument is omitted, 
normalize-space() assumes the context node.

The following is an XML fragment with leading newline and tab characters 
following the <generouswhitespace> start tag. There are 10 trailing spaces and 
a newline preceding the </generouswhitespace> end tag, as follows:

<generouswhitespace>
There is a l o t of white space here!
</generouswhitespace>

The white space in this fragment can be normalized with the following XSLT 
template rule, which encloses the text content of the <generouswhitespace> 
element in brackets ([]).

<xsl:template match="generouswhitespace">
  [<xsl:value-of select="normalize-space(.)" />]
</xsl:template>

The output appears as follows.

[There is a l o t of white space here!]

All leading white space has been trimmed, and each block of extraneous white 
space within the text node has been normalized to a single space.
Controlling White Space with the DOM

When a text file is opened with the xmlDoc.load method or the xmlDoc.loadXML method (where xmlDoc is an XML DOM document), the parser strips most white space from the file, unless specifically directed otherwise. The parser notes within each node whether one or more spaces, tabs, newlines, or carriage returns follow the node in the text by setting a flag. This method is efficient, reducing both the size of each XML file and the number of calculations required to redisplay the XML in a browser. However, because this information is lost, an XML document stored in this manner can lose formatting information in its content. Tabs, in particular, can be lost, because they are not formally recognized in the default mode as anything but white space.

XSLT uses the DOM, not the source document, to guide the transformation. Because the white space has already been stripped to process the XML into the DOM, white space characters are lost even before the transformation takes place. Most of the XSLT-related methods for specifying white space in the source data document or style sheets are applied too late to make a difference in formatting.

The preserveWhiteSpace property tells the XML parser whether or not to convert white space from the initial source file that acts against the XML DOM. The default value of this property is False: Extraneous white space characters are stripped, and the file is reduced to the smallest possible stream. If you want the XML document to retain extraneous white space when converted to a DOM document, you must set preserveWhiteSpace to True prior to loading the file into the DOM.

If you set the preserveWhiteSpace property from True to False and then back to True for a given DOM document, the spaces will not reappear. Setting the property to False removes the white space from the DOM, which cannot reconstruct it.

If you are working with XML as a data format streamed to some other process, disable preserveWhiteSpace by setting it, or allowing it to default, to False. If retaining positional information is important—for example, in conversions to
non-XML formats like tab-separated data—set preserveWhiteSpace to True. Be aware that this option increases the number of characters, and places more demands on the browser.

**Example**

This example shows how white space can be programmatically controlled, using the DOM. To run the example, view the HTML file in Internet Explorer.

In the first block of results, the content of each original `<whitespace>` element appears on a separate line. In the second block, where the preserveWhiteSpace property is set to False, all the content appears on a single line. In addition, the first block is indented, while the second block is not. The indents and line breaks in the first block are a result of newline and tab characters in the source document between the boundaries of the `<whitespace>` elements. This white space is affected by the preserveWhiteSpace property.

The white space within the `<whitespace>` elements is not affected by the value of the preserveWhiteSpace property. To remove white space only text children of elements, use the XPath `normalize-space()` function.

The example uses the following three files.

**striporpres.xml**
The XML source document. Each of the `<whitespace>` elements contains different kinds of white space, including tags, spaces, and newlines.

**striporpres.xsl**
An XSLT style sheet that makes invisible white space visible for demonstration purposes. The `striporpres.xsl` style sheet consists of two template rules. The first rule applies to the source document root node, instantiating an HTML `<pre>` element node in the result tree. The transformed `<whitespace>` elements are placed in this node, which is handled by the second template rule. As in the `<xsl:strip-space>` and `<xsl:preserve-space>` example, the XPath `translate()` function is used to make the invisible white space visible.

**striporpres.htm**
An HTML file that contains Microsoft® JScript®. The `preserveStripPreserve` JScript function is called when the file is loaded. This function performs the following tasks:
Sets the preserveWhiteSpace property.
- Loads `striporpres.xml` and `striporpres.xsl` into separate DOMDocument objects.
- Alternately sets the preserveWhitespace property of the DOMDocument object to True or False.

The script also uses the transformNode method to apply the `striporpres.xls` style sheet to the `striporpres.xml` document, and assigns the resulting string to the `innerHTML` property. This property is used to display the results in Internet Explorer.

**XML File (striporpres.xml)**

There is no `<?xml-stylesheet?>` processing instruction. Instead, a style sheet is applied programmatically to the document using the transformNode method. Unlike a pure XSLT solution, this technique allows you to set and reset the preserveWhiteSpace property.

Copy this text to a file. Then, find each `tabhere` and replace it with a tab character (that is, press the TAB key).

```xml
<?xml version='1.0'?><whitespaceTest>
  <whitespace>Tabs[tabheretabhere]</whitespace>
  <whitespace>Spaces[ ]</whitespace>
  <whitespace>Newlines[ ]</whitespace>
</whitespaceTest>
```

**XSLT File (striporpres.xsl)**

```xml
<?xml version='1.0'?><xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <pre>
      <xsl:apply-templates />
    </pre>
  </xsl:template>
  <xsl:template match="whitespace">
    <!-- Use translate() XPath function to convert
```
character X to character Y. -->
<xsl:value-of select="translate(.,', &#10; &#13; &#9;','-NRT')"/>
</xsl:template>

</xsl:stylesheet>

HTML File (striporpres.htm)

<html>
<head>
<title>Demo: Controlling White Space Output via the DOM</title>
<script language='JScript'>
<!--
function preserveStripPreserve() {
    // Associate the result tree object with any element(s) whose
    // id attribute is "testResults."
    var objResTree = document.all['testResults'];
    // Declare two new Mxml DOMDocument objects.
    var objSrcTree = new ActiveXObject('Msxml2.DOMDocument');
    var objXSLT = new ActiveXObject('Msxml2.DOMDocument');
    // Load the two DOMDocuments with the XML document and the
    // XSLT style sheet.
    objSrcTree.preserveWhiteSpace = true;
    objSrcTree.load('stripOrPres.xml');
    objXSLT.load('stripOrPres.xsl');
    // Use the transformNode method to apply the XSLT to the XML.
    strResult = objSrcTree.transformNode(objXSLT);
    // Now reset preserveWhiteSpace to false, and re-load the source...
    objSrcTree.preserveWhiteSpace = false;
    objSrcTree.load('stripOrPres.xml');
    // ...and rerun the transform. Note the concatenation of the
    // this transformation's result tree to the one created when
    // preserveWhiteSpace was true.
    strResult = strResult + objSrcTree.transformNode(objXSLT);
    // Assign the resulting string to the result tree object's
    // innerHTML property.
    objResTree.innerHTML = strResult;
    return true;
}
-->
<script>
<body onload='preserveStripPreserve()'>
<div id='testResults'></div>
</body>
</html>

Formatted Output

Tabs[TT]
Processor Output

No XSLT processor output stream is available in Internet Explorer for this example.

See Also

Example of `<xsl:preserve-space>` and `<xsl:strip-space>` | Stripping White Space Using `normalize-space()`
How the Browser Handles the preserveWhiteSpace Property

If you have HTML presented as well-formed XML, the preserveWhiteSpace property should have no impact on the Web page displayed in the browser. HTML browsers automatically remove excess white space upon display, unless it is contained in `<pre></pre>` (preserve) or `<xmp></xmp>` (example) tags.

In the MSXML processor, if preserveWhiteSpace is set or defaulted to False, the XSLT white space features will have no effect. In addition, for Internet Explorer 5.0 and later, the XML parser is set to ignore unnecessary white space from XSLT transformations. (In other words, preserveWhiteSpace is set to False). To see these results, you must first transform the output through an Active Server Page (ASP) or Microsoft JScript® script, or a similar mechanism. You cannot rely on XSLT alone to see the result of most of the white space options.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Devel
Controlling White Space Using the Style Sheet

White space only nodes in the XSLT style sheet, as opposed to those in the source document, are stripped by the XSLT processor.

It can be an advantage to have extraneous white space stripped from a style sheet. You can format the style sheet for easy readability and maintenance, knowing that extra tabs, newlines, and so on will not appear in the result tree.

However, this can be a disadvantage when you need white space of a particular kind in the result tree. For example, if you had to insert a blank space in a table cell, you might use the following.

<td> </td>

However, if your style sheet places this sequence of text in the result tree, it will appear as follows.

<td></td>

Not all browsers will display a NULL cell properly.

Use the <xsl:text> element around any white space only text nodes.

<td><xsl:text> </xsl:text></td>

For more information, see Inserting White Space Using <xsl:text>.
Controlling White Space Using the MSXML Processor

In most cases, the behavior of the MSXML processor conforms to white space handling rules imposed by the XML and XSLT specifications.

The following are exceptions.

- Adjacent text nodes are not treated as one node. Adjacent text nodes can appear if:
  - The user inserts them into the tree through the DOM API.
  - CDATA sections border #PCDATA nodes.
  - An entity reference containing text borders a #PCDATA or CDATA node. Character references, such as &\#10; (to represent a newline), are automatically expanded by the parser and handled correctly.

Few style sheets will encounter any of these conditions. Furthermore, even if adjacent text nodes exist, users seldom query for text nodes directly. In most cases, users obtain text nodes by referring to element and attribute nodes.

- When the preserveWhitespace property is False, the white space stripped by the DOM is reduced to flags that mark where white space existed. These flags are used to output extra white space when <xsl:copy-of> is used, or when the text value of a node is retrieved. These flags act as virtual nodes that cannot be queried, but nevertheless behave like text nodes with a value of &\#10; (newline) on output. This does not occur if preserveWhitespace is True, because flags are not used.

- CDATA nodes containing only white space are never stripped.
How Do I?

The following tutorials demonstrate some common XSLT tasks.

- Initiate XSLT in a Script
- Represent Data in XML
- Reference Data Values
- Use CSS in XSLT
- Perform Iterations
- Operate on Data Values
- Control the Flow of Transformation
- Process Multiple XML Documents

This set of task-based, hands-on exercises shows you how to create an XSLT solution. The scenario is to design a simple regional sales reporting system for a fictitious book publishing enterprise, Scootney Publishing. This involves creating an XML document to represent the quarterly sales data, specifying template rules in an XSLT style sheet to handle business logic, and presenting the sales results in a format that can be viewed from the Web.

You will work with the following three files throughout the exercises.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales.xml</td>
<td>An XML document that contains the input data.</td>
</tr>
<tr>
<td>Transform.xsl</td>
<td>An XSLT style sheet used to transform the input data.</td>
</tr>
<tr>
<td>Ancillary.xml</td>
<td>An XML file from which the XSLT style sheet extracts additional data.</td>
</tr>
</tbody>
</table>

Bold text will be used to highlight parts of the code that are relevant and important in each step. Files created at the end of each step might represent only a part of the final solution. As a reference, the complete listings of these three files, plus the output in HTML, are shown in Final Results of the XSLT How Do I? Examples, at the end of this section.
Before you start

To complete the exercises, you need the following:

- MSXML 4.0 or later, available from the MSDN® Online Downloads.
- Microsoft Internet Explorer 5.0 or later.
- An HTML or text editor, such as Notepad.
- Optionally, Internet Explorer Tools for Validating XML and Viewing XSLT Output. This utility allows you to access the View XSL Output and Validate XML commands by right-clicking on Internet Explorer.
Initiate XSLT in a Script

Currently, the default XML parser for Internet Explorer is MSXML 2.0 or MSXML 3.0, depending on the version of Internet Explorer. With the help of the xmslinst.exe utility, you can change the default parser to MSXML 2.6 or MSXML 3.0. However, this can often cause unintended side-effects for some applications. Therefore, such a practice is not enabled for MSXML versions 4.0 and later.

Until Internet Explorer ships with MSXML 4.0 or later as its default XML parser, the MSXML 4.0 and later features are available in Internet Explorer only via scripting, when an XML DOM object is instantiated using the MSXML4-specific ProgID. This means that you cannot use `<?xml-stylesheet type="text/xsl" href="Products.xsl"?>` in an XML document to invoke MSXML 4.0 functionality.

However, you can write either server-side or client-side script to use MSXML 5.0 instead. In this SDK, we provide two ways to do this:

- [Initiate XSLT from an HTML Page](#)
- [Initiate XSLT from a Command Prompt](#)

The following examples illustrate each method. You can use the code in these examples to test other samples in this SDK. Just change the XML and XSLT file names as appropriate.
Initiate XSLT from an HTML Page

Example

When the following HTML page is loaded, the init() function is invoked. This function loads the hello.xml and hello.xsl files into XML DOM objects: srcTree and xsltTree, respectively. It then applies the XSLT transformation by calling the transformNode method on the srcTree XML DOM object.

You can use this HTML page to test many of the other XSLT examples in this SDK, as well. You might want to use this approach if the output of the transformation is HTML. Otherwise, use the other approach, Initiate XSLT from a Command Prompt.

For hello.xml and hello.xsl, see Hello, World! (XSLT). The following is the code for the HTML page, hello.htm.

hello.htm

<HTML>
<HEAD>
<TITLE>sample</TITLE>
<SCRIPT language = "javascript">
    function init()
    {
        var srcTree = new ActiveXObject("Msxml2.DOMDocument.5.0");
        srcTree.async=false;
        // You can substitute other XML file names here.
        srcTree.load("hello.xml");

        var xsltTree = new ActiveXObject("Msxml2.DOMDocument.5.0");
        xsltTree.async = false;
        // You can substitute other XSLT file names here.
        xsltTree.load("hello.xsl");

        resTree.innerHTML = srcTree.transformNode(xsltTree);
    }
</SCRIPT>
</HEAD>

<BODY onload = "init()" >
    <div id="resTree"></div>
</BODY>
Note  MSXML 5.0 is used here, with the help of the version dependent ProgID of the DOM, Msxm12.DOMDocument.5.0. When you use this method to start the transformation, any XSLT style sheet that is already embedded in the source document is ignored. For more information about GUIDs and ProgIDs, see GUID and ProgID Information.

Try It!

1. Copy the code above, and paste it into a text file. Save the file as hello.htm.
2. Copy the XML source file, hello.xml, from the Hello, World! topic in the XSLT Starter Kit. Paste it into a text file, and save the file as hello.xml, in the same directory where you saved hello.htm.
3. Copy the XSLT style sheet, hello.xsl, from Hello, World!. Paste it into a text file, and save the file as hello.xsl, in the same directory where you saved hello.htm and hello.xml.
Initiate XSLT from a Command Prompt

You can use the following JScript file to test XSLT samples quickly from a command prompt. This approach will work for transformations with any type of output.

xsltTest.js

var oArgs = WScript.Arguments;

if (oArgs.length == 0)
{
    WScript.Echo ("Usage : cscript xslt.js xml xsl");
    WScript.Quit();
}

xmlFile = oArgs(0) + ".xml";

xslFile = oArgs(1) + ".xsl";

var xsl = new ActiveXObject("MSXML2.DOMDOCUMENT.5.0");
var xml = new ActiveXObject("MSXML2.DOMDocument.5.0");

xml.validateOnParse = false;
xml.async = false;
xml.load(xmlFile);

if (xml.parseError.errorCode != 0)
    WScript.Echo ("XML Parse Error : " + xml.parseError.reason);

xsl.async = false;
xsl.load(xslFile);

if (xsl.parseError.errorCode != 0)
    WScript.Echo ("XSL Parse Error : " + xsl.parseError.reason);

//WScript.Echo (xml.transformNode(xsl.documentElement));

try
{
    WScript.Echo (xml.transformNode(xsl.documentElement));
}
catch(err)
{
}

To test samples with xsltTest.js
1. Copy the JScript code above, and paste it into a text file. Save the file as `xsltTest.js` in your system directory, such as `c:\Windows`. This allows you to invoke the script from any directory. If you save `xsltTest.js` to any other directory, you can either invoke it from that directory or invoke it using its absolute path.

2. Copy the XML and XSLT files you want to test. Paste them into text files, and save them to a test directory, such as `c:\test`. Name the files as appropriate—for example, `sampleXML.xml` and `sampleXSLT.xsl`.

3. Open a command prompt, and navigate to your test directory. Enter the following command:

   `xsltTest sampleXML sampleXSLT`

   Notice the absence of file extensions on the arguments.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Represent Data in XML

As with any other application, the first thing you must decide when you plan an XML application is how to represent your data. Data design is important because it affects which template rules you will use in the XSLT style sheet, and what kind of performance your XSLT solution can achieve.

In an XSLT solution, data comes in the form of XML documents. You can store data values in an XML element, either as the content of the element or as an attribute value. As an illustration, this example puts company information as the content of some elements, and the sales numbers as attributes of others. All this information is contained in the input XML file, Sales.xml.

To create the Sales.xml file

1. Create a folder on your computer, such as C:\Documents and Settings\jsmith\My Documents\xsltTut.
2. Use your HTML or text editor to create a new file. Copy the following code into that file.

```xml
<?xml version="1.0"?>
<sales>
  <summary>
    <heading>Scootney Publishing</heading>
    <subhead>Regional Sales Report</subhead>
    <description>Sales report for the West Coast, Central and</description>
  </summary>
  <data>
    <region>
      <name>West Coast</name>
      <quarter number="1" books_sold="24000" />
      <quarter number="2" books_sold="38600" />
      <quarter number="3" books_sold="44030" />
      <quarter number="4" books_sold="21000" />
    </region>
    <region>
      <name>Central</name>
      <quarter number="1" books_sold="11000" />
      <quarter number="2" books_sold="16080" />
      <quarter number="3" books_sold="25000" />
      <quarter number="4" books_sold="29000" />
    </region>
  </data>
</sales>
```
3. In the folder that you created in Step 1, save the file as Sales.xml. You can leave the file open.
4. In Windows Explorer, locate and then double-click the Sales.xml file. This allows you to view the file in Internet Explorer.

   Internet Explorer shows the file as an unformatted XML listing. This is the default display for Internet Explorer when viewing an XML file.

Basic XSLT processing involves the following:

1. Selecting a specified piece or pieces of data.
2. Inspecting the values of this data.
3. Doing something with the data. For example, the application might display the company name and calculate the annual total sales of different regions.
4. The remaining exercises in this section show you how to do all this, one step at time. The next exercise shows how to Reference Data Values—that is, how to select specific data values out of the input XML document.
Reference Data Values

Suppose that you want to display your company information and sales data in a format that is more readable than the default tree view in Internet Explorer. You can do this by creating the following XSLT file, Transform.xsl. This file transforms the heading data values in the Sales.xml file into a well structured HTML format. The basic operations involve retrieving the data values of the specified elements or attributes. In XSLT, you use XPath pattern matching to do this. This is illustrated in Transform.xsl by use of the match and select attributes of the xsl:template and xsl:value-of elements, respectively. Notice how XPath expressions are constructed, in bold below, to reference element content and an attribute value.

To create the Transform.xsl file

1. Use your HTML or text editor to create a new file. Copy the following code into that file.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
                <!-- Referencing a data value as element content: -->
                <title><xsl:value-of select="/summary/heading"/></title>
            </head>
            <body>
                <h1><xsl:value-of select="/summary/heading"/></h1>
                <h2><xsl:value-of select="/summary/subhead"/></h2>
                <p><xsl:value-of select="/summary/description"/></p>
                </DIV>
                Region: <xsl:value-of select="/data/region[1]/name"/>
            </DIV>
            <!-- Referencing a data value as an attribute value: -->
            <DIV>
                First quarter sales: <xsl:value-of select="/data/region[1]/quarter/@books_sold[1]"/>
            </DIV>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```
2. In the same folder in which you saved the Sales.xml file, save the current file as Transform.xsl.

To specify the XSLT file to use with the XML file

1. In your HTML or text editor, open the Sales.xml file (if not already opened).
2. Locate the following code:
   
   ```xml
   <?xml version="1.0"?>
   ```

3. After this line, add the following code:
   
   ```xml
   <?xml-stylesheet type="text/xsl" href="transform.xsl"?>
   ```

   This processing instruction identifies Transform.xsl as the XSLT file to be used for transforming Sales.xml.

4. Save and close Sales.xml.

To view the transformation results

1. In Internet Explorer, press F5 to refresh Sales.xml. This shows how Transform.xsl formats the data in the <summary> element, as well as the first quarter results of the West Region.
2. If you installed the Internet Explorer Tools for Validating XML and Viewing XSLT Output, you can also right-click in the Internet Explorer window and select View XSL Output from the pop-up menu.
How the Transform Works

At the beginning of Transform.xsl, the xsl:stylesheet line declares that this is an XSLT file. The xsl:output line tells the XSLT processor to produce HTML output instead of the default output method of "xml".

The xsl:template and xsl:value-of lines use the XPath expressions to pick out the content of an element and an attribute value. For example, <xsl:template match="/"/> defines a template rule for the root element of the source XML document and all of its children. <xsl:value-of select="/summary/heading"/> returns the content of the <heading> element contained in the <summary> element. The <xsl:value-of select="/data/region[1]/quarter/@books_sold[1]"/> returns the value of the books_sold attribute for the first quarter in the first region, as ordered in the source document. The HTML tags format the retrieved data values.

Now you know how to select a given element, its content, or its attribute value. The next exercise shows how to Use CSS in XSLT.
Use CSS in XSLT
The XSLT language allows you to manipulate and transform data. An XSLT
style sheet handles data presentation with HTML tags, as shown above, when the
output is in HTML format. You can also use Cascading Style Sheets (CSS) to
format data, just as you would with an ordinary HTML page. The following
example uses CSS within an XSLT style sheet.
To add a CSS style sheet to the Transform.xsl file
1. In the XSLT style sheet you created in the previous step, add a <style>
element with CSS style declarations within the <head> element. The
<style> element should appear after the <title> element, as follows:
<head>
<!-- Referencing a data value as element content: -->
<title><xsl:value-of select="//summary/heading"/></title>
<style>
H1 {background-color:black; color:gold;}
.desc {color:blue; font-style:italic}
</style>
</head>

2. Replace
<p><xsl:value-of select="//summary/description"/></p>

with the following statement:
<p class="desc"><xsl:value-of select="//summary/description"/></p>

3. Save the XSLT file and refresh the Sales.xml file in Internet Explorer.
The next exercise shows how to Perform Iterations — that is, how to work with
a set of elements.


Perform Iterations

The sales data in Sales.xml consists of three regions: West Coast, Central, and East Coast. There are four quarters for each region.

...<data>
<region>
   <name>West Coast</name>
   <quarter number="1" books_sold="24000" />
   <quarter number="2" books_sold="38600" />
   <quarter number="3" books_sold="44030" />
   <quarter number="4" books_sold="21000" />
</region>
<region>
   <name>Central</name>
   <quarter number="1" books_sold="11000" />
   <quarter number="2" books_sold="16080" />
   <quarter number="3" books_sold="25000" />
   <quarter number="4" books_sold="29000" />
</region>
<region>
   <name>East Coast</name>
   <quarter number="1" books_sold="27000" />
   <quarter number="2" books_sold="31400" />
   <quarter number="3" books_sold="40100" />
   <quarter number="4" books_sold="30000" />
</region>
</data>...

There are two repeating elements in this file: <region>, and within each <region>, <quarter>. This suggests an array-like or list-like data structure. As in other programming languages, such as C/C++, Visual Basic, and JScript, you can handle this kind of repeating data set in XSLT with a loop, or recursion. This exercise demonstrates how to process an array-like or list-like data set in loops with the help of the xsl:for-each instruction.

To format the <data> section into table cells

1. Use your HTML or text editor to open Transform.xsl and locate the </body> line.
2. Immediately above the \</body\> line, replace the two \<DIV\>\ldots\</DIV\> statements with the following code:

\begin{verbatim}
<table>
<!-- Create a table heading row, filling in the quarter numbers from left to right. -->
<tr>
   <th>Region\Quarter</th>
   <xsl:for-each select="//\data/region[1]/quarter">
      <th>Q\text{\number}
   </xsl:for-each>
   <th>Total</th>
</tr>
<!-- Create a table body row for each quarter of each region, filling cells with the number of books sold. -->
<xsl:for-each select="//\data/region">
<tr>
   <th style="text-align:left"><\text{\value}{\select{name}}/</th>
   <xsl:for-each select="\quarter">
      <td style="text-align:right">
         <\text{\value}{\select{format-number(@books\_sold,'###,###')}}</td>
   </xsl:for-each>
   <td style="text-align:right;font-weight:bold;">
      <!-- Total will go here. -->
   </td>
</tr>
</xsl:for-each>
</table>
\end{verbatim}

3. Save \textbf{Transform.xsl}. You can leave this file open.

\textbf{To view the transformation of the \textit{<data>} element}

1. In Internet Explorer, press F5 to refresh \textbf{Sales.xml}.

The new portion of the sales report should look like this:

\begin{center}
\begin{tabular}{lcccccc}
\textbf{Region} & \textbf{Quarter} & Q1 & Q2 & Q3 & Q4 & Total \\
\textbf{West Coast} & & 24,000 & 38,600 & 44,030 & 21,000 \\
\textbf{Central} & & 11,000 & 16,080 & 25,000 & 29,000 \\
\textbf{East Coast} & & 27,000 & 31,400 & 40,100 & 30,000 \\
\end{tabular}
\end{center}
2. If you installed the Internet Explorer Tools for Validating XML and Viewing XSLT Output, you can also right-click in the Internet Explorer window and then click **View XSL Output** menu item.
How the Transform Works

This topic focuses on those parts of the transformation that are related to the xsl:for-each statements.

The first part of these new statements, xsl:for-each, tells the parser to do something every time it finds the pattern defined by the second part of the statement. For example, the statement

```xml
<xsl:for-each select="/data/region[1]/quarter">
```

indicates that something should be done every time the parser finds the `<quarter>` element within the first `<region>` element.

The formatting statements that follow each xsl:for-each statement define exactly what the parser has to do when it encounters the specified pattern. For example, the formatting statement that follows the

```xml
<xsl:for-each select="/data/region[1]/quarter">
```

statement is

```xml
<th>Q<xsl:value-of select="@number"/>
```

This formatting statement indicates that the parser needs to look inside the `<quarter>` element, get the value of the `number` attribute, and insert that value into the table header after the letter "Q".

So far, we dealt with data display in an XSLT transformation. However, XSLT is not limited to display, or style. You can use XSLT to implement logic, as well as to enact programming flow-control. The next exercise shows how to Operate on Data Values—that is, how to perform arithmetic operations in XSLT.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Operate on Data Values

Suppose a user wants to compare the annual sales figures for each region. Although this information is not hard coded in the source XML document, Sales.xml, you can derive it from the data that is already in Sales.xml. To do this, use the XPath function, \texttt{sum()}, to add the quarterly results for each region.

To calculate total annual sales per region

1. In your HTML or text editor, open \texttt{Transform.xsl} and delete the following line:
   
   \texttt{<!-- Total will go here. -->}

2. Replace the deleted line with the following code:
   
   \texttt{<xsl:value-of select="format-number(sum(quarter/@books_sold),',##")"/>}

3. Save and close \texttt{Transform.xsl}.

To view the summarized data

- In Internet Explorer, press F5 to refresh the display of Sales.xml. The Total column now shows the total number of books sold for the year per each region:

<table>
<thead>
<tr>
<th>Region\Quarter</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>West Coast</td>
<td>24,000</td>
<td>38,600</td>
<td>44,030</td>
<td>21,000</td>
<td>127,630</td>
</tr>
<tr>
<td>Central</td>
<td>11,000</td>
<td>16,080</td>
<td>25,000</td>
<td>29,000</td>
<td>81,080</td>
</tr>
<tr>
<td>East Coast</td>
<td>27,000</td>
<td>31,400</td>
<td>40,100</td>
<td>30,000</td>
<td>128,500</td>
</tr>
</tbody>
</table>
How the Function Works

An XPath function takes XPath expressions as its arguments, does something to the input data, and returns the result as a new XPath expression. For more information about XPath functions, see XPath Reference.

The sum(quarter/@books_sold) XPath function takes a node-set, quarter/@books_sold, as input. It then returns the sum of the data values for the books_sold attribute of each <quarter> element in the current context. Here, the current context is the current <region> element, as defined by the XPath expression, //data/region, at the beginning of the outer loop.

As with other programming languages, XSLT supports flow control during the transformation. The next exercise shows how to Control the Flow of Transformation by using conditional statements to compare whether or not each sales region met its sales goal figure.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
Control the Flow of Transformation

Flow control allows your program, or transformation, to handle different tasks based on different conditions. To implement flow control in XSLT, you use xsl:if or xsl:choose. The analogous statements in Visual Basic are If ... Then ... End If and IF ... Then ... Else ... End If. The analogous statements in C/C++ and JavaScript are if (?) {...} and if (?) { ... } else { ... }.

In this exercise, you will define a quarterly sales goal. You will then add an <xsl:choose> conditional statement to determine whether a region meets this goal for a specified quarter. When a region meets its goal for a quarter, the sales figure will be displayed in green. Otherwise, the sales figure will be displayed in red. To accomplish this, you will add a style attribute to the <td> HTML tag. This attribute will be set to green if the sales goal is met, and set to red if it is not.

To add conditional statements

1. In your HTML or text editor, open Transform.xsl and locate the following line:
   ```xml
   <xsl:output method="html"/>
   ```

2. Immediately below this line, add the following code:
   ```xml
   <xsl:param name="low_sales" select="21000"/>
   ```

   This creates a parameter variable named low_sales, which represents the sales goal, and assigns it the value of 21000. To reference this parameter, use $low_sales in any ensuing XPath expressions. Now, you can insert the parameter variable low_sales whenever you want to refer to the sales goal. Setting a variable enables you to change the value of the variable in one place and update its value consistently throughout the file.

3. Find and select the following code:
   ```xml
   <td style="text-align:right">
       <xsl:value-of select="format-number(@books_sold,'###,###')"/>
   </td>
   ```
4. Replace it with the following code:

```xml
<td>
  <xsl:attribute name="style">
    <xsl:choose>
      <xsl:when test="number(@books_sold &lt;= $low_sales)">
        color:red;
      </xsl:when>
      <xsl:otherwise>
        color:green;
      </xsl:otherwise>
    </xsl:choose>
    text-align:right;
  </xsl:attribute>
  <xsl:value-of select="format-number(@books_sold,'###,###')"/>
</td>
```

5. Save `Transform.xsl`. You can leave the file open.

To view the conditionalized data

- In Internet Explorer, press F5 to refresh the display of `Sales.xml`. The page now looks like this:

**Scootney Publishing**

**Regional Sales Report**

Sales report for the West Coast, Central and East Coast regions.

<table>
<thead>
<tr>
<th>Region\Quarter</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>West Coast</td>
<td>24,000</td>
<td>38,600</td>
<td>44,030</td>
<td>21,000</td>
<td>127,630</td>
</tr>
<tr>
<td>Central</td>
<td>11,000</td>
<td>16,080</td>
<td>25,000</td>
<td>29,000</td>
<td>81,080</td>
</tr>
<tr>
<td>East Coast</td>
<td>27,000</td>
<td>31,400</td>
<td>40,100</td>
<td>30,000</td>
<td>128,500</td>
</tr>
</tbody>
</table>

**Note** To change how the page appears, you can change the value assigned to the `low_sales` variable. For example, change the value from 21000 to 31000 and then save `Transform.xls`. Return to Internet Explorer and click **Refresh** to see how the page changes.
How the Conditional Statement Works

The code that you just added to Transform.xsl uses a combination of the <xsl:choose>, <xsl:when>, and <xsl:otherwise> elements to form a set of conditional statements.

```xml
<xsl:choose>
  <xsl:when test="number(@books_sold &lt;= $low_sales)">
    color:red;
  </xsl:when>
  <xsl:otherwise>
    color:green;
  </xsl:otherwise>
</xsl:choose>
```

The <xsl:choose> element declares that the enclosed child elements are conditional statements. The first child element, <xsl:when test="number(@books_sold &lt;= $low_sales)">color:red;</xsl:when>, states that if the value of the books_sold attribute in the <quarter> element is smaller than or equal to the value of the low_sales variable, color:red will be assigned to the style attribute of the <td> HTML element. This causes the number to be output in red. If the books_sold value is greater than the low_sales value, the next statement, <xsl:otherwise>color:green;</xsl:otherwise>, sets the output color to green instead of red.

The next exercise shows how to Process Multiple XML Documents.
Process Multiple XML Documents

You can incorporate XML from external documents by using the `document()` function. This function takes as an argument either the relative or absolute URL of the document you want to retrieve. The URL is specified as a string or as a node in which the text value of that node resolves to a URL. For example, the following statement loads the `Ancillary.xml` document and holds it in the `ancillary` variable.

```xml
<xsl:variable name="ancillary" select="document('ancillary.xml')"/>
```

The `Ancillary.xml` document is located in the same directory as the original source XML document, and returns the resulting node set in the variable named `ancillary`. Alternatively, you could declare this ancillary document in the `Sales.xml` file:

```xml
...<sales>
  <ancillary_doc href="ancillary.xml"/>
...</sales>
```

In this case, you would reference it in `Transform.xsl` as follows:

```xml
<xsl:variable name="ancillary" select="document(//ancillary_doc/@href)"
```

In the first case, the argument of the `document` function is a string. In the second case, the argument is a node set.

With `document()`, you load the entire contents of the document, and then select elements from the document by using XML Path Language (XPath) expressions.

In this exercise, first you will create the external document, `Ancillary.xml`. This document contains a watermark and a copyright notice for the sales report. Then you will add code to `Transform.xsl` to retrieve the `Ancillary.xml` file and format the data in that file.

To create `Ancillary.xml`
1. In your HTML or text editor, open a new, empty file.
2. Add the following code to the file:

   ```xml
   <?xml version='1.0'?>
   <document>
     <watermark>Scootney Publishing</watermark>
     <copyright>Copyright 2000, Scootney Publishing. All Rights Re</copyright>
   </document>
   ```

3. In the same folder as Sales.xml, save the file as Ancillary.xml. Then close the file.

**To update Transform.xsl to extract data from Ancillary.xml**

1. In your HTML or text editor, open Transform.xsl and locate the `<body>` line.
2. Immediately below the `<body>` line, add the following code:

   ```xsl
   <xsl:variable name="ancillary" select="document('ancillary.xml')"
   ```

   This loads the Ancillary.xml document and returns the resulting node set in the variable named ancillary.

3. Directly below the line that you just added, add the following code:

   ```xml
   <DIV style="position:absolute;font-size:96;font-family:Times New color:#F0F0F0;z-index:-1">
     <xsl:value-of select="$ancillary//watermark"/>
   </DIV>
   ```

   This code reads the `<watermark>` element from Ancillary.xml and formats the string it contains as a watermark. The XPath expression, $ancillary//watermark, resolves to all `<watermark>` elements in the Ancillary.xml file.

   **Note** As an alternative to the variable declaration, you can accomplish the same effect by referencing the external XML document directly, as follows:

   ```xsl
   <xsl:value-of select="document('ancillary.xml')//watermark"/>
   ```

   However, using a variable to hold the external document is more efficient if you need to reference the document more than once, as is
the case in this exercise.

4. Immediately above the </body> tag, add the following code:

```xml
<DIV style="font-size:9">
  <xsl:value-of select="$ancillary//copyright"/>
</DIV>
```

These lines insert and format the copyright notice string from Ancillary.xml.

5. Save Transform.xsl.

To view the source and ancillary files

- In Internet Explorer, press F5 to refresh the display of Sales.xml. The page now looks like this:

**Scootney Publishing**

**Regional Sales Report**

Sales report for the West Coast, Central and East Coast regions.

<table>
<thead>
<tr>
<th>Region</th>
<th>Quarter</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>West</td>
<td>Coast</td>
<td>24,000</td>
<td>38,600</td>
<td>44,030</td>
<td>21,000</td>
<td>127,630</td>
</tr>
<tr>
<td>Central</td>
<td></td>
<td>11,000</td>
<td>16,080</td>
<td>25,000</td>
<td>29,000</td>
<td>81,080</td>
</tr>
<tr>
<td>East</td>
<td>Coast</td>
<td>27,000</td>
<td>31,400</td>
<td>40,100</td>
<td>30,000</td>
<td>128,500</td>
</tr>
</tbody>
</table>

Copyright 2000, Scootney Publishing. All Rights Reserved.
Final Results of the XSLT How Do I? Examples

As a reference, the following topics contain the finished XML, XSL, and HTML files produced in this section.

- XML Source Document (Sales.xml)
- XSLT Transformation File (Transform.xsl)
- Additional XML Document (Ancillary.xml)
- XSLT Output File (in HTML Format)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Developer's Guide
XML Source Document (Sales.xml)

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="transform.xsl"?>
<sales>
  <summary>
    <heading>Scootney Publishing</heading>
    <subhead>Regional Sales Report</subhead>
    <description>Sales report for the West Coast, Central and East</description>
  </summary>
  <data>
    <region name="West Coast">
      <quarter number="1" books_sold="24000" />
      <quarter number="2" books_sold="38600" />
      <quarter number="3" books_sold="44030" />
      <quarter number="4" books_sold="21000" />
    </region>
    <region name="Central">
      <quarter number="1" books_sold="11000" />
      <quarter number="2" books_sold="16080" />
      <quarter number="3" books_sold="25000" />
      <quarter number="4" books_sold="29000" />
    </region>
    <region name="East Coast">
      <quarter number="1" books_sold="27000" />
      <quarter number="2" books_sold="31400" />
      <quarter number="3" books_sold="40100" />
      <quarter number="4" books_sold="30000" />
    </region>
  </data>
</sales>
XSLT Transformation File (Transform.xsl)

When you finish the tutorial, **Transform.xsl** is as follows:

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:param name="low_sales" select="21000"/>

    <xsl:template match="/">
        <html>
            <head>
                <title><xsl:value-of select="//summary/heading"/></title>
            </head>
            <body>
                <xsl:variable name="ancillary" select="document('ancillary.xml')"/>
                <div style="position:absolute;font-size:96;font-family:Times New Roman;color:#F0F0F0;z-index:-1">
                    <xsl:value-of select="$ancillary//watermark"/>
                </div>
                <h1><xsl:value-of select="//summary/heading"/></h1>
                <h2><xsl:value-of select="//summary/subhead"/></h2>
                <p><xsl:value-of select="//summary/description"/></p>
                <table>
                    <!-- create table heading row, filling in the quarter numbers from left to right -->
                    <tr>
                        <th>Region</th>
                        <th>Quarter</th>
                        <xsl:for-each select="//data/region[1]/quarter">
                            <th>Q<xsl:value-of select="@number"/></th>
                        </xsl:for-each>
                        <th>Total</th>
                    </tr>

                    <!-- create a table body row for each quarter, for each region, filling cells with the number of books sold -->
                    <xsl:for-each select="/data/region">
                        <tr>
                            <th style="text-align:left"><xsl:value-of select="name"/></th>
                            <xsl:for-each select="quarter">
                                <td>
                                    <xsl:value-of select=""/>
                                </td>
                            </xsl:for-each>
                        </tr>
                    </xsl:for-each>
                </table>
        </body>
    </xsl:template>
</xsl:stylesheet>
```
<table>
  <tr>
    <td style="text-align:right;font-weight:bold;">
      <xsl:value-of select="format-number(sum(quarter/@books_sold),'###,###')"/>
    </td>
  </tr>
</table>

<div style="font-size:9">
  <xsl:value-of select="$ancillary//copyright"/>
</div>
Additional XML Document (Ancillary.xml)

<?xml version='1.0'?>
<document>
  <watermark>Scootney Publishing</watermark>
  <copyright>Copyright 2000, Scootney Publishing. All Rights Reserv</copyright>
</document>
XSLT Output File (in HTML Format)

The complete file produced as output from the XSLT processor is shown here. To view this file after you finished the tutorial:

1. View Sales.xml in Internet Explorer.
2. If you installed the Internet Explorer Tools for Validating XML and Viewing XSLT Output, right-click in Internet Explorer and then click View XSL Output.

```html
<html>
<head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-16">
    <title>Scootney Publishing</title>
</head>
<body>
    <div style="position:absolute;font-size:96;font-family:Times New Roman;color:#F0F0F0;z-index:-1">Scootney Publishing</div>
    <h1>Scootney Publishing</h1>
    <h2>Regional Sales Report</h2>
    <p>Sales report for the West Coast, Central and East Coast regions.</p>
    <table>
        <tr>
            <th>Region\Quarter</th>
            <th>Q1</th>
            <th>Q2</th>
            <th>Q3</th>
            <th>Q4</th>
            <th>Total</th>
        </tr>
        <tr>
            <th style="text-align:left">West Coast</th>
            <td style="color:green; text-align:right; ">
                24,000</td>
            <td style="color:green; text-align:right; ">
                38,600</td>
            <td style="color:green; text-align:right; ">
                44,030</td>
            <td style="color:red; text-align:right; ">
                21,000</td>
            <td style="text-align:right;font-weight:bold;">127,630</td>
        </tr>
    </table>
</body>
</html>
```
<table>
<thead>
<tr>
<th>Region</th>
<th>2000</th>
<th>2001</th>
<th>2002</th>
<th>2003</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>11,000</td>
<td>16,080</td>
<td>25,000</td>
<td>29,000</td>
<td>81,080</td>
</tr>
<tr>
<td>East Coast</td>
<td>27,000</td>
<td>31,400</td>
<td>40,100</td>
<td>30,000</td>
<td>128,500</td>
</tr>
</tbody>
</table>

Copyright 2000, Scootney Publishing. All Rights Reserved.
XSLT Reference

The XSLT Reference for Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office contains the following sections:

- XSLT Elements
- XSLT Syntax
- XSLT Functions

See Also

transformNode Method | transformNodeToObject Method | selectNodes Method | selectSingleNode Method | Template-Driven Transformations | Supported Features in XSLT Releases

Other Resources

XSL Transformations (XSLT) Version 1.0 (W3C Recommendation 16 November 1999)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
# XSLT Elements

The XSLT implementation of MSXML versions 4.0 and later includes all XSLT vocabularies, as specified in the World Wide Web (W3C) standards. In addition, it includes one element extension, `<msxsl:script>`. The following table provides a brief summary of all the XSLT elements.

## XSLT Elements

<table>
<thead>
<tr>
<th>XSLT Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xsl:apply-imports</code></td>
<td>Invokes an overridden template rule.</td>
</tr>
<tr>
<td><code>xsl:apply-templates</code></td>
<td>Directs the XSLT processor to find the appropriate template to apply, based on the type and context of each selected node.</td>
</tr>
<tr>
<td><code>xsl:attribute</code></td>
<td>Creates an attribute node and attaches it to an output element.</td>
</tr>
<tr>
<td><code>xsl:attribute-set</code></td>
<td>Defines a named set of attributes.</td>
</tr>
<tr>
<td><code>xsl:call-template</code></td>
<td>Invokes a template by name.</td>
</tr>
<tr>
<td><code>xsl:choose</code></td>
<td>Provides multiple conditional testing in conjunction with the <code>&lt;xsl:otherwise&gt;</code> element and <code>&lt;xsl:when&gt;</code> element.</td>
</tr>
<tr>
<td><code>xsl:comment</code></td>
<td>Generates a comment in the output.</td>
</tr>
<tr>
<td><code>xsl:copy</code></td>
<td>Copies the current node from the source to the output.</td>
</tr>
<tr>
<td><code>xsl:copy-of</code></td>
<td>Inserts subtrees and result tree fragments into the result tree.</td>
</tr>
<tr>
<td><code>xsl:decimal-format</code></td>
<td>Declares a decimal-format, which controls the interpretation of a format pattern used by the <code>format-number</code> function.</td>
</tr>
<tr>
<td><code>xsl:element</code></td>
<td>Creates an element with the specified name in the output.</td>
</tr>
<tr>
<td><code>xsl:fallback</code></td>
<td>Calls template content that can provide a reasonable substitute to the behavior of the new element when encountered.</td>
</tr>
<tr>
<td><code>xsl:for-each</code></td>
<td>Applies a template repeatedly, applying it in turn to each node in a set.</td>
</tr>
<tr>
<td><code>xsl:if</code></td>
<td>Allows simple conditional template fragments.</td>
</tr>
<tr>
<td><code>xsl:import</code></td>
<td>Imports another XSLT file.</td>
</tr>
<tr>
<td><code>xsl:include</code></td>
<td>Includes another XSLT file.</td>
</tr>
<tr>
<td><code>xsl:key</code></td>
<td>Declares a named key for use with the <code>key()</code> function in XML Path Language (XPath) expressions.</td>
</tr>
</tbody>
</table>
Sends a text message to either the message buffer or a message dialog box.

Replaces the prefix associated with a given namespace with another prefix.

Inserts a formatted number into the result tree.

Provides multiple conditional testing in conjunction with the `<xsl:choose>` element and `<xsl:when>` element.

Specifies options for use in serializing the result tree.

Declares a named parameter for use within an `<xsl:stylesheet>` element or an `<xsl:template>` element. Allows you to specify a default value.

Preserves white space in a document.

Generates a processing instruction in the output.

Defines global variables and functions for script extensions.

Specifies sort criteria for node lists selected by `<xsl:for-each>` or `<xsl:apply-templates>`.

Strips white space from a document.

Specifies the document element of an XSLT file. The document element contains all other XSLT elements.

Defines a reusable template for generating the desired output for nodes of a particular type and context.

Generates text in the output.

Performs the same function as `<xsl:stylesheet>`.

Inserts the value of the selected node as text.

Specifies a value bound in an expression.

Provides multiple conditional testing in conjunction with the `<xsl:choose>` element and `<xsl:otherwise>` element.

Passes a parameter to a template.

* Denotes a Microsoft proprietary extension to support scripting.

See Also

Other Resources

XHTML™ 1.0: The Extensible HyperText Markup Language
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:apply-imports> Element

Invokes an overridden template rule defined in an imported style sheet.

<xsl:apply-imports />

Element Information

Number of occurrences  Unlimited

Parent elements  
output elements

Child elements  (No child elements)

Remarks

A template rule that is being used to override a template rule in an imported style sheet can use the <xsl:apply-imports> element to invoke the overridden template rule.

At any point in the processing of a style sheet, there is a current template rule. When a template rule is chosen by matching a pattern, it becomes the current template rule until its template has been instantiated. When an <xsl:for-each> element is instantiated, the current template rule becomes null while the <xsl:for-each> element is applied to each element in the source document.

The <xsl:apply-imports> element processes the current node using only template rules that were imported into the style sheet element that contains the current template rule. The node is processed in the current template rule's mode. It is an error if <xsl:apply-imports> is instantiated when the current template rule is null.

This section contains the following examples.
• Example 1 of <xsl:apply-imports>
• Example 2 of <xsl:apply-imports>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example 1 of `<xsl:apply-imports>`

This example demonstrates how to use `<xsl:apply-imports>` to reuse code more effectively. The example uses four main files:

- The XML source file, `ops.xml`. This data file defines three operations: add (+), sub (-) and mul (*).
- The main XSLT style sheet, `ops.xsl`. This file contains the template rules for the operations, including two `<xsl:import>` elements. The imported style sheets perform the arithmetic and string operations on a given data source.
- An imported style sheet, `arith.xsl`. This XSLT file performs arithmetic operations on each `<op>` element.
- Another imported style sheet, `str.xsl`. This XSLT file performs custom string operations. Here add (+) is treated as a string concatenation; for example, 1+2 becomes 12. Similarly, mul (*) is treated as a reverse concatenation; 1*2 becomes 21. Notice that sub (-) is an undefined string operation.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (ops.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="ops.xsl"?>
<ops>
  <desc>Some binary operations</desc>
  <op name="add" symbol="+">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="sub" symbol="-">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="mul" symbol="*">
    <operand>1</operand>
    <operand>2</operand>
  </op>
</ops>
```
Main XSLT File (ops.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:import href="arith.xsl"/>
  <xsl:import href="str.xsl"/>
  <xsl:template match="op">
    <xsl:value-of select="operand[1]"/>
    <xsl:value-of select="@symbol"/>
    <xsl:value-of select="operand[2]"/>
    = <xsl:apply-imports/>
    <br/>
  </xsl:template>
</xsl:stylesheet>
```

Imported XSLT File (arith.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="op[@symbol='+]">
    <xsl:value-of select="sum(operand)"/>
    (from arith.xsl)
  </xsl:template>
  <xsl:template match="op[@symbol='-']">
    <xsl:value-of select="number(operand[1])-number(operand[2])"/>
    (from arith.xsl)
  </xsl:template>
  <xsl:template match="op[@symbol='*']">
    <xsl:value-of select="number(operand[1])*number(operand[2])"/>
    (from arith.xsl)
  </xsl:template>
</xsl:stylesheet>
```
**Imported XSLT File (str.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="desc">
    <DIV><xsl:value-of select="."/></DIV>
  </xsl:template>
  <xsl:template match="op[@name='add']">
    <xsl:value-of select="operand[1]"/>
    <xsl:value-of select="operand[2]"/>
    (from str.xsl)
  </xsl:template>
  <xsl:template match="op[@name='mul']">
    <xsl:value-of select="operand[2]"/>
    <xsl:value-of select="operand[1]"/>
    (from str.xsl)
  </xsl:template>
</xsl:stylesheet>
```

**Output**

You should get the following output:

Some binary operations
1+2 = 12 (from str.xsl)
1-2 = -1 (from arith.xsl)
1*2 = 21 (from str.xsl)

**Remarks**

The last imported style sheet has the highest import precedence. In this example, **str.xsl** is imported last, and therefore has a higher import precedence than **arith.xsl**. Both imported style sheets have templates for `add` and `mul` operations. Only those from **str.xsl** are called. The sub operation defined in **arith.xsl** is used, however, because no sub operation is defined in **str.xsl**.

Suppose we reversed the order of the `<xsl:import>` elements in the main XSLT file:

```xml
<xsl:import href="arith.xsl"/>
<xsl:import href="str.xsl"/>
```

In this case, the output would be the following:

Some binary operations
1+2 = 3 (from arith.xsl)
1-2 = -1 (from arith.xsl)
1*2 = 2 (from arith.xsl)
Also, if the `<xsl:apply-imports/>` instruction is absent from the overriding template rule for the `<op>` in the main XSLT file (`ops.xsl`), the output is the following:

Some binary operations
1+2 =
1-2 =
1*2 =

That is, the template rule in the importing style sheet overrides the related template rules in the imported style sheets. The `<xsl:apply-imports/>` instruction lets you reactivate these overridden template rules in a different manner.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example 2 of `<xsl:apply-imports>`

This example demonstrates how to use `<xsl:apply-imports>` when you want to add to the functionality of a rule in an imported file, but you don't want to replace the rule entirely. This example uses three main files:

- The **books.xml** sample file, slightly altered.
- The **sample.xsl** style sheet sets the font as Arial and imports the other style sheet. When a book element is found, the Arial font is applied to the elements in the node. Then the `<xsl:apply-imports>` tag applies the matching rules in the imported sample-import style sheet.
- The **sample-import.xsl** style sheet is the imported style sheet. It provides line breaks, and the formatting for bold, italic, and blue text.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (books.xml)**

Use the [Sample XML File (books.xml)](#).

**Main XSLT File (sample.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:import href="sample-import.xsl"/>
  <xsl:output method="html"/>

  <xsl:template match="book">
    <font face="Arial">
      <xsl:apply-imports/>
    </font>
  </xsl:template>

</xsl:stylesheet>
```

**Imported XSLT File (sample-import.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```
Try It!

1. Go to the Sample XML File (books.xml).
2. Copy books.xml and paste it into a file.
3. Below the line <?xml version="1.0"?>, add the following line:

   <?xml-stylesheet type="text/xsl" href="sample.xsl"?>

4. Test the file using one of the approaches outlined in Initiate XSLT in a Script.

Output
This is the formatted output:

**XML Developer's Guide** By: Gambardella, Matthew

*Midnight Rain* By: Ralls, Kim

*Maeve Ascendant* By: Corets, Eva

*Oberon's Legacy* By: Corets, Eva

*The Sundered Grail* By: Corets, Eva

*Lover Birds* By: Randall, Cynthia

*Splish Splash* By: Thurman, Paula

*Creepy Crawlies* By: Knorr, Stefani

*Paradox Lost* By: Kress, Peter

*Microsoft .NET: The Programming Bible* By: O'Brien, Tim

*MSXML3: A Comprehensive Guide* By: O'Brien, Tim

*Visual Studio 7: A Comprehensive Guide* By: Galos, Mike

This is the processor output:

<html>
<body>
<font face="Arial"><i><b>XML Developer's Guide</b></i></font> By: <font color="blue"><i><b>Midnight Rain</b></i></font> By: <font color="blue"><i><b>Creepy Crawlies</b></i></font> By: <font color="blue"><i><b>Paradox Lost</b></i></font> By: <font color="blue"><i><b>Microsoft .NET: The Programming Bible</b></i></font> By: <font color="blue"><i><b>MSXML3: A Comprehensive Guide</b></i></font> By: <font color="blue"><i><b>Visual Studio 7: A Comprehensive Guide</b></i></font>
</body>
</html>

See Also

<xsl:import Element>
<xsl:apply-templates> Element

Directs the XSLT processor to find the appropriate template to apply, based on the type and context of each selected node.

```xml
<xsl:apply-templates
    select = "Expression"
    mode = "QName">
</xsl:apply-templates>
```

Attributes

**select**
Can be used to process nodes selected by an expression, instead of processing all children. The value of the select attribute is an expression, which must evaluate to a node-set. The selected node-set is processed in document order, unless a different sort order has been specified.

**mode**
The mode attribute allows an element to be processed multiple times, each time producing a different result. If `<xsl:template>` does not have a match attribute, it cannot have a mode attribute. If an `<xsl:apply-templates>` element has a mode attribute, it applies only to template rules from `<xsl:template>` elements that have a mode attribute with the same value. If an `<xsl:apply-templates>` element does not have a mode attribute, it applies only to template rules from `<xsl:template>` elements that do not have a mode attribute.

Element Information

**Number of occurrences**  Unlimited
`xsl:attribute`, `xsl:comment`, `xsl:copy`, `xsl:element`,
`xsl:fallback`, `xsl:for-each`, `xsl:if`, `xsl:message`,
`xsl:otherwise`, `xsl:param`, `xsl:processing-instruction`,
`xsl:template`, `xsl:variable`, `xsl:when`, `xsl:with-param`,
output elements

**Parent elements**
`xsl:otherwise`, `xsl:param`, `xsl:processing-instruction`,
`xsl:template`, `xsl:variable`, `xsl:when`, `xsl:with-param`,
output elements

**Child elements**
`xsl:sort`, `xsl:with-param`
Remarks

The `<xsl:apply-templates>` element first selects a set of nodes using the expression specified in the `select` attribute. If this attribute is left unspecified, all children of the current node are selected. For each of the selected nodes, `<xsl:apply-templates>` directs the XSLT processor to find an appropriate `<xsl:template>` to apply. Templates are tested for applicability by comparing the node to the XPath expression specified in the template's `match` attribute. If more than one template satisfies the match pattern, the one appearing with the highest priority is chosen. If several templates have the same priority, the last in the style sheet is chosen.

For more information, see [Handling Irregular Data Hierarchies](#).

The following topic provides an example of the `<xsl:apply-templates>` element.

- [Example of `<xsl:apply-templates>`](#)
Example of `<xsl:apply-templates>`

The style sheet in this example formats customer data in XML into an HTML `<TABLE>` element. In the output table, each row represents a customer and the columns represent the customer's name, address, and phone number. The `<xsl:sort>` element sorts the customers by state, with all customers from a single state sorted by name.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (customers.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="applyt.xsl" ?>
<customers>
  <customer>
    <name>John Smith</name>
    <address>123 Oak St.</address>
    <state>WA</state>
    <phone>(206) 123-4567</phone>
  </customer>
  <customer>
    <name>Zack Zwyker</name>
    <address>368 Elm St.</address>
    <state>WA</state>
    <phone>(206) 423-4537</phone>
  </customer>
  <customer>
    <name>Albert Aikens</name>
    <address>368 Elm St.</address>
    <state>WA</state>
    <phone>(206) 423-4537</phone>
  </customer>
  <customer>
    <name>Albert Gandy</name>
    <address>6984 4th St.</address>
    <state>WA</state>
    <phone>(206) 433-4547</phone>
  </customer>
  <customer>
    <name>Peter Furst</name>
    <address>456 Pine Av.</address>
```
```xml
<state>CA</state>
<phone>(209) 765-4321</phone>
</customer>
<customer>
<name>Dan Russell</name>
<address>9876 Main St.</address>
<state>PA</state>
<phone>(323) 321-7654</phone>
</customer>
</customers>

XSLT File (applyt.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="/">
  <HTML>
    <BODY>
      <TABLE border="1" cellspacing="0" cellpadding="2">
        <xsl:apply-templates select="customers/customer">
          <xsl:sort select="state"/>
          <xsl:sort select="name"/>
        </xsl:apply-templates>
      </TABLE>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="customer">
  <TR>
    <xsl:apply-templates select="name"/>
    <xsl:apply-templates select="address"/>
    <xsl:apply-templates select="state"/>
    <xsl:apply-templates select="phone"/>
    <xsl:apply-templates select="phone" mode="accountNumber"/>
  </TR>
</xsl:template>

<xsl:template match="name">
  <TD STYLE="font-size:14pt font-family:serif">
    <xsl:apply-templates />
  </TD>
</xsl:template>

<xsl:template match="address">
  <TD> <xsl:apply-templates /> </TD>
</xsl:template>
```
<xsl:template match="state">
    <TD> <xsl:apply-templates /> </TD>
</xsl:template>

<xsl:template match="phone">
    <TD> <xsl:apply-templates /> </TD>
</xsl:template>

<xsl:template match="phone" mode="accountNumber">
    <TD STYLE="font-style:italic">
        1-<xsl:value-of select="."/>-001
    </TD>
</xsl:template>
</xsl:stylesheet>

Output

This is the formatted output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>State</th>
<th>Phone</th>
<th>Account Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter Furst</td>
<td>456 Pine Av.</td>
<td>CA</td>
<td>(209) 765-4321</td>
<td>1-(209) 765-4321-001</td>
</tr>
<tr>
<td>Dan Russell</td>
<td>9876 Main St.</td>
<td>PA</td>
<td>(323) 321-7654</td>
<td>1-(323) 321-7654-001</td>
</tr>
<tr>
<td>Albert Aikens</td>
<td>368 Elm St.</td>
<td>WA</td>
<td>(206) 423-4537</td>
<td>1-(206) 423-4537-001</td>
</tr>
<tr>
<td>Albert Gandy</td>
<td>6984 4th St.</td>
<td>WA</td>
<td>(206) 433-4547</td>
<td>1-(206) 433-4547-001</td>
</tr>
<tr>
<td>John Smith</td>
<td>123 Oak St.</td>
<td>WA</td>
<td>(206) 123-4567</td>
<td>1-(206) 123-4567-001</td>
</tr>
<tr>
<td>Zack Zwyker</td>
<td>368 Elm St.</td>
<td>WA</td>
<td>(206) 423-4537</td>
<td>1-(206) 423-4537-001</td>
</tr>
</tbody>
</table>

This is the processor output:

<HTML>
<BODY>
<TABLE border="1" cellspacing="0" cellpadding="2">
    <TR>
        <TD STYLE="font-size:14pt font-family:serif">Peter Furst</TD>
        <TD>456 Pine Av.</TD>
        <TD>CA</TD>
        <TD>(209) 765-4321</TD>
        <TD>1-(209) 765-4321-001</TD>
    </TR>
    <TR>
        <TD STYLE="font-size:14pt font-family:serif">Dan Russell</TD>
        <TD>9876 Main St.</TD>
        <TD>PA</TD>
        <TD>(323) 321-7654</TD>
        <TD>1-(323) 321-7654-001</TD>
    </TR>
    <TR>
        <TD STYLE="font-size:14pt font-family:serif">Albert Aikens</TD>
        <TD>368 Elm St.</TD>
        <TD>WA</TD>
        <TD>(206) 423-4537</TD>
        <TD>1-(206) 423-4537-001</TD>
    </TR>
    <TR>
        <TD STYLE="font-size:14pt font-family:serif">Albert Gandy</TD>
        <TD>6984 4th St.</TD>
        <TD>WA</TD>
        <TD>(206) 433-4547</TD>
        <TD>1-(206) 433-4547-001</TD>
    </TR>
    <TR>
        <TD STYLE="font-size:14pt font-family:serif">John Smith</TD>
        <TD>123 Oak St.</TD>
        <TD>WA</TD>
        <TD>(206) 123-4567</TD>
        <TD>1-(206) 123-4567-001</TD>
    </TR>
    <TR>
        <TD STYLE="font-size:14pt font-family:serif">Zack Zwyker</TD>
        <TD>368 Elm St.</TD>
        <TD>WA</TD>
        <TD>(206) 423-4537</TD>
        <TD>1-(206) 423-4537-001</TD>
    </TR>
</TABLE>
</BODY>
</HTML>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:attribute> Element

Creates an attribute node and attaches it to an output element.

```xml
<xsl:attribute
   name = "attribute-name"
   namespace = "uri-reference">
</xsl:attribute>
```

Attributes

name
Required. The name of the attribute to create. If this value is a qualified name (QName), the attribute node is created in the namespace currently bound to the prefix, unless it is overridden by a namespace attribute. The value of the name attribute is interpreted as an attribute value template (expressions in curly braces are evaluated and converted to strings as in the <xsl:value-of> element). This allows the name of the attribute to be calculated or obtained from the source XML.

namespace
The namespace Uniform Resource Identifier (URI) of the created attribute. If the name attribute contains a QName, the prefix specified there will be bound to the namespace specified in the namespace attribute. This might result in the addition of other namespace declarations when serializing. This value is interpreted as an attribute value template.

Element Information

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited</th>
</tr>
</thead>
</table>

Parent elements

- xsl:copy
- xsl:element
- xsl:fallback
- xsl:for-each
- xsl:if
- xsl:message
- xsl:otherwise
- xsl:param
- xsl:template
- xsl:variable
- xsl:when
- xsl:with-param
- xsl:apply-imports
- xsl:apply-templates
- xsl:call-template
- xsl:choose
- xsl:copy
- xsl:copy-of

Child elements

- xsl:fallback
- xsl:for-each
- xsl:if
- xsl:message
- xsl:number
- xsl:text
- xsl:value-of
- xsl:variable
Remarks

The contents of this element specify the value of the attribute.

Attributes can be added or modified during transformation by placing the <xsl:attribute> element within elements that generate output, such as the <xsl:copy> element. Note that <xsl:attribute> can be used directly on output elements and not only in conjunction with <xsl:element>.

All attributes must be applied before children are added to the element.

Examples

This short example generates an attribute that obtains its value from the XML source. It generates output in the form <IMG src="value-from-XML-source"/>, using an XPath expression that retrieves the appropriate data from the XML source—here, "imagenames/imagename".

XSLT file

```xml
<IMG>
    <xsl:attribute name="src">
        <xsl:value-of select="imagenames/imagename" />
    </xsl:attribute>
</IMG>
```

Output

```
<IMG src=" imagenames/imagename"/>
```

While the <xsl:attribute> element can be extremely useful for dynamically creating output attributes that are not known prior to transforming a document, you do not need to use this element if you already know the attributes. In the preceding example, you might already know that an IMG element must contain a src attribute. Because you know this requirement prior to transforming the document, you would not have to use the <xsl:attribute> element. You could simplify the transformation syntax and still achieve the same result by using the following:
By using the `<xsl:attribute>` element instead of attribute value templates, you can:

- Calculate the name of the attribute.
- Use conditionals, templates, and attributes sets in conjunction with attribute generation.
- Add attributes to an element generated by the `<xsl:copy>` or `<xsl:element>` element.

This section also contains the following full examples.

- Example 1 of `<xsl:attribute>`
- Example 2 of `<xsl:attribute>`
- Example 3 of `<xsl:attribute>`

See Also

Selecting and Outputting Attributes | Generating Comments, Processing Instructions, and Elements | Qualified Names
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example 1 of `<xsl:attribute>`

This example copies the `<myElement>` element and adds a copied attribute with the value "true".

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (myelem.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="attrcopied.xsl" ?>
<root>
  <myElement>My Data</myElement>
  <myElement>My Other Data</myElement>
</root>
```

**XSLT File (attrcopied.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  
  <xsl:template match="myElement">
    <xsl:copy>
      <xsl:attribute name="copied">true</xsl:attribute>
      <xsl:apply-templates /
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

**Output**

This is the formatted output:

My DataMy Other Data

The following is the processor output, with line breaks added for clarity.

```xml
<?xml version="1.0"?>
<myElement copied="true">My Data</myElement>
```
<myElement copied="true">My Other Data</myElement>
Example 2 of `<xsl:attribute>`

This example demonstrates how to use attribute value templates to use content in the XML source document as the name of an attribute.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (attribute.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="attribute.xsl"?>
<investment>
    <type>stock</type>
    <name>Microsoft</name>
    <price type="high">100</price>
    <price type="low">94</price>
</investment>
```

**XSLT File (attribute.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="investment">
        <xsl:element name="{type}"
            <xsl:attribute name="name" >
                <xsl:value-of select="name"/>
            </xsl:attribute>
            <xsl:for-each select="price">
                <xsl:attribute name="{type}" >
                    <xsl:value-of select="."/>
                </xsl:attribute>
            </xsl:for-each>
        </xsl:element>
    </xsl:template>

1. </xsl:stylesheet>

**Output**
There is no formatted output. This is the processor output:

```xml
<?xml version="1.0"?>
<stock name="Microsoft" high="100" low="94">
</stock>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example 3 of <xsl:attribute>

This example shows the use of <xsl:attribute name="{name()}">.<

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (attribute.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="attribute.xsl"?>
<root>
  <bar>bar1</bar>
  <bar>bar2</bar>
  <bar>bar3</bar>
  <bar>bar4</bar>
  <bar>bar5</bar>
</root>
```

**XSLT File (attribute.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/bar">
    <xsl:element name="myElement">
      <xsl:attribute name="{name()}">
        <xsl:value-of select="."/>
      </xsl:attribute>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

**Output**

There is no formatted output. This is the processor output:

```xml
<?xml version="1.0"?>
<myElement bar="bar1" />
<myElement bar="bar2" />
```
<myElement bar="bar5" />
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:attribute-set> Element

Defines a named set of attributes.

```
<xsl:attribute-set
   name = QName
   use-attribute-sets = QNames >
</xsl:attribute-set>
```

Attributes

**name**
Required. The name of the attribute set. The value of the `name` attribute is a qualified name.

**use-attribute-sets**
A white space-separated list of attribute sets, specified as a qualified name. Specifying this attribute declares each attribute in each listed attribute set.

Element Information

**Number of occurrences** Unlimited

**Parent elements** xsl:stylesheet, xsl:transform

**Child elements** xsl:attribute

Remarks

The content of the `<xsl:attribute-set>` element consists of zero or more `<xsl:attribute>` elements that specify the attributes in the set. To use attribute sets, specify a `use-attribute-sets` attribute on `<xsl:element>`, `<xsl:copy>`, or `<xsl:attribute-set>` elements.

An error occurs if using `use-attribute-sets` attributes on `<xsl:attribute-set>` elements causes an attribute set to directly or indirectly use itself.

Specifying a `use-attribute-sets` attribute can be an efficient way to declare multiple attributes on an element. To accomplish the same results using
<xsl:attribute>, you would have to use an <xsl:attribute> element for each attribute in each named attribute set. You would add these <xsl:attribute> elements as content of the <xsl:element>, <xsl:copy>, or <xsl:attribute-set> element.

Attribute sets can also be used by specifying an xsl:use-attribute-sets attribute on a literal result element. For a literal result element, attributes will be added in the following order:

1. Attributes from attribute sets named in an xsl:use-attribute-sets attribute, in the order listed in that attribute.
2. Attributes specified on the literal result element.
3. Any attributes specified by <xsl:attribute> elements will be added.

Because adding an attribute to an element replaces any existing attribute of that element with the same name, attributes specified in attribute sets can be overridden by attributes specified on the literal result element itself.

The template within each <xsl:attribute> element in an <xsl:attribute-set> element is instantiated each time the attribute set is used. It is instantiated using the same current node and current node list that is used for instantiating the element that bears the use-attribute-sets or xsl:use-attribute-sets attribute. However, it is the position of the <xsl:attribute> element in the style sheet—rather than the position of the element bearing the use-attribute-sets or xsl:use-attribute-sets attribute—that determines which variable bindings are visible. Therefore, only variables and parameters declared by top-level <xsl:variable> and <xsl:param> elements are visible.

Multiple definitions of an attribute set with the same expanded-name are merged. An attribute from a definition that has higher import precedence takes precedence over an attribute from a definition that has lower import precedence.

An error occurs if there are two attribute sets that have the same expanded-name and equal import precedence, and that both contain the same attribute, unless there is a definition of the attribute set with higher import precedence that also contains the attribute.

The following topic provides an example of the <xsl:attribute-set> element.

- Example of <xsl:attribute-set>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of `<xsl:attribute-set>`

The following example creates a named attribute set called `title-style` and uses it in a template rule.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (book.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="attrset.xsl" ?>
<book>
  <chapter>
    <heading>The First Heading</heading>
  </chapter>
  <chapter>
    <heading>The Next Heading</heading>
  </chapter>
</book>
```

**XSLT File (attrset.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="chapter/heading">
  <fo:block quadding="start" xsl:use-attribute-sets="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>

</xsl:stylesheet>
```

**Output**
This is the formatted output:

The First Heading
The Next Heading

The following is the processor output, with line breaks added for clarity.

<?xml version="1.0"?>
<fo:block font-size="12pt"
          font-weight="bold"
          quadding="start"
          xmlns:fo="http://www.w3.org/1999/XSL/Format">
The First Heading
</fo:block>
<fo:block font-size="12pt"
          font-weight="bold"
          quadding="start"
          xmlns:fo="http://www.w3.org/1999/XSL/Format">
The Next Heading
</fo:block>

See Also

<xsl:attribute> Element | <xsl:element> Element | <xsl:copy> Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:call-template> Element

Invokes a template by name.

```xml
<xsl:call-template
    name = QName>
</xsl:call-template>
```

Attributes

name
   Required. The name of the template to be invoked.

Element Information

Number of occurrences       Unlimited
   xsl:attribute, xsl:comment, xsl:copy, xsl:element,
   xsl:fallback, xsl:for-each, xsl:if, xsl:message,
   xsl:otherwise, xsl:param, xsl:processing-instruction,
   xsl:template, xsl:variable, xsl:when, xsl:with-param,
   output elements

Parent elements            xsl:otherwise, xsl:param, xsl:processing-instruction,
   xsl:template, xsl:variable, xsl:when, xsl:with-param,
   output elements

Child elements             xsl:with-param

Remarks

The `<xsl:call-template>` enables you to invoke a named template—that is, an `<xsl:template>` element—that has an assigned name attribute. If an `<xsl:template>` element has a name attribute, it might, but need not, also have a match attribute. An `<xsl:call-template>` element invokes a template by name; it has a required name attribute that identifies the template to be invoked. Unlike `<xsl:apply-templates>` , `<xsl:call-template>` does not change the current node or the current node-list.

An error occurs if a style sheet contains more than one template with the same name and with the same import precedence.
An `<xsl:call-template>` element can contain any number of `<xsl:with-param>` elements. However, it cannot contain other XSLT elements.

The following topic provides an example of `<xsl:call-template>`.

- [Example of `<xsl:call-template>`]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of `<xsl:call-template>`

The following example uses `<xsl:call-template>` element to transform XSLT in a modular fashion. The example uses three main files:

- The XML source file, `topic.xml`. This file represents a topic in a book publication.
- The main XSLT file, `topic.xsl`. This file controls which information is displayed.
- The called XSLT file, `ui.xsl`. This file determines how the information is rendered.

Note To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (topic.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="topic.xsl"?>
<topic name="My_topic"
      title="My Topic">
  <meta>
    <owner>
      <name>Jane</name>
      <email>jane@topicfactory.com</email>
    </owner>
    <since></since>
  </owner>
  <history>
    <created-by>
      <name>John</name>
      <email>john@topicfactory.com</email>
      <date>Nov 5, 2001</date>
    </created-by>
    <modifiers>
    </modifiers>
  </history>
  <keyword></keyword>
  <refs></refs>
</meta>

<para name="para1" title="First Paragraph">
The first para has both name and title.
```
the second para has a title but no name.

Third para has neither name nor title.
Component XSLT File (ui.xsl)

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template name="topic_title">
        <xsl:param name="editable"/>
        <xsl:param name="value"/>
        <H2>
            <xsl:attribute name="CONTENTEDITABLE">
                <xsl:value-of select="$editable"/>
            </xsl:attribute>
            <xsl:value-of select="$value"/>
        </H2>
    </xsl:template>
    <xsl:template name="para_title">
        <xsl:param name="value"/>
        <xsl:param name="editable"/>
        <DIV STYLE="font-size:16;
            font-family:Arial;
            font-weight:bold;
            font-style:italic"
            CONTENTEDITABLE="{$editable}">
            <xsl:value-of select="$value"/>
        </DIV>
    </xsl:template>
    <xsl:template name="text">
        <xsl:param name="value"/>
        <xsl:param name="editable"/>
        <SPAN CONTENTEDITABLE="{$editable}">
            <xsl:value-of select="$value"/>
        </SPAN>
    </xsl:template>
</xsl:stylesheet>
```

Output

The following is the formatted output:
This is the processor output:

<h2 CONTENTEDITABLE="true">My Topic</h2>

<p CONTENTEDITABLE="true">
<div STYLE="font-size:16; font-family:Arial; font-weight:bold; font-style:italic" CONTENTEDITABLE="true">
First Paragraph
</div>

The first para has both name and title.
</p>

<p CONTENTEDITABLE="true">
<div STYLE="font-size:16; font-family:Arial; font-weight:bold; font-style:italic" CONTENTEDITABLE="true">
Second Paragraph
</div>

the second para has a title but no name.
</p>

Third para has neither name nor title.

<p CONTENTEDITABLE="true">
<div STYLE="font-size:16; font-family:Arial; font-weight:bold; font-style:italic" CONTENTEDITABLE="true">
First Paragraph
</div>

The first para has both name and title.
</p>

<p CONTENTEDITABLE="true">
<div STYLE="font-size:16; font-family:Arial; font-weight:bold; font-style:italic" CONTENTEDITABLE="true">
Second Paragraph
</div>

Second Paragraph
</p>
The second para has a title but no name.

The third para has neither name nor title.

Remarks

The main XSLT file, `topic.xsl`, controls which information is displayed. It hides the content of `<meta>` element and controls the order of displayed items. It also calls template rules defined in the component XSLT file, `ui.xsl`.

The `ui.xsl` file contains only named template rules that can be called from the first XSLT file. Each named template rule acts like a regular function, taking two input parameters `$value` and `$editable`, and producing HTML output. The `$value` parameter passes the text to be displayed; `$editable` is used to determine whether the output text can be edited (when using Internet Explorer). However, unlike a regular function, the order of input parameters in the named template rule does not have to match the order specified in the calling template rule.

Notice that the template rules are independent of the nodes defined in the source XML document. The `ui.xsl` file is therefore an example of how you can write a general purpose UI library that can be invoked from any other XSLT files.

See Also

<xsl:template> Element | <xsl:with-param> Element

Other Resources
XSLT Samples Viewer
<xsl:choose> Element

Tests multiple conditions in conjunction with the <xsl:otherwise> and <xsl:when> elements.

<xsl:choose>
</xsl:choose>

Element Information

Number of occurrences  Unlimited
  xsl:attribute, xsl:comment, xsl:copy, xsl:element,
  xsl:fallback, xsl:for-each, xsl:if, xsl:message,
  xsl:otherwise, xsl:param, xsl:processing-instruction,
  xsl:template, xsl:variable, xsl:when, xsl:with-param,

Parent elements       xsl:otherwise, xsl:param, xsl:processing-instruction,
  xsl:template, xsl:variable, xsl:when, xsl:with-param,
  output elements

Child elements        xsl:otherwise, xsl:when

Remarks

The <xsl:when> children of the <xsl:choose> element are tested, in order from top to bottom, until a test attribute on one of these elements accurately describes conditions in the source data, or until an <xsl:otherwise> element is reached. Once an <xsl:when> or <xsl:otherwise> element is chosen, the <xsl:choose> block is exited. No explicit break or exit statement is required.

For simple conditional testing, use the <xsl:if> element.

Example

The following example shows a template for <order> elements, and inserts a size indicator before the contents of each <order>. This size indicator is based on the value of the <total> element within each <order> element. If the total is less than 10, the text "(small)" is added. If the total is less than 20, the text "(medium)" is added. If the total is 20 or more, the text "(large)" is added.
Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (order.xml)

```xml
<?xml version="1.0"?><orders>
    <order>
        <lineitem/>
        <lineitem/>
        <total>9</total>
    </order>
    <order>
        <lineitem/>
        <lineitem/>
        <total>19</total>
    </order>
    <order>
        <lineitem/>
        <lineitem/>
        <total>29</total>
    </order>
</orders>
```

XSLT File (refchoose.xsl)

```xml
<?xml version="1.0"?><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="order">
        <xsl:choose>
            <xsl:when test="total &lt; 10">
                (small)
            </xsl:when>
            <xsl:when test="total &lt; 20">
                (medium)
            </xsl:when>
            <xsl:otherwise>
                (large)
            </xsl:otherwise>
        </xsl:choose>
        <xsl:apply-templates />
        <BR/>
    </xsl:template>
</xsl:stylesheet>
```
Output

(small) 9

(medium) 19

(large) 29

See Also

[Defining Conditional Templates Using `<xsl:if>` and `<xsl:choose>`](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:comment> Element

Generates a comment in the output.

<xsl:comment>
</xsl:comment>

Element Information

Number of occurrences  Unlimited

Remarks

The text generated by the children of <xsl:comment> appears between the starting characters (<!--) and the closing characters (--}).

Example

In the following example, the news.xsl style sheet transforms the news.xml document, and inserts a comment into the XSLT output.

   Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (news.xml)

<?xml version ="1.0"?>
<?xml-stylesheet type="text/xsl" href="news.xsl"?>
<news>
  <story1>Here is the top news story.</story1>
  <story2>Here is the next news story.</story2>
</news>

**XSLT File (news.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="/">
  <HTML>
    <BODY>
      <xsl:comment>insert top news story</xsl:comment>
      <P>
        <xsl:value-of select="/story1"/>
      </P>
    </BODY>
  </HTML>
</xsl:template>
</xsl:stylesheet>
```

**Output**

This is the formatted output:

Here is the top news story.

This is the processor output:

```xml
<HTML>
  <BODY><!--insert top news story-->
  <P>Here is the top news story.</P>
  </BODY>
</HTML>
```

**See Also**

[Generating Comments, Processing Instructions, and Elements](#)

**Other Resources**

[Internet Explorer Tools for Validating XML and Viewing XSLT Output](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:copy> Element

Copies the current node from the source to the output.

<xsl:copy
   use-attribute-sets = QNames />
</xsl:copy>

Attributes

use-attribute-sets

A white space separated list of attribute sets, specified as a qualified name.
A white space separated list of attribute sets, specified as a qualified name.
Specifying this attribute declares each attribute in each listed attribute set.

Element Information

Number of occurrences  Unlimited

Parent elements


Child elements


Remarks

The <xsl:copy> element creates a node in the output with the same name, namespace, and type as the current node. Attributes and children are not copied automatically. This element makes identity transformation possible.
Example

The following example performs an identity transformation on an entire document. The identity transformation copies each node in the source to the output to provide a logically equivalent tree. It does not yield character-by-character equivalence: Entities will be expanded, and white space not marked as significant might be removed.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (booksshort.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="identityxfm.xsl"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.</description>
  </book>
</catalog>
```
XSLT File (identityxfm.xsl)

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

    <xsl:template match="/ | @* | node()">
        <xsl:copy>
            <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>

Output

This is a portion of the formatted output, truncated on the right side:

Gamardella, Matthewcomputer44.952000-10-01An in-depth look
and her own childhood to become queen of the world.Corets, EvaFa

This is the processor output:

<?xml version="1.0"?><xsl:stylesheet type="text/xsl"
    href="identityxfm.xsl"?><catalog><book id="bk101"><author>Gambardell
Matthew</author><title>XML Developer's Guide</title><genre>Computer</genre><price>44.95</price><publish_date>2000-10-01</publish_date><description>An in-depth look at creating
applications with XML.</description></book><book id="bk102">
...
</book></catalog>

See Also

Generating Comments, Processing Instructions, and Elements
<xsl:copy-of> Element

Inserts subtrees and result tree fragments into the result tree.

<xsl:copy-of
   select = Expression/>

Attributes

select
   Required. XPath expression identifying nodes to be copied. The subtree below each of these nodes is copied to the result tree in its entirety.

Element Information

Number of occurrences     Unlimited
   xsl:attribute, xsl:comment, xsl:copy, xsl:element,
   xsl:fallback, xsl:for-each, xsl:if, xsl:message,
   xsl:otherwise, xsl:param, xsl:processing-instruction,
   xsl:template, xsl:variable, xsl:when, xsl:with-param,
   output elements

Parent elements            (No child elements)
Child elements

Remarks

When the result of evaluating the expression is a result tree fragment, the complete fragment is copied into the result tree. When the result is a node-set, all the nodes in the set are copied in document order into the result tree. Copying an element node copies the attribute nodes, namespace nodes, and children of the element node, as well as the element node itself. A root node is copied by copying its children. When the result is neither a node-set nor a result tree fragment, the result is converted to a string and then inserted into the result tree, as with <xsl:value-of>.

Example
This sample XSLT file operates against the sample XML file to find a `<person>` element with `<given-name>` and `<family-name>` child elements. The paragraph will contain the first `<given-name>` child element of the current node, including any attributes and child elements, followed by a space and the first `<family-name>` child element, including any attributes and child elements, of the current node.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (family.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="family.xsl"?>
<family>
  <person>
    <given-name age="10">
      <name>Fred</name>
      <nick-name>Freddy</nick-name>
    </given-name>
    <family-name>Smith</family-name>
  </person>
  <person>
    <given-name age="10">
      <name>Robert</name>
      <nick-name>Bob</nick-name>
    </given-name>
    <family-name>Smith</family-name>
  </person>
</family>
```

**XSLT File (family.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="person">
  <p>
    <xsl:copy-of select="given-name"/> 
    <xsl:text> </xsl:text>
    <xsl:copy-of select="family-name"/>
  </p>
</xsl:template>

</xsl:stylesheet>
```
Output

This is the formatted output:

Fred Freddy Smith

Robert Bob Smith

The following is the processor output, with line breaks added for clarity.

```xml
<?xml version="1.0"?>
<p><given-name age="10">
  <name>Fred</name>
  <nick-name>Freddy</nick-name>
  </given-name>
  <family-name>Smith</family-name></p>
<p><given-name age="10">
  <name>Robert</name>
  <nick-name>Bob</nick-name>
  </given-name>
  <family-name>Smith</family-name></p>
```

See Also

<xsl:value-of> Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:decimal-format> Element

Declares a decimal-format, which controls the interpretation of a format pattern used by the format-number function. If there is a name attribute, the element declares a named decimal-format; otherwise, it declares the default decimal-format. The value of the name attribute is a qualified name (QName). An error occurs if you declare either the default decimal-format or a decimal-format with a given name more than once (even with different import precedence), unless it is declared every time with the same value for all attributes (taking into account any default values).

For more information, see format-number Function.

<xsl:decimal-format
  name = QName
  decimal-separator = char
  grouping-separator = char
  infinity = string
  minus-sign = char
  NaN = string
  percent = char
  per-mille = char
  zero-digit = char
  digit = char
  pattern-separator = char />

Attributes

name
  Optional. The name of the decimal-format. If there is a name attribute, the element declares a named decimal-format; otherwise, it declares the default decimal-format. The value of the name attribute is a qualified name. It is an error to declare either the default decimal-format or a decimal-format with a given name more than once unless it is declared every time with the same value for all attributes (taking into account any default values).

decimal-separator
  Optional. The character used to separate the integer and the fraction part of a number. The default is ".".

grouping-separator
Optional. The character used to separate groups of digits. The default is ",".

infinity
Optional. The string used to represent infinity; the default value is the string "Infinity".

minus-sign
Optional. The character used as the default minus sign; the default value is the hyphen-minus character ("-", "#x2D").

NaN
Optional. The string used to represent the NaN value; the default value is the string "NaN".

percent
Optional. The character used as a percent sign; the default value is the percent character ("%").

per-mille
Optional. The character used as a per mille (per thousand) sign; the default value is the Unicode per-mille character ("#x2030").

zero-digit
Optional. The character used as the digit zero; the default value is the digit zero ("0").

digit
Optional. The character used in a format pattern to indicate a place where a leading zero digit is required. The default value is "#".

pattern-separator
Optional. The character used to separate positive and negative sub patterns in a pattern. The default value is the semi-colon character (";"). An example of the semi-colon used in a pattern is the following:
\texttt{format-number(-123.45, '###,###.00;(###,###.00)')}

**Element Information**

- **Number of occurrences**: Unlimited
- **Parent elements**: \texttt{xsl:stylesheet}, \texttt{xsl:transform}
- **Child elements**: (No child elements)

The following topic provides an example of \texttt{xsl:decimal-format}.

- **Example of \texttt{xsl:decimal-format}**
Example of `<xsl:decimal-format>`

This example shows how to format numbers using the `<xsl:decimal-format>` element.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (root.xml)**

You can view either this file or the XSLT file in Internet Explorer; the XSLT file calls itself.

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="decimalformat.xsl"?>
<root/>
```

**XSLT File (decimalformat.xsl)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="decimalformat.xsl"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:decimal-format name="us" decimal-separator='.' grouping-separator=','
    <xsl:decimal-format name="european" decimal-separator=',' grouping-separator='.'
    <xsl:decimal-format name="example" decimal-separator="." grouping-separator=",
    infinity="INFINITY" minus-sign="-" NaN="Not a Number" percent="%"
    per-mille="m" zero-digit="0" digit="#" pattern-separator=";"/>
    <xsl:template match="/">
        <HTML><BODY>
            <table border="1" cellspacing="2" cellpadding="0">
                <tr align="center">
                    <td>Data</td>
                    <td>Default</td>
                    <td>European</td>
                    <td>US</td>
                    <td>Example</td>
                </tr>
                <tr align="right">
                    <td>24535.2</td>
                </tr>
        </table>
    </xsl:template>
</xsl:stylesheet>
```
<table>
<thead>
<tr>
<th></th>
<th>US (###,###.00)</th>
<th>European (###,###.00)</th>
<th>European (###.###,00)</th>
<th>Example (###,###.00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24535.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-123.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>193 / 200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This is the formatted output:

<table>
<thead>
<tr>
<th>Data</th>
<th>Default</th>
<th>European</th>
<th>US</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>24535.2</td>
<td>24,535.20</td>
<td>24.535,20</td>
<td>24,535.20</td>
<td>24,535.20</td>
</tr>
<tr>
<td>-123.45</td>
<td>-123.45</td>
<td>-123.45</td>
<td>-123.45</td>
<td>-123.45</td>
</tr>
<tr>
<td>-123.45</td>
<td>(123.45)</td>
<td>(123,45)</td>
<td>(123.45)</td>
<td>(123.45)</td>
</tr>
<tr>
<td>193 ÷ 200</td>
<td>96.5%</td>
<td>96.5%</td>
<td>96.5%</td>
<td>96.5%</td>
</tr>
<tr>
<td>193 ÷ 200</td>
<td>965%</td>
<td>965%</td>
<td>965%</td>
<td>965m</td>
</tr>
<tr>
<td>1 ÷ 0</td>
<td>Infinity</td>
<td>Infinity</td>
<td>Infinity</td>
<td>INFINITY</td>
</tr>
<tr>
<td>a ÷ 0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

The following is a portion of the processor output stream.

<HTML><BODY>
<table border="1" cellspacing="0" cellpadding="2">
<tr align="center">
<td>Data</td>
<td>Default</td>
<td>European</td>
<td>US</td>
<td>Example</td>
</tr>
<tr align="right">
<td>24535.2</td>
<td>24,535.20</td>
<td>24,535,20</td>
<td>24,535.20</td>
<td>24,535.20</td>
</tr>
<tr align="right">
<td>1 ÷ 0</td>
<td>Infinity</td>
<td>Infinity</td>
<td>Infinity</td>
<td>INFINITY</td>
</tr>
</table>
</BODY></HTML>
See Also

format-number Function | Qualified Names
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:element> Element

Creates an output element with the specified name.

```xml
<xsl:element
    name = "element-name"
    namespace = "uri-reference"
    use-attribute-sets = QNames>
</xsl:element>
```

Attributes

name
Required. The name of the element to create. If this value is a qualified name (QName), the element node is created in the namespace currently bound to the prefix, unless overridden by a namespace attribute. The value of the name attribute is interpreted as an attribute value template—that is, expressions in curly braces are evaluated and converted to strings, as in `<xsl:value-of>`. This allows the name of the element to be calculated or obtained from the source XML.

namespace
The namespace URI of the created element. If the name attribute contains a QName, the prefix specified there will be bound to the namespace specified in the namespace attribute. This might require additional namespace declarations when serializing. The namespace value is interpreted as an attribute value template.

use-attribute-sets
A white space separated list of attribute sets, specified as a qualified name. Specifying this attribute declares each attribute in each listed attribute set.

Element Information

Number of occurrences Unlimited

Parent elements
**Remarks**

The `<xsl:element>` element allows an element to be created with a computed name. The name of the element to be created is specified by a required `name` attribute and an optional `namespace` attribute. The content of the `<xsl:element>` element is a template for the attributes and children of the created element.

This element provides an escaping mechanism for creating elements with namespace clashes, such as XSLT itself.

**Example**

The `<xsl:element>` element is necessary because XSLT elements cannot be used as output elements. This example shows how to output an `<xsl:template>` element.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (item.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="element.xsl" ?>
<root>
  <item>My Item</item>
</root>
```

**XSLT File (element.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
<xsl:template match="item">
  <xsl:element name="xsl:template">
```

Output

This is the formatted output:

My Item

The following is the processor output, with line breaks added for clarity.

```
<?xml version="1.0"?>
<xsl:template match="cost" xml:space="preserve"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  My Item</xsl:template>
```

See Also

Generating Comments, Processing Instructions, and Elements | Qualified Names
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:element> Element

The <xsl:element> element is designed to handle XSLT elements that cannot be handled by the parser: for example, elements that are part of a new version or unrecognized extension. The <xsl:element> element works by calling template content that provides a reasonable substitute for the behavior of the new element.

<xsl:element/>
</xsl:element>

Attributes

None

Element Information

Number of occurrences  Unlimited  

Parent elements  

Child elements

Remarks

When an XSLT document is first loaded, the XSLT preparser performs validation on all XSLT elements. If the version passed in the <xsl:stylesheet> element is greater than that supported by the parser, the parser will execute the <xsl:element> children of any unfamiliar element that is encountered. If an
<xs1:fallback> element has no content, the parser does not perform any action. If an element is supported, the templates in its <xs1:fallback> children are never instantiated. If the version given in the style sheet is the same as that supported by the browser, an error will be called.

Fallbacks are part of the forward-processing mechanism that XSLT uses for handling upgrades. By creating alternative ways of handling a command if an element is not supported, forward-processing guarantees that the code used is relatively robust and insensitive to differences in parser conformity.

In addition to handling versioning differences, the fallback mechanism can also be used to handle elements defined by namespace extensions. Because adding functionality to the base specification is typically very difficult, namespace extensions let developers define functionality beyond that specified by the XSLT specification. If the namespace is not supported (no definition for the namespace is provided), or if a namespace function or element is not defined, the fallback mechanism can be used to provide alternate functionality.

To guarantee that the extension is interpreted within the XSLT operation, you need to set the extension-element-prefixes attribute in the <xs1:stylesheet> element so that it contains the name of the prefix to interpret. In addition, declare the namespace for that prefix.

The following topic provides an example of <xs1:fallback>.

- Example of <xs1:fallback>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of `<xsl:fallback>`

The `fallback.xsl` style sheet provides alternate processing for an XML document. The `fallback.xsl` style sheet attempts to process the XML document with a hypothetical `<xsl:import-table>` element. Because this element is not supported in the current version of the parser, the document is alternately processed with the script inside the fallback element.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (records.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="fallback.xsl"?>
<records>
  <record>
    <name>Adam Barr</name>
    <address>222 Cherry</address>
    <phone>555-797-2355</phone>
  </record>
  <record>
    <name>Jeff Adell</name>
    <address>730 Elm</address>
    <phone>555-797-5555</phone>
  </record>
</records>
```

**XSLT File (fallback.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

  <xsl:template match="/">
    <HTML>
      <HEAD><TITLE>Output Table</TITLE></HEAD>
      <BODY>
      <xsl:import-table href="blah.html" name="sample">
        <xsl:fallback>
          <p>
            This version of the parser does not support the creation of a table with the 'xsl:import-table' element, so the follow
          </p>
        </xsl:fallback>
      </xsl:import-table>
    </BODY>
  </xsl:template>

</xsl:stylesheet>
```
table has been generated using the 'fallback' element.
</p>
<table border='2'>
  <xsl:for-each select='records/record'>
    <tr>
      <td><xsl:value-of select='name'/></td>
      <td><xsl:value-of select='address'/></td>
      <td><xsl:value-of select='phone'/></td>
    </tr>
  </xsl:for-each>
</table>
</xsl:fallback>
</xsl:import-table>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

Output

In this scenario, the current version of the parser does not support the creation of a table with the <xsl:import-table> element, so the following table is generated using the <xsl:fallback> element:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam Barr</td>
<td>222 Cherry</td>
<td>555-797-2355</td>
</tr>
<tr>
<td>Jeff Adell</td>
<td>730 Elm</td>
<td>555-797-5555</td>
</tr>
</tbody>
</table>

See Also

system-property Function | element-available Function | function-available Function
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:for-each> Element

Applies a template repeatedly—that is, to each node in a set.

<xsl:for-each
    select = Expression>
</xsl:for-each>

Attributes

select
    Required. Expression evaluated on the current context to determine the set of nodes to iterate over.

Element Information

Number of occurrences     Unlimited
    xsl:attribute, xsl:comment, xsl:copy, xsl:element,
    xsl:fallback, xsl:for-each, xsl:if, xsl:message,
    xsl:otherwise, xsl:param, xsl:processing-instruction,
    xsl:template, xsl:variable, xsl:when, xsl:with-param,
    output elements
    xsl:apply-imports, xsl:apply-templates, xsl:attribute,
    xsl:call-template, xsl:choose, xsl:comment,
    xsl:copy, xsl:copy-of, xsl:element, xsl:fallback,
    xsl:for-each, xsl:if, xsl:message, xsl:number,
    xsl:processing-instruction, xsl:sort, xsl:text,
    xsl:value-of, xsl:variable

Parent elements
    xsl:otherwise, xsl:param, xsl:processing-instruction,
    xsl:template, xsl:variable, xsl:when, xsl:with-param,
    output elements
    xsl:apply-imports, xsl:apply-templates, xsl:attribute,
    xsl:call-template, xsl:choose, xsl:comment,
    xsl:copy, xsl:copy-of, xsl:element, xsl:fallback,
    xsl:for-each, xsl:if, xsl:message, xsl:number,
    xsl:processing-instruction, xsl:sort, xsl:text,
    xsl:value-of, xsl:variable

Child elements

Remarks

The <xsl:for-each> element establishes the context for iteration. The XSLT transformation instructions within this loop are to be applied to the selected nodes. Each source element selected by <xsl:for-each> becomes a new context against which any pattern matching within the <xsl:for-each> occurs.
Example

The XSLT file in this example defines the structure of an output document. The output is a top-level HTML element containing `<BODY>` and `<TABLE>` elements. The table contains repeated rows for each customer. The XSLT file also uses templates to create `<TD>` elements for the name, address, and phone source elements.

Note To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

XML File (customers.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="foreach.xsl" ?>
<customers>
  <customer>
    <name>John Smith</name>
    <address>123 Oak St.</address>
    <state>WA</state>
    <phone>(206) 123-4567</phone>
  </customer>
  <customer>
    <name>Zack Zwyker</name>
    <address>368 Elm St.</address>
    <state>WA</state>
    <phone>(206) 423-4537</phone>
  </customer>
  <customer>
    <name>Albert Aikens</name>
    <address>368 Elm St.</address>
    <state>WA</state>
    <phone>(206) 423-4537</phone>
  </customer>
  <customer>
    <name>Albert Gandy</name>
    <address>6984 4th St.</address>
    <state>WA</state>
    <phone>(206) 433-4547</phone>
  </customer>
  <customer>
    <name>Peter Furst</name>
    <address>456 Pine Av.</address>
    <state>CA</state>
    <phone>(209) 765-4321</phone>
  </customer>
</customers>
```
<customer>
  <name>Dan Russell</name>
  <address>9876 Main St.</address>
  <state>PA</state>
  <phone>(323) 321-7654</phone>
</customer>

XSLT File (foreach.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE>
          <xsl:for-each select="customers/customer">
            <xsl:sort select="state" order="descending"/>
            <xsl:sort select="name"/>
            <TR>
              <TD><xsl:value-of select="name" /></TD>
              <TD><xsl:value-of select="address" /></TD>
              <TD><xsl:value-of select="phone" /></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>

</xsl:stylesheet>
```

Output

This is the formatted output:

- Albert Aikens 368 Elm St.  (206) 423-4537
- Albert Gandy 6984 4th St.  (206) 433-4547
- John Smith 123 Oak St.  (206) 123-4567
- Zack Zwyker 368 Elm St.  (206) 423-4537
- Dan Russell 9876 Main St.  (323) 321-7654
- Peter Furst 456 Pine Av.  (209) 765-4321

This is the processor output:
See Also

Creating and Populating an HTML Template
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:if> Element

Allows simple conditional template fragments.

```xml
<xsl:if
    test = boolean-Expression>
</xsl:if>
```

**Attributes**

**test**

Required. The condition in the source data to test. If the expression in this attribute evaluates to True when cast to a Boolean, the content of `<xsl:if>` is instantiated and placed in the output. Node-sets are cast to a Boolean True if they contain at least one node.

**Element Information**

**Number of occurrences** Unlimited

- `<xsl:fallback`, `<xsl:for-each`, `<xsl:if`, `<xsl:message`
- `<xsl:otherwise`, `<xsl:param`, `<xsl:processing-instruction`

**Parent elements**

- `<xsl:apply-templates`, `<xsl:attribute`, `<xsl:call-template`
- `<xsl:choose`, `<xsl:comment`, `<xsl:copy`, `<xsl:copy-of`
- `<xsl:element`, `<xsl:for-each`, `<xsl:if`, `<xsl:processing-instruction`
- `<xsl:text`, `<xsl:value-of`, `<xsl:variable`

**Child elements**

- `<xsl:template`, `<xsl:variable`, `<xsl:with-param`

**Remarks**

The content is a template. The expression is evaluated, and the resulting object is converted to a Boolean. If the result is True, the content template is instantiated; otherwise, nothing is created.
Examples

- Example 1 of <xsl:if>
- Example 2 of <xsl:if>
- Example 3 of <xsl:if>
- Example 4 of <xsl:if>

See Also

Defining Conditional Templates Using <xsl:if> and <xsl:choose> | XPath Syntax
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example 1 of <xsl:if>

In this example, the names in a group of names are formatted as a comma-separated list.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (names.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="ifcomma.xsl" ?>
<namelist>
  <name>Albert</name>
  <name>Terrance</name>
  <name>Will</name>
  <name>Sylvia</name>
  <name>Timothy</name>
  <name>Gordon</name>
  <name>James</name>
  <name>Robert</name>
  <name>Dan</name>
  <name>Sasha</name>
</namelist>
```

XSLT File (ifcomma.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

  <xsl:template match="namelist/name">
    <xsl:apply-templates/>
    <xsl:if test="position() != last()">,	</xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Output

This is the formatted output:
Albert, Terrance, Will, Sylvia, Timothy, Gordon, James, Robert, Dan, Sasha

This is the processor output:

<?xml version="1.0" encoding="UTF-16"?>Albert, Terrance, Will, Sylvia, Timothy, Gordon, James, Robert, Dan, Sasha

See Also

[Example 2 of <xsl:if>] | [Example 3 of <xsl:if>] | [Example 4 of <xsl:if>]


Example 2 of `<xsl:if>`

Another way to separate the names by commas is to check a name to see if it is the first. In some circumstances this performs better than Example 1 of `<xsl:if>`, because the `last()` function requires that the entire set of names be found and counted, while this approach does not.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (names.xml)**

Use the `names.xml` listing in Example 1 of `<xsl:if>`, but edit the `href` attribute to point to `ifcomma2.xsl`.

**XSLT File (ifcomma2.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="namelist/name">
    <xsl:if test="position() != 1">,
    <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

**Output**

This is the formatted output:

Albert, Terrance, Will, Sylvia, Timothy, Gordon, James, Robert, Dan, Sasha

This is the processor output:

```xml
<?xml version="1.0" encoding="UTF-16"?>Albert, Terrance, Will, Sylvia, Timothy, Gordon, James, Robert, Dan, Sasha
```
Example 3 of <xsl:if>

The following item template rule colors every other table row yellow.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (items.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="ifyellow.xsl" ?>
<items>
  <item>Car</item>
  <item>Pen</item>
  <item>LP Record</item>
  <item>Wisdom</item>
  <item>Cell phone</item>
  <item>Film projector</item>
  <item>Hole</item>
  <item>Canopy</item>
  <item>Widget</item>
  <item>Concept</item>
  <item>Null character</item>
</items>
```

XSLT File (ifyellow.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="/">
<html>
<body>
<table border="1" cellpadding="2" cellspacing="0" width="50%">
<xsl:apply-templates/>
</table>
</body>
</html>
</xsl:template>

<xsl:template match="item">
<tr>
    <xsl:if test="position() mod 2 = 0">
```
<xsl:attribute name="bgcolor">yellow</xsl:attribute>
</xsl:if>
<xsl:apply-templates/>
</tr>
</xsl:template>
</xsl:stylesheet>

Output

This is the formatted output:

<table>
<thead>
<tr>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pen</td>
</tr>
<tr>
<td>LP Record</td>
</tr>
<tr>
<td>Wisdom</td>
</tr>
<tr>
<td>Cell phone</td>
</tr>
<tr>
<td>Film projector</td>
</tr>
<tr>
<td>Hole</td>
</tr>
<tr>
<td>Canopy</td>
</tr>
<tr>
<td>Widget</td>
</tr>
<tr>
<td>Concept</td>
</tr>
<tr>
<td>Null character</td>
</tr>
</tbody>
</table>

This is the processor output:

<html>
<body>
<table border="1" cellpadding="2" cellspacing="0" width="50%">
<tr>Car</tr>
<tr bgcolor="yellow">Pen</tr>
<tr>LP Record</tr>
<tr bgcolor="yellow">Wisdom</tr>
<tr>Cell phone</tr>
<tr bgcolor="yellow">Concept</tr>
<tr><xsl:apply-templates/>&lt;/tr&gt;&lt;/table&gt;&lt;/body&gt;&lt;/html&gt;

See Also

Example 1 of <xsl:if> | Example 2 of <xsl:if> | Example 4 of <xsl:if>
Example 4 of <xsl:if>

This example inserts the text "International Stock " when the <stock> element has an attribute named international.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (stock.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="ifstock.xsl" ?>
<stocks>
    <stock international="yes">Microsoft</stock>
    <stock>Wingtip Toys</stock>
    <stock international="yes">Contoso Pharmaceuticals</stock>
    <stock>Contoso, Ltd</stock>
    <stock international="yes">Fabrikam, Inc.</stock>
</stocks>
```

XSLT File (ifstock.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="/">
    <html><body>
        <xsl:apply-templates/>
    </body></html>
</xsl:template>

<xsl:template match="stock">
    <p/>
    <xsl:if test="@international">International Stock </xsl:if>
    <xsl:apply-templates />
</xsl:template>
</xsl:stylesheet>
```

Output

This is the formatted output:
International Stock Microsoft

Wingtip Toys

International Stock Contoso Pharmaceuticals

Contoso, Ltd

International Stock Fabrikam, Inc.

The following is the processor output, with line breaks added for clarity.

<html><body>
<p>International Stock Microsoft</p>
<p>Wingtip Toys</p>
<p>...</p>
<p>International Stock Fabrikam, Inc.</p>
</body></html>

See Also

Example 1 of <xsl:if> | Example 2 of <xsl:if> | Example 4 of <xsl:if>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:import> Element

Imports another XSLT file.

<xsl:import
href = "uri-reference" />

Attributes

href
   Required. A Uniform Resource Identifier (URI) reference that identifies the XSLT file to be imported.

Element Information

Number of occurrences Unlimited
Parent elements xsl:stylesheet, xsl:transform
Child elements (No child elements)

Remarks

An XSLT file can import another XSLT file using an <xsl:import> element. Importing an XSLT file is the same as including it except that definitions and template rules in the importing file take precedence over those in the imported XSLT file. The <xsl:import> element has an href attribute whose value is a URI reference identifying the style sheet to be imported. A relative URI is resolved relative to the base URI of the <xsl:import> element.

The <xsl:import> element is allowed only as a top-level element. The <xsl:import> element children must precede all other element children of an xsl:stylesheet element, including any xsl:include element children. When xsl:include is used to include a style sheet, any xsl:import elements in the included document are moved up in the included document to follow any existing xsl:import elements in the included document.
The `<xsl:stylesheet>` elements encountered during processing of a style sheet that contains `<xsl:import>` elements are treated as forming an import tree. In the import tree, each `<xsl:stylesheet>` element has one import child for each `<xsl:import>` element that it contains. Any `<xsl:include>` elements are resolved before constructing the import tree. An `<xsl:stylesheet>` element in the import tree is defined to have lower import precedence than another `<xsl:stylesheet>` element in the import tree if it would be visited before that `<xsl:stylesheet>` element in a post-order traversal of the import tree (that is, a traversal of the import tree in which an `<xsl:stylesheet>` element is visited after its import children). Each definition and template rule has import precedence determined by the `<xsl:stylesheet>` element that contains it.

For example, assume the following conditions are true.

- XSLT file A imports XSLT files B and C, in that order.
- XSLT file B imports XSLT file D.
- XSLT file C imports XSLT file E.

The order of import precedence (lowest first) is D, B, E, C, A.

**Note** Because `<xsl:import>` elements are required to occur before any definitions or template rules, an implementation that processes imported XSLT files at the point at which it encounters the `<xsl:import>` element will encounter definitions and template rules in increasing order of import precedence.

In general, a definition or template rule with higher import precedence takes precedence over a definition or template rule with lower import precedence. This is defined in detail for each kind of definition and for template rules.

It is an error if an XSLT file directly or indirectly imports itself. Apart from this, the case where an XSLT file with a particular URI is imported in multiple places is not treated specially. The import tree will have a separate `<xsl:stylesheet>` element for each place that it is imported.

**Note** If `<xsl:apply-imports>` is used, the behavior may be different from the behavior if the XSLT file had been imported only at the place with the highest import precedence.
Example

XSLT File (importer.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

    <xsl:import href="article.xsl"/>
    <xsl:import href="bigfont.xsl"/>

    <xsl:attribute-set name="note-style">
        <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:attribute-set>

</xsl:stylesheet>
```

For complete working examples, see the following topics:

- Finished Example for Importing and Including
- Example of `<xsl:include>`

See Also

- `<xsl:include>` Element | `<xsl:apply-imports>` Element | Importing and Including Style Sheets
<xsl:include> Element

Includes another XSLT file.

```xml
<xsl:include
    href = "uri-reference"/>
```

Attributes

**href**

Required. A Uniform Resource Identifier (URI) reference identifying the XSLT file to be included.

Element Information

**Number of occurrences** Unlimited

**Parent elements** xsl:stylesheet, xsl:transform

**Child elements** (No child elements)

Remarks

An XSLT file can include another XSLT file using the `<xsl:include>` element. The `href` attribute value is a URI reference identifying the file to be included. The relative URI is resolved with relation to the base URI of the `<xsl:include>` element.

The `<xsl:include>` element is only allowed as the child of the `<xsl:stylesheet>` element.

Inclusions are processed in Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office as if they occurred at the tree level. Note that this is not the same as the Document Object Model (DOM) view of the XML tree. The resource located by the `href` attribute value is parsed as an XML document, and the children of the `<xsl:stylesheet>` element in this document replace the `<xsl:include>` element in the including document. Unlike with `<xsl:import>`,
the included template rules have the same import precedence as the ones in the including style sheet. Thus, when the including and the included style sheets both contain a template rule for one element, which template is called depends on the document order. If the template in the including stylesheet follows the included template, the former is invoked. Otherwise, the latter is invoked.

An error occurs if an XSLT file directly or indirectly includes itself.

Including an XSLT file multiple times can cause errors because of duplicate definitions. Such multiple inclusions are less obvious when they are indirect. For example, A, B, C, and D are all XSLT files. Files B, C, and D include other XSLT files, as follows:

- B includes A.
- C includes A.
- D includes both B and C.

Then, the XSLT file, D, will indirectly include the XSLT file, A, twice. You can avoid this error by separating everything in B (other than the inclusion of A) into a separate XSLT file, B¹, and changing B to include B¹ and A. Then, use this same approach on C and, finally, change D to include A, B¹, C¹.

**Example**

The following topic provides an example of `<xsl:include>`.

- Example of `<xsl:include>`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of `<xsl:include>`

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML file (collection.xml)**

```xml
<?xml version="1.0"?>
<COLLECTION>
    <BOOK>
        <TITLE>Lover Birds</TITLE>
        <AUTHOR>Cynthia Randall</AUTHOR>
        <PUBLISHER>Lucerne Publishing</PUBLISHER>
    </BOOK>
    <BOOK>
        <TITLE>The Sundered Grail</TITLE>
        <AUTHOR>Eva Corets</AUTHOR>
        <PUBLISHER>Lucerne Publishing</PUBLISHER>
    </BOOK>
    <BOOK>
        <TITLE>Splish Splash</TITLE>
        <AUTHOR>Paula Thurman</AUTHOR>
        <PUBLISHER>Scootney</PUBLISHER>
    </BOOK>
</COLLECTION>
```

**XSLT File (xslinclude.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" omit-xml-declaration="yes"/>

    <xsl:template match="/">
        <xsl:for-each select="COLLECTION/BOOK">
            <xsl:apply-templates select="TITLE"/>
            <xsl:apply-templates select="AUTHOR"/>
            <xsl:apply-templates select="PUBLISHER"/>
            <BR/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```
<!-- The following template rule will not be called, because the related template in the including stylesheet will be called. If we move this template so that it follows the xsl:include instruction, this one will be called instead.-->
<xsl:template match="TITLE">
  <DIV STYLE="color:blue">
    Title: <xsl:value-of select="."/>
  </DIV>
</xsl:template>

<xsl:include href="xslincludefile.xsl"/>
</xsl:stylesheet>

Included XSLT File (xslincludefile.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xsl:space="preserve">
  <xsl:template match="TITLE">
    Title - <xsl:value-of select="."/>
  </xsl:template>
  
  <xsl:template match="AUTHOR">
    Author - <xsl:value-of select="."/>
  </xsl:template>
  
  <xsl:template match="PUBLISHER">
    Publisher - <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>

Output

This is the formatted output:

Title - Lover Birds
Author - Cynthia Randall
Publisher - Ballantine Books

Title - Catwings
Author - Eva Corets
Publisher - Lucerne Publishing

Title - Home Town
Author - Paula Thurman
Publisher - Scootney

This is the processor output:

Title - Lover Birds
Author - Cynthia Randall
Publisher - Lucerne Publishing

Title - Catwings
Author - Eva Corets
Publisher - Lucerne Publishing

Title - Splish Splash
Author - Paula Thurman
Publisher - Scootney

See Also

<xsl:import> Element
<xsl:element name="key"> Element

The <xsl:element name="key"> element declares a named key—that is, a name-value pair assigned to a specified element in an XML document. The key is used with the key() function in XPath expressions to help you access the assigned elements in a complex XML document efficiently.

```xml
<xsl:element name="key"
    name = QName
    match = Pattern
    use = Expression>
</xsl:element>
```

**Attributes**

- **name**
  Specifies the name of the key.

- **match**
  Specifies the pattern that identifies a node that holds this key.

- **use**
  Specifies what is to be used as the value of the key. The key value can be one of the following: an attribute, a child element, or the content of the matched element.

**Element Information**

- **Number of occurrences**: Unlimited
- **Parent elements**: xsl:stylesheet
- **Child elements**: (No child elements)

**Remarks**

You can use keys as generalized IDs for referencing elements in an XML document. To do this, first use <xsl:element name="key"> to declare a key for a node. Then call the key() function to retrieve the node, supplying the name and value of the key as the function arguments,. For more information, see the example below.
Multiple keys can be declared on a node. This provides multiple ways to query a node.

When you work with relatively large XML documents, a key is often a fast way to query elements. This is because it essentially indexes the locations of the desired nodes ahead of time. However, when `<xs1:key>` elements are compiled, the XSLT processor will not create such indexes unless a `key()` function is called against the keys. This ensures that the time-intensive indexing operation is performed only when it is necessary.

In a stylesheet, `<xs1:key>` elements are top-level elements, and cannot appear within a template. To avoid circular references, you cannot use parameter or variable references as part of an `<xs1:key>` match.

**Example**

The following topic provides an example of the `<xs1:key>` element.

- [Example of `<xs1:key>`]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of <xsl:key>

This example defines a key named "title-search" that locates book elements that have an author attribute equal to "David Perry".

   Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (titles.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="key_sample.xsl" ?>
<titles>
   <book title="XML Today" author="David Perry"/>
   <book title="XML and Microsoft" author="David Perry"/>
   <book title="XML Productivity" author="Jim Kim"/>
</titles>
```

XSLT File (key_sample.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
   <xsl:key name="title-search" match="book" use="@author"/>
   <xsl:template match="/">
      <HTML>
         <BODY>
            <xsl:for-each select="key('title-search', 'David Perry')">
               <div>
                  <xsl:value-of select="@title"/>
               </div>
            </xsl:for-each>
         </BODY>
      </HTML>
   </xsl:template>
</xsl:stylesheet>
```

Output

This is the formatted output:
XML Today

XML and Microsoft

This is the processor output:

```html
<HTML>
<BODY>
<div>XML Today</div>
<div>XML and Microsoft</div>
</BODY>
</HTML>
```

See Also

[key Function]  [id Function]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Reference
The `<xsl:message>` element sends a text message to either the message buffer or a message dialog box, depending on the environment in which the element call is made. It also raises a system-level error message that can be trapped through normal error-handling mechanisms.

```xml
<xsl:message
    terminate = "yes" | "no" >
</xsl:message>
```

### Attributes

**terminate**

Specifies whether the transformation should terminate upon executing this instruction. This attribute can have one of two string values: "yes" or "no". When the `terminate` attribute is set to "yes", the content of the element is displayed as the part of the system-level error message, and the transformation terminates. When it is set to "no", the transformation proceeds, ignoring the error message. The default value is "no".

### Element Information

#### Number of occurrences

Unlimited

#### Parent elements

- `<xsl:attribute>`
- `<xsl:comment>`
- `<xsl:copy>`
- `<xsl:element>`
- `<xsl:fallback>`
- `<xsl:for-each>`
- `<xsl:if>`
- `<xsl:message>`
- `<xsl:otherwise>`
- `<xsl:param>`
- `<xsl:processing-instruction>`
- `<xsl:template>`
- `<xsl:variable>`
- `<xsl:when>`
- `<xsl:with-param>`
- `<output>`

#### Child elements

- `<xsl:apply-templates>`
- `<xsl:attribute>`
- `<xsl:call-template>`
- `<xsl:choose>`
- `<xsl:comment>`
- `<xsl:copy>`
- `<xsl:copy-of>`
- `<xsl:element>`
- `<xsl:for-each>`
- `<xsl:if>`
- `<xsl:processing-instruction>`
- `<xsl:text>`
- `<xsl:value-of>`
- `<xsl:variable>`

### Remarks
The `<xsl:message>` element provides a mechanism to debug XSLT style sheets in progress. Whenever an `<xsl:message>` element is encountered, if the terminate flag is set to "yes", the XSLT processor quits, and sends a system-level error message. Expressions contained within the `<xsl:message>` element evaluate relative to the current context, making `<xsl:message>` a good way to watch individual elements.

**Example**

The following topic provides an example of the `<xsl:message>` element.

- [Example of `<xsl:message>`]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of <xsl:message>

This example demonstrates use of the <xsl:message> element.

The style sheet verifies that the <name> element within a <record> element has been filled in. If a <name> element is empty, a message indicating that the XML is invalid is output.

In the XML file, a name is not supplied for the <name> element in the second record element.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (records.xml)

```xml
<?xml version='1.0'?>
<xml-stylesheet type="text/xsl" href="message.xsl" ?>
<records>
  <record>
    <name>David Perry</name>
    <address>222 Cherry</address>
    <phone>555-797-79797</phone>
  </record>
  <record>
    <name></name>
    <address>312 Elm</address>
    <phone>555-797-79797</phone>
  </record>
</records>
```

XSLT File (message.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

  <xsl:template match="/">
    <xsl:apply-templates select="*"/>
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```
Output

The formatted output is the following error message in Internet Explorer:

    A name field is empty.

This the processor output:

<?xml version="1.0" encoding="UTF-16"?>

See Also

<xsl:comment> Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
The `<xsl:namespace-alias>` element replaces the prefix associated with a given namespace with another prefix.

```
<xsl:namespace-alias
    stylesheet-prefix = NCName
    result-prefix = NCName/>
```

### Attributes

- **stylesheet-prefix**
  - The prefix for the namespace URI used in the XSLT file.
- **result-prefix**
  - The prefix for the namespace URI used in the result tree.

### Element Information

- **Number of occurrences**: Unlimited
- **Parent elements**: `<xsl:stylesheet>`
- **Child elements**: (No child elements)

### Remarks

Sometimes the content that an XSLT file generates is other XSLT. This presents a challenge with namespaces because there is no explicit way to declare two prefixes with the same namespace without the processor treating both prefixes as the same namespace and acting on them. With the `<xsl:namespace-alias>` command, you can assign an interim namespace to an alternate prefix, apply the style sheet, and then map the alternate namespace to the XSLT one.

Although XSL-to-XSL file generation is the primary use for this command, it is not the only one. You can use this technique anywhere you have colliding namespaces (for example, with the xsi: schema data type namespace). The top-level element must be `<xsl:namespace-alias>`.
Example

The following topic example provides an example of `<xsl:namespace-alias>`.

- Example of `<xsl:namespace-alias>`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example of `<xsl:namespace-alias>`

This example demonstrates how to use `<xsl:namespace-alias>` to declare the alt: prefix as an alias for the XSLT namespace in the literal result tree, when the xsl: prefix is bound for the same namespace URI in the stylesheet. The output is another XSLT style sheet.

**XML File (mymin.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="alias.xsl"?>
<myelem/>
```

**XSLT File (alias.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:alt="http://www.w3.org/1999/XSL/Transform-alternate">

<xsl:namespace-alias stylesheet-prefix="alt" result-prefix="xsl"/>
<xsl:param name="browser" select="'InternetExplorer'"/>

<xsl:template match="/>
    <alt:stylesheet>
        <xsl:choose>
            <xsl:when test="$browser='InternetExplorer'">
                <alt:import href="IERoutines.xsl"/>
                <alt:template match="/">
                    <div>
                        <alt:call-template name="showTable"/>
                    </div>
                </alt:template>
            </xsl:when>
            <xsl:otherwise>
                <alt:import href="OtherBrowserRoutines.xsl"/>n
                <alt:template match="/">
                    <div>
                        <alt:call-template name="showTable"/>
                    </div>
                </alt:template>
            </xsl:otherwise>
        </xsl:choose>
    </alt:stylesheet>
</xsl:template>
```

JScript file (test.js)

```javascript
var xmldoc = new ActiveXObject("Msxml2.DOMDocument.5.0");
xmldoc.async=false;
xmldoc.load("mymin.xml");

var xsldoc= new ActiveXObject("Msxml2.DOMDocument.5.0");
xsldoc.async = false;
xsldoc.load("alias.xsl");

var outfile = new ActiveXObject("Msxml2.DOMDocument.5.0");
outfile.async=false;

var strResult;
strResult = xmldoc.transformNode(xsldoc);
outfile.loadXML(strResult);
outfile.save("output.xsl");

WScript.Echo("Output.xsl was made.");
```

Try It!

1. Copy the XML file above and paste it into a text file. Save the file as `mymin.xml` to a directory on your local drive.
2. Copy the XSLT file above and paste it into a text file. Save the file as `alias.xsl` in the same directory.
3. Copy the JScript listing above, and paste it into a text file. Save the file as `test.js`, in the same directory.
4. Double click the `test.js` file from Windows Explorer to launch the application. Alternatively, you can type "test.js" from a command prompt.

   **Note**  Under operating systems other than Windows 2000 or Windows XP, you might need to install Windows Scripting Host (wscript.exe), if it is not already installed.

   The following output file, `output.xsl`, appears on your local drive.

5. Verify that your output is the same as that listed below.

Output
When you run test.js, you should see the following output in a message box or console window:

Output.xsl was made.

The processor also outputs the following stream as an XSLT file, output.xsl, into the folder where you stored the sample files. White space has been added here for clarity.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="IERoutines.xsl" />
<xsl:template match="/">
    <div>
        <xsl:call-template name="showTable" />
    </div>
</xsl:template>
</xsl:stylesheet>
```

See Also

<xsl:import> Element  |  <xsl:include> Element
**<xsl:number> Element**

Inserts a formatted number into the result tree.

```xml
<xsl:number
  level = "single" | "multiple" | "any"
  count = Pattern
  from = Pattern
  value = number-Expression
  format = { string }
  lang = { nmtoken }
  letter-value = { "alphabetic" | "traditional" }
  grouping-separator = { char }
  grouping-size = { number }
/>```

**Attributes**

level
  Specifies what levels of the source tree should be considered; it has the values "single", "multiple" and "any". The default is "single".

count
  A pattern that specifies what nodes should be counted at those levels. If the count attribute is not specified, then it defaults to the pattern that matches any node with the same node type as the current node and, if the current node has an expanded-name, with the same expanded-name as the current node.

from
  A pattern that specifies where counting starts.

value
  Specifies the expression to be converted to a number and output to the result tree. If no value attribute is specified, the `<xsl:number>` element inserts a number based on the position of the current node in the source tree.

format
  A sequence of tokens that specifies the format to be used for each number in the list. If there are no format tokens, then the default value of 1 is used which generates a sequence 1 2 … 10 11 12…. Each number after the first is separated from the preceding number by the separator token preceding
the format token used to format that number. If there are no separator
tokens, a period character is used (".").

<table>
<thead>
<tr>
<th>Format token</th>
<th>Sequence generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 3 4 5 ... 10 11 12 ...</td>
</tr>
<tr>
<td>01</td>
<td>01 02 03 ... 19 10 11 ... 99 100 101...</td>
</tr>
<tr>
<td>A</td>
<td>A B C ... Z AA AB AC...</td>
</tr>
<tr>
<td>i</td>
<td>i ii iii iv v vi vii viii ix x...</td>
</tr>
<tr>
<td>I</td>
<td>I II III IV V VI VII VIII IX X...</td>
</tr>
</tbody>
</table>

lang

Specifies which language's alphabet is used. If no lang value is specified,
the language is determined from the system environment.

letter-value

Disambiguates between numbering sequences that use letters. One
numbering sequence assigns numeric values to letters in alphabetic
sequence, and the other assigns numeric values to each letter in some other
manner traditional in that language. In English, these correspond to the
numbering sequences specified by the format tokens "a" and "i". In some
languages, the first member of each sequence is the same, and so the format
token alone would be ambiguous. A value of "alphabetic" specifies the
alphabetic sequence; a value of "traditional" specifies the other
sequence. Default is "alphabetic".

grouping-separator

Gives the separator used as a grouping (e.g. thousands) separator in decimal
numbering sequences. For example, grouping-separator="," and
grouping-size="3" would produce numbers of the form 1,000,000. If only
one of the grouping-separator and grouping-size attributes is specified,
it is ignored.

grouping-size

Specifies the size (normally 3) of the grouping. For example, grouping-
separator="," and grouping-size="3" would produce numbers of the
form 1,000,000. If only one of the grouping-separator and grouping-
size attributes is specified, it is ignored.

Element Information
Number of occurrences | Unlimited
---|---
Child elements | (No child elements)

Remarks

The number to be inserted is specified by an expression contained in the `value` attribute. The expression is evaluated and the resulting object is converted to a number. The number is rounded to an integer and then converted to a string and inserted in the result tree.

The `<xsl:number>` element first constructs a list of positive integers using the level, count and from attributes:

- **When `level`="single"**, it goes up to the first node in the ancestor-or-self axis that matches the count pattern, and constructs a list of length one containing one plus the number of preceding siblings of that ancestor that match the count pattern. If there is no such ancestor, it constructs an empty list. If the `from` attribute is specified, then the only ancestors that are searched are those that are descendants of the nearest ancestor that matches the `from` pattern. Preceding-siblings has the same meaning here as with the preceding-sibling axis.
- **When `level`="multiple"**, it constructs a list of all ancestors of the current node in document order followed by the element itself; it then selects from the list those nodes that match the count pattern; it then maps each node in the list to one plus the number of preceding siblings of that node that match the count pattern. If the `from` attribute is specified, then the only ancestors that are searched are those that are descendants of the nearest ancestor that matches the `from` pattern. Preceding siblings has the same meaning here as with the preceding-sibling axis.
- **When `level`="any"**, it constructs a list of length one containing the number of nodes that match the count pattern and belong to the set containing the current node and all nodes at any level of the document that are before the current node in document order, excluding any namespace and attribute nodes (in other words the union of the members of the preceding and
ancestor-or-self axes). If the from attribute is specified, then only nodes after the first node before the current node that match the from pattern are considered.

The following languages/numbering schemes are supported. "Format Token" corresponds to the format attribute, "Language" corresponds to the lang attribute, and "Letter-Value" corresponds to the letter-value attribute.

<table>
<thead>
<tr>
<th>Description</th>
<th>Format Token</th>
<th>Language</th>
<th>Letter-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Western</td>
<td>0x0031 (1)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Upper-case letter</td>
<td>0x0041 (A)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Lower-case letter</td>
<td>0x0061 (a)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Upper-case Roman</td>
<td>0x0049 (I)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Lower-case Roman</td>
<td>0x0069 (i)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Upper-case Russian (Cyrillic)</td>
<td>0x0410</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Lower-case Russian (Cyrillic)</td>
<td>0x0430</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Hebrew Alphabetic</td>
<td>0x05d0</td>
<td>n/a</td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Hebrew Traditional</td>
<td>0x05d0</td>
<td>n/a</td>
<td>Traditional</td>
</tr>
<tr>
<td>Arabic</td>
<td>0x0623</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Hindi Consonants</td>
<td>0x0905</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Hindi Vowels</td>
<td>0x0915</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Hindi Numbers</td>
<td>0x0967</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Thai Letters</td>
<td>0x0e01</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Thai Numbers</td>
<td>0x0e51</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Japanese Aiueo (double-byte)</td>
<td>0x30a2</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Japanese Iroha (double-byte)</td>
<td>0x30a4</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Korean Chosung</td>
<td>0x3131</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Taiwanese Decimal</td>
<td>0x4e01</td>
<td>&quot;zh-tw&quot;</td>
<td>n/a</td>
</tr>
<tr>
<td>Korean Decimal</td>
<td>0x4e01</td>
<td>&quot;ko&quot;</td>
<td>n/a</td>
</tr>
<tr>
<td>Asian Decimal</td>
<td>0x4e01</td>
<td>any other lang</td>
<td>n/a</td>
</tr>
<tr>
<td>Asian Kanji</td>
<td>0x58f1</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Language</td>
<td>Format</td>
<td>Letter-value</td>
<td>Other Lang.</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Traditional Taiwanese</td>
<td>0x58f9</td>
<td>&quot;zh-tw&quot;</td>
<td>n/a</td>
</tr>
<tr>
<td>Traditional Chinese</td>
<td>0x58f9</td>
<td>any other lang</td>
<td>n/a</td>
</tr>
<tr>
<td>Chinese &quot;Zodiac&quot; 12</td>
<td>0x5b50</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Chinese &quot;Zodiac&quot; 10</td>
<td>0x7532</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Chinese &quot;Zodiac&quot; 60</td>
<td>0x7532, 0x5b50</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Korean Ganada</td>
<td>0xac00</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Korean Decimal</td>
<td>0xc77c</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Korean 99</td>
<td>0xd558</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Western (double-byte)</td>
<td>0xff11</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Japanese Aiueo (single-byte)</td>
<td>0xff71</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Japanese Iroha (single-byte)</td>
<td>0xff72</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

If the format token alone is enough to disambiguate a particular numbering scheme, there is no need to specify the language or letter-value.

Here are some examples of conversion specifications.

format="&#x30A2;" specifies Katakana numbering.

format="&#x30A4;" specifies Katakana numbering in the "iroha" order.

format="&#x0E51;" specifies numbering with Thai digits.

format="&#x05D0;" letter-value="traditional" specifies traditional Hebrew numbering.

format="&#x10D0;" letter-value="traditional" specifies Georgian numbering.

format="&#x0381;" letter-value="traditional" specifies classical Greek numbering.

format="&#x0430;" letter-value="traditional" specifies Old Slavic numbering.

**Example**
This example demonstrates the use of the `<xsl:number>` element by numbering items in a sorted list with Western and then lower-case Roman numbering.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (items.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="numelem.xsl" ?>
<items>
  <item>Car</item>
  <item>Pen</item>
  <item>LP Record</item>
  <item>Wisdom</item>
  <item>Cell phone</item>
  <item>Film projector</item>
  <item>Hole</item>
  <item>Canopy</item>
  <item>Widget</item>
  <item>Concept</item>
  <item>Null character</item>
</items>
```

**XSLT File (numelem.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:template match="items">
  <xsl:for-each select="item">
    <xsl:sort select="."/>
    <xsl:number value="position()" format="1. "/>
    <xsl:value-of select="."/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

**Output**

This is the formatted output:
1. Canopy, i) Canopy
2. Car, ii) Car
3. Cell phone, iii) Cell phone
4. Concept, iv) Concept
5. Film projector, v) Film projector
6. Hole, vi) Hole
7. LP Record, vii) LP Record
8. Null character, viii) Null character
9. Pen, ix) Pen
10. Widget, x) Widget
11. Wisdom, xi) Wisdom

This is the processor output:

```xml
<?xml version="1.0" encoding="UTF-16"?>1. Canopy,
   i) Canopy<br />
2. Car,
   ii) Car<br />
3. Cell phone,
   ... xi) Wisdom<br />

See Also

Inserting Automatic Numbering Using XSLT
```
<xsl:otherwise> Element

Provides multiple conditional testing in conjunction with the <xsl:choose> and <xsl:when> elements.

<xsl:otherwise>
</xsl:otherwise>

**Element Information**

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child elements</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**

Provides a default condition for <xsl:choose>. Other alternatives are indicated by <xsl:when> elements.

For simple conditional testing, use the <xsl:if> element.

**Example**

See the example for <xsl:choose>.

**See Also**

[Defining Conditional Templates Using <xsl:if> and <xsl:choose>](#) | [<xsl:when> Element](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:output> Element

Specifies options for use in serializing the result tree.

<xsl:output
  method = "xml" | "html" | "text" | QName
  version = nmtoken
  encoding = string
  omit-xml-declaration = "yes" | "no"
  standalone = "yes" | "no"
  doctype-public = string
  doctype-system = string
  cdata-section-elements = QNames
  indent = "yes" | "no"
  media-type = string
 />

Attributes

method

Identifies the overall method used for outputting the result tree. If there is no prefix, it identifies a method specified in this document that must be one of "xml", "html", "text", or a QName that is not an NCName). If there is a prefix, it is expanded and identifies the output method.

The default for the method attribute is chosen as follows. If any of the following conditions are true, the default output method is "html":

- The root node of the result tree has an element child.
- The expanded-name of the first element child of the root node (that is, the document element) of the result tree has local part "html" (in any combination of uppercase and lowercase) and a null namespace URI.
- Any text nodes preceding the first element child of the root node of the result tree contain only white space characters.

Otherwise, the default output method is "xml". The default output method should be used if there are no <xsl:output> elements or if none of the <xsl:output> elements specifies a value for the method attribute.
version
Specifies version 1.0 in relation to the "xml" output method; this appears in the output XML declaration as 
<?xml version="1.0"?>.

encoding
Specifies the preferred character encoding that the parser should use to encode sequences of characters as sequences of bytes. The value of the attribute is treated case-insensitively; it must contain only printable ASCII characters and be a registered character set, or begin with x-.

omit-xml-declaration
Specifies whether the XSLT processor should output an XML declaration; the value must be yes or no.

standalone
Specifies whether the XSLT processor should output a standalone document declaration; the value must be yes or no.

doctype-public
Specifies the public identifier to be used in the DTD.

doctype-system
Specifies the system identifier to be used in the DTD.

cdata-section-elements
Specifies a list of the names of elements whose text node children should be output using CDATA sections.

indent
Specifies additional white space to add when outputting the result tree; the value must be yes or no.

media-type
Specifies the media type (MIME content type) of the data that results from outputting the result tree.

Element Information

Number of occurrences Unlimited
Parent elements xsl:stylesheet, xsl:transform
Child elements (No child elements)

Remarks
A style sheet can contain multiple <xsl:output> elements and can include style
sheets that also contain `<xs1:output>` elements. All the `<xs1:output>` elements occurring in a style sheet are merged into a single effective `<xs1:output>` element. It is an error if there is more than one such value for an attribute. The values of attributes are defaulted after the `<xs1:output>` elements have been merged; different output methods may have different default values for an attribute.

**Examples**

The following topics provide examples of `<xs1:output>`.

- [Example 1 of `<xs1:output>`](#)
- [Example 2 of `<xs1:output>`](#)
Example 1 of `<xsl:output>`

This is a simple "Hello, World!" demonstration.

**Note** To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (hello.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<messages>
  <message>Hello World!</message>
</messages>
```

**XSLT File (hello.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>

  <xsl:template match="/message">
    <html>
      <head>
        <title>Message</title>
      </head>
      <body>
        <p><xsl:value-of select="."/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

**Output**

This is the formatted output:

Hello World!

This is the processor output:
Hello World!
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Example 2 of <xsl:output>

This example outputs the result tree as HTML.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (booksshort.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="outputelem.xsl"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.</description>
  </book>
</catalog>
```

XSLT File (outputelem.xsl)
Gambardella, MatthewXML Developer's GuideComputer44.952000-10-01An in-depth look at creating applications with XML.Ralls, KimMidnightRainFantasy5.952000-12-16A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.Corets, EvaMaeve AscendantFantasy5.952000-11-17After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.

This is the processor output:

<html>Gambardella, MatthewXML Developer's GuideComputer44.952000-10-01An in-depth look at creating applications with XML.Ralls, KimMidnightRainFantasy5.952000-12-16A former architect battles corporate zombie an evil sorceress, and her own childhood to become queen of the world.Corets, EvaMaeve AscendantFantasy5.952000-11-17After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.</html>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
**<xsl:param> Element**

Declares a named parameter for use within an `<xsl:stylesheet>` element or an `<xsl:template>` element. Allows specification of a default value.

```xml
<xsl:param
    name = QName
    select = Expression>
</xsl:param>
```

**Attributes**

- **name**
  Required. Specifies the name of the parameter.

- **select**
  The value of the attribute is an expression, and the value of the variable is the object that results from evaluating the expression. When this attribute is specified, the `<xsl:param>` element must be empty.

**Element Information**

**Number of occurrences**  Unlimited

**Parent elements**  `<xsl:stylesheet>`, `<xsl:template>`, `<xsl:transform>`
  `<xsl:apply-templates>`, `<xsl:attribute>`, `<xsl:call-template>`,
  `<xsl:choose>`, `<xsl:comment>`, `<xsl:copy>`, `<xsl:copy-of>`,
  output elements

**Child elements**

**Remarks**

The value specified on the `<xsl:param>` element is a default value for binding. When the template or style sheet containing `<xsl:param>` is invoked, parameters are passed that are used in place of the default values.

The `<xsl:param>` element must be declared as a direct child of an
The <xsl:template> element. If not declared as a direct child, the value of the <xsl:param> element will not be accessible and an error will occur. For example:

```xml
<xsl:template name="getcount">
  <xsl:element name="strong">
    <xsl:param name="counted">
      <xsl:value-of select="count(//book)"/>
    </xsl:param>
    Total Book Count: <xsl:value-of select="$counted"/>
  </xsl:element>
</xsl:template>
```

In the previous example, the only direct child of the <xsl:template> element is the <strong> element. As such, the value of the <xsl:param> element cannot be correctly evaluated by the parser and results in the following error:

**Keyword xsl:param may not be used here.**

The correct way to place this element so that it can be evaluated within the context of the <xsl:template> element would be:

```xml
<xsl:template name="getcount">
  <xsl:param name="counted">
    <xsl:value-of select="count(//book)"/>
  </xsl:param>
  <xsl:element name="strong">
    Total Book Count: <xsl:value-of select="$counted"/>
  </xsl:element>
</xsl:template>
```

The value of the parameter can be an object of any type that can be returned by an expression. The <xsl:param> element can specify the value of the variable in three alternative ways:

- If the element has a select attribute, the value of the attribute must be an expression and the value of the parameter is the object that results from evaluating the expression. In this case, the content of the element must be empty.
- If the element does not have a select attribute and has non-empty content such as one or more child nodes, the content specifies the value. The content is a template that is instantiated to give the value of the parameter. The value is a result tree fragment equivalent to a node-set containing just a single root node having as children the sequence of nodes produced by
instantiating the template. The base URI of the nodes in the result tree fragment is the base URI of the element.

An error occurs if a member of the sequence of nodes created by instantiating the template is an attribute node or a namespace node, because a root node cannot have an attribute node or a namespace node as a child.

- If the content is empty and does not have a select attribute, the value of the parameter is an empty string. Thus
  
  `<xsl:param name="x"/>`

  is equivalent to

  `<xsl:param name="x" select="''"/>`

  **Note** When a parameter is used to select nodes by position, be careful not to do the following:
  `<xsl:param name="n">2</xsl:param>`
  ...
  `<xsl:value-of select="item[$n]"/>`

  This will output the value of the first item element, because the variable "n" will be bound to a result tree fragment, not a number. Instead, do either

  `<xsl:param name="n" select="2"/>`
  ...
  `<xsl:value-of select="item[$n]"/>`

  or

  `<xsl:param name="n">2</xsl:param>`
  ...
  `<xsl:value-of select="item[number($n)]"/>`

  **Note** The following is a convenient way to specify the empty node-set as the default value of a parameter.
  `<xsl:param name="x" select="/.."/>`

**Example**

This example defines a named template for a "numbered-block" with an argument to control the format of the number.
Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (catmat.xml)

None; the XSLT file calls itself. View the XSLT file in Internet Explorer.

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="paramelem.xsl"?>
<lists>
  <ol>
    <li>the</li>
    <li>cat</li>
    <ol>
      <li>sat</li>
      <li>on</li>
      <li>the</li>
    </ol>
    <li>mat</li>
  </ol>
</lists>
```

XSLT File (paramelem.xsl)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="paramelem.xsl"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:template match="ol/li">
    <br/>
    <xsl:call-template name="numbered-block"/>
  </xsl:template>

  <xsl:template match="ol//ol/li">
    <br/>
    &nbsp;&nbsp;&nbsp;&nbsp;
    <xsl:call-template name="numbered-block">
      <xsl:with-param name="format">a. </xsl:with-param>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="numbered-block">
    <xsl:param name="format">1. </xsl:param>
    <fo:block>
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```
Output

This is the formatted output:

1. the
2. cat
   a. sat
   b. on
   c. the
3. mat

The following is the processor output, with white space added for clarity.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<br xmlns:fo="http://www.w3.org/1999/XSL/Format" />
<fo:block xmlns:fo="http://www.w3.org/1999/XSL/Format">1. the</fo:block>
<br xmlns:fo="http://www.w3.org/1999/XSL/Format" />
<br xmlns:fo="http://www.w3.org/1999/XSL/Format" />
<br xmlns:fo="http://www.w3.org/1999/XSL/Format" />
<fo:block xmlns:fo="http://www.w3.org/1999/XSL/Format">b. on</fo:block>
<br xmlns:fo="http://www.w3.org/1999/XSL/Format" />
<fo:block xmlns:fo="http://www.w3.org/1999/XSL/Format">c. the</fo:block>
<br xmlns:fo="http://www.w3.org/1999/XSL/Format" />
```

See Also

<xsl:with-param> Element | <xsl:variable> Element | <xsl:call-template>
Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:preserve-space> Element

Preserves white space only text nodes that may appear in those elements as specified by the elements attribute.

<xsl:preserve-space
   elements = tokens />

Attributes

elements
   This is a required attribute. The value is a white space separated list of name tokens of the nodes whose white space only text nodes must be preserved.

Element Information

Number of occurrences Unlimited
Parent elements   xsl:stylesheet, xsl:transform
Child elements    (No child elements)

Remarks

The <xsl:preserve-space> element preserves white-space-only text nodes in the specified elements. It has no effect on the white space characters in text nodes with both white space and non-white-space characters. Here preservation of white-space-only text nodes means that the nodes from the source document will be kept in the result document. The <xsl:strip-space> does the opposite; it strips the white-space-only text nodes in the specified nodes.

By default, all white-space-only text nodes are preserved. If an element name matches a name test in an <xsl:strip-space> element, it is removed from the set of white-space-preserving element names. If an element name matches a name test in an <xsl:preserve-space> element, it is added back to the set of white-space-preserving element names.
For more information, see "Whitespace Stripping", Section 3.4 of the XSLT W3 Recommendation.

**Example**

The following example illustrates the effects of preserving and stripping white-space-only text nodes using `<xsl:preserve-space>` and `<xsl:strip-space>`.

**XML File (source.xml)**

```xml
<?xml version="1.0"?>
<document>
<text>
	
</text>
<text> ;
</text>
<text>
This is a sample text
</text>
<code>
	
</code>
<code> ;
</code>
<code>
This is a sample code
</code>
</document>
```

**XSLT File (trans.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
    <!--
    The following xsl:preserve-space is not necessary.
    It is included to emphasize the fact that white-space-only
text nodes are to be preserved on the <code> elements.
    -->
    <xsl:preserve-space elements="code"/>
    <xsl:strip-space elements="text"/>

    <xsl:template match="/">
        code elements:
        <xsl:apply-templates select="/code"/>

        text elements:
        <xsl:apply-templates select="/text"/>
    </xsl:template>
</xsl:stylesheet>
```
Try It!

1. Copy the code above and save it in appropriate files on your local drive.
2. Launch the XSLT transformation, using the `msxsl.exe` utility from a command prompt, as follows:

   
   ```
   msxsl source.xml trans.xsl
   ```

   **Important**  Do not start the transformation from Internet Explorer. The browser performs some space-stripping operations that are not compatible with the XSLT specifications. This can cause the XSLT transformation to appear ill-behaved.

Output

This is the standard output:

```

code elements:

code # 1
has 1 text().
"---"

code # 2
has 1 text().
"--;"

code # 3
has 1 text().
```
"NThis-is-a-Tsample-codeTNTN"

text elements:

text # 1
has 0 text().
"

  text # 2
has 1 text().
"--;"

  text # 3
has 1 text().
"NThis-is-a-Tsample-textTNTN"

Notice that the transformation yields one text node for the first <code> element, but no text node for the first <text> element. This is because these elements both have a white-space-only text node that is preserved in <code> but stripped in <text>, according to the <xsl:preserve-space> and <xsl:strip-space> instructions listed at the beginning of the style sheet. The second and third elements of each kind show that the text nodes that are not white-space-only are not affected by those instructions.

See Also

<xsl:strip-space> Element | Controlling White Space with XSLT
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:processing-instruction> Element

Generates a processing instruction node in the output.

```xml
<xsl:processing-instruction
    name = "pi-name">
</xsl:processing-instruction>
```

Attributes

name

Required. The NCName of the processing instruction.

Element Information

Number of occurrences

Unlimited

Parent elements

```
```

Child elements

Remarks

The `<xsl:processing-instruction>` element generates a processing instruction node in the output. The name is indicated by the `name` attribute. The content of the element provides the rest of the processing instruction.

The XML declaration is not a processing instruction, and should be generated by setting attributes on an `<xsl:output>` element.
Example

This example shows a template that generates the XML declaration and a style sheet processing instruction in the output.

XML File (customers.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="pi.xsl" ?>
<customers>
  <customer>
    <name>James Smith</name>
    <address>123 Elm St.</address>
    <phone>(123) 456-7890</phone>
  </customer>
  <customer>
    <name>Amy Jones</name>
    <address>456 Oak Ave.</address>
    <phone>(156) 789-0123</phone>
  </customer>
</customers>
```

XSLT File (pi.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method='xml' version='1.0'/>
  <xsl:template match="/">
    <xsl:processing-instruction name="xml-stylesheet">
      <xsl:text type="text/xsl" href="style.xsl"/>
    </xsl:processing-instruction>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="@* | *">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Try It!

1. Copy the code above and save it in appropriate files on your local drive.
2. Run the example, using the `msxsl.exe` utility from the command
prompt, as follows:

msxsl customers.xml pi.xsl -o new-cust.xml

Output

The result of this transformation is the same XML file with a new style sheet embeded in it. The output file, new-cust.xml, should look as follows:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<customers>
  <customer>
    <name>James Smith</name>
    <address>123 Elm St.</address>
    <phone>(123) 456-7890</phone>
  </customer>
  <customer>
    <name>Amy Jones</name>
    <address>456 Oak Ave.</address>
    <phone>(156) 789-0123</phone>
  </customer>
</customers>
```

See Also

Generating Comments, Processing Instructions, and Elements
<msxsl:script> Element

Contains script blocks, so that custom functions can be used in an XSLT transformation. This is a top-level element.

```xml
<msxsl:script
    language = "language-name"
    implements-prefix = "prefix of user's namespace">
</msxsl:script>
```

Attributes

**language**
The Active Scripting language used for the functions defined within this element. If left unspecified, Microsoft® JScript® (compatible with ECMA 262 language specification) is used. This attribute accepts the same values as the `language` attribute on the HTML `<SCRIPT>` element.

**implements-prefix**
Required. Declares a namespace and associates it with the script block. The value of this attribute is the prefix that represents the namespace.

Element Information

**Number of occurrences** Unlimited

**Parent elements** `xsl:stylesheet`, `xsl:transform`

**Child elements** (No child elements)

Remarks

The `<msxsl:script>` element belongs to the namespace urn:schemas-microsoft-com:xslt. You can declare variables and define functions within this element. This element can appear within the `<xsl:stylesheet>` element. A script block thus nested is treated as a global script block.

You can also instantiate COM objects in the `<msxsl:script>` element. However, a user's security settings might prevent your script from instantiating a client-
side object.

For high performance, avoid script blocks from XSLT files, because they require loading the script engine multiple times. Instead, create a COM object that is equivalent to the script, producing a DLL. Then use addObject and pass the DLL.

**Example**

This example illustrates how to use `<msxsl:script>` to define a script block with a namespace prefix of `user` to declare a function called `xml()`. The `xml()` function takes a node-list as an argument. Notice how this function, `xml(nodeList)` in the `user` namespace, is called from the select attribute of `<xsl:value-of>`.

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (customers.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="script.xsl" ?>
<customers>
  <customer>
    <name>John Smith</name>
    <address>123 Elm St.</address>
    <phone>(123) 456-7890</phone>
  </customer>
  <customer>
    <name>Mary Jones</name>
    <address>456 Oak Ave.</address>
    <phone>(156) 789-0123</phone>
  </customer>
</customers>
```

**XSLT File (script.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version='1.0'
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="JScript" implements-prefix="user">
```

```xml
```
function xml(nodelist) {
    return nodelist.nextNode().xml;
}
</msxsl:script>

<xsl:template match="/">
    <xsl:value-of select="user:xml(.)"/>
</xsl:template>

</xsl:stylesheet>

Output

This is the formatted output:

<?xml version="1.0"?>  
<?xml-stylesheet type="text/xsl" href="script.xsl"?>
<customers>
    <customer>
        <name>John Smith</name>
        <address>123 Elm St.</address>
        <phone>(123) 456-7890</phone>
    </customer>
    <customer>
        <name>Mary Jones</name>
        <address>456 Oak Ave.</address>
        <phone>(156) 789-0123</phone>
    </customer>
</customers>

This is the processor output:

<?xml version="1.0" encoding="UTF-16"?>&lt;?xml version="1.0"?&gt;
&lt;?xml-stylesheet type="text/xsl" href="script.xsl"?&gt;
&lt;customers&gt;
    &lt;customer&gt;
        &lt;name&gt;John Smith&lt;/name&gt;
        &lt;address&gt;123 Elm St.&lt;/address&gt;
        &lt;phone&gt;(123) 456-7890&lt;/phone&gt;
    &lt;/customer&gt;
    &lt;customer&gt;
        &lt;name&gt;Mary Jones&lt;/name&gt;
        &lt;address&gt;456 Oak Ave.&lt;/address&gt;
        &lt;phone&gt;(156) 789-0123&lt;/phone&gt;
    &lt;/customer&gt;
&lt;/customers&gt;

See Also

<xsl:value-of> Element
<xsl:sort> Element

Specifies sort criteria for node lists selected by <xsl:for-each> or <xsl:apply-templates>.

<xsl:sort
   select = "string-Expression"
   lang = { nmtoken }
   data-type = { "text" | "number" | QName }
   order = { "ascending" | "descending" }
   case-order = { "upper-first" | "lower-first" }
/>
If no data type is specified, the type of the expression will be used as the default.

**order**
The sort order for the strings. The default value is "ascending".

**case-order**
The order in which to sort strings by capitalization. The default value is "upper-first", which sorts the strings with uppercase letter first.

---

**Element Information**

**Number of occurrences** Unlimited

**Parent elements**  
- xsl:apply-templates
- xsl:for-each

**Child elements**  
(No child elements)

---

**Remarks**

This sorting algorithm ignores dashes when sorting. This is the "word-sort" method, intended to sort words like "co-op" next to "coop".

For example, the following sort order is produced:

- 30a-allow
- 30-allow

Thus the sort treats the items as though they were:

- 30aallow
- 30allow

---

**Example**

See the example in [Sorting XML Using <xsl:sort>].

---

**See Also**
Sorting XML Using <xsl:sort>
<xsl:strip-space> Element

Strips white-space-only text nodes that might appear in elements specified by the elements attributes.

<xsl:strip-space
   elements = tokens />

Attributes

elements

The value of this required attribute is a white-space-separated list of name tokens of the nodes whose white-space-only text nodes should be stripped.

Element Information

Number of occurrences Unlimited
Parent elements xsl:stylesheet, xsl:transform
Child elements (No child elements)

Remarks

The <xsl:strip-space> element strips white-space-only text nodes in the specified elements.

By default, all white-space-only text nodes are preserved. If an element name matches a name test in an <xsl:strip-space> element, it is removed from the set of white-space-preserving element names. If an element name matches a name test in an <xsl:preserve-space> element, then it is added back to the set of white-space-preserving element names.

For more information, see "Whitespace Stripping", Section 3.4 of XSL Transformation (XSLT) Version 1.0, W3C Recommendation 16 November, 1999.

Example
See the example in `<xsl:preserve-space>` Element.

**See Also**

[Controlling White Space with XSLT](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:stylesheet> Element

Specifies the document element of an XSLT file, which contains all other XSLT elements. This element is a synonym for the <xsl:transform> element.

<xsl:stylesheet
  id = "id"
  extension-element-prefixes = "NCNames"
  exclude-result-prefixes = "NCNames"
  version = "number">
</xsl:stylesheet>

Attributes

id
A unique identifier to facilitate embedding XSLT files.

extension-element-prefixes
The namespace to be used as an extension namespace. The value is a white-space-separated list of namespace prefixes. The namespace bound to each of the prefixes is designated as an extension namespace. The default namespace (as declared by xmlns) can be designated as an extension namespace by including #default in the list of namespace prefixes. The designation of a namespace as an extension namespace is effective within the subtree of the style sheet rooted at the element bearing the extension-element-prefixes; a subtree rooted at an <xsl:stylesheet> element does not include any style sheets imported or included by children of that <xsl:stylesheet> element.

exclude-result-prefixes
The namespace Uniform Resource Identifier (URI) as an excluded namespace. The value is a white-space separated list of namespace prefixes. The namespace bound to each of the prefixes is designated as an excluded namespace. The default namespace (as declared by xmlns) may be designated as an excluded namespace by including #default in the list of namespace prefixes. The designation of a namespace as an excluded namespace is effective within the subtree of the style sheet rooted at the element bearing the exclude-result-prefixes; a subtree rooted at an <xsl:stylesheet> element does not include any style sheets imported or
included by children of that <xsl:stylesheet> element.

**version**
Required. The version of XSLT that the XSLT file requires. Value should be set to "1.0" for this version of XSLT.

**Element Information**

**Number of occurrences** One
**Parent elements** (No parent elements)
**Child elements**
- xsl:attribute-set
- xsl:import
- xsl:include
- xsl:output
- xsl:param
- xsl:template
- xsl:variable
- msxsl:script

**Remarks**

Every XSTL file must declare <xsl:stylesheet> as its document element.

A style sheet can use a series of template rule declarations (<xsl:template>) to prescribe explicit transformations on a data set. It can use <xsl:include> and/or <xsl:import> to incorporate other style sheets; this allows you to reuse existing, tested template rules. The style sheet can also support global variables (<xsl:param>) and named constants (<xsl:variable>), to be used throughout the transformations. The <xsl:param> element can be used to pass parameters into a style sheet. By using the extension element, <msxsl:script>, a style sheet can even allow custom functions to be implemented, although this practice is generally discouraged.

The following example shows an <xsl:stylesheet> element with several common namespaces.

An XSLT style sheet must declare at least the namespace for the XSL transformation. In MXSML versions 4.0 and later, this namespace must be declared as follows:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

If elements or data types from other namespaces are processed, the style sheet must declare the related namespaces as well. For example, the following XSLT
style sheet declaration stipulates that additional XML vocabularies, which are defined for Microsoft extensions and XSL Formatting Objects (XSL-FO), are to be used.

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

**Example**

This example shows a complete XSLT file that contains a set of templates. The root template (`match="/"`) defines the structure of the overall output document, and the other templates define the structure of the `<name>`, `<address>`, and `<phone>` elements. This HTML output file was produced by using `msxml.exe` at the command prompt.

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (customers.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="customers.xsl"?>
<customers>
    <customer>
        <name>John Smith</name>
        <address>123 Elm St.</address>
        <phone>(123) 456-7890</phone>
    </customer>
    <customer>
        <name>Mary Jones</name>
        <address>456 Oak Ave.</address>
        <phone>(156) 789-0123</phone>
    </customer>
</customers>
```

**XSLT File (customers.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```xml
<xsl:template match="/"/>
```
Output

This is the formatted output:

John Smith 123 Elm St. (123) 456-7890
Mary Jones 456 Oak Ave. (156) 789-0123

This is the processor output:

<HTML>
<BODY>
<TABLE>
<TR>
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>123 Elm St.</td>
<td>(123) 456-7890</td>
</tr>
<tr>
<td>Mary Jones</td>
<td>456 Oak Ave.</td>
<td>(156) 789-0123</td>
</tr>
</tbody>
</table>

**See Also**

<xsl:transform> Element | Working with Namespaces and Prefix Aliases
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:template> Element

Defines a reusable template for generating the desired output for nodes of a particular type and context.

```xml
<xsl:template
    name= Qname
    match = Pattern
    priority = number
    mode = QName
</xsl:template>
```

Attributes

name
The QName to be expanded. If it has a prefix, the prefix is expanded into a Uniform Resource Identifier (URI) reference using the namespace declarations in effect on the attribute in which the name occurs. The expanded-name, consisting of the local part of the name and the possibly null URI reference is used as the name of the template. The default namespace is not used for unprefixed names. If an <xsl:template> element has a name attribute, it can, but need not, also have a match attribute.

match
A pattern that identifies the source node or nodes to which the rule applies. The match attribute is required unless the <xsl:template> element has a name attribute. The content of the <xsl:template> element is the template that is instantiated when the template rule is applied.

priority
The priority number for the template. All matching template rules that have lower priority than the matching template rule or rules with the highest priority are eliminated from consideration. The value of this must be a real number from 0–9, positive or negative, matching the production number with an optional leading minus sign (−). The default priority is computed as follows:

- If the pattern contains multiple alternatives separated by |, it is treated equivalently to a set of template rules, one for each alternative.
- If the pattern has the form of a QName preceded by a child or attribute...
axis specifier, or has the form processing-instruction literal preceded by a child or attribute axis specifier, the priority is 0.

- If the pattern is a name preceded by a child or attribute axis specifier, the priority is -0.25.
- Otherwise, if the pattern consists of just a node test preceded by a child or attribute axis specifier, the priority is -0.5.
- Otherwise, the priority is 0.5.

Thus the most common kind of pattern (a pattern that tests for a node with a particular type and a particular expanded-name) has priority 0. The next less specific kind of pattern (a pattern that tests for a node with a particular type and an expanded-name with a particular namespace URI) has priority -0.25. Patterns less specific than this (patterns that just test for nodes with particular types) have priority -0.5. Patterns more specific than the most common kind of pattern have priority 0.5.

**mode**

The mode value. This value allows an element to be processed multiple times, each time producing a different result. If `<xsl:template>` does not have a match attribute, it must not have a mode attribute. If an `<xsl:apply-templates>` element has a mode attribute, it applies only to those template rules from `<xsl:template>` elements that have a mode attribute with the same value; if an `<xsl:apply-templates>` element does not have a mode attribute, it applies only to those template rules from `<xsl:template>` elements that do not have a mode attribute.

**Element Information**

<table>
<thead>
<tr>
<th><strong>Number of occurrences</strong></th>
<th>Unlimited</th>
</tr>
</thead>
</table>
| **Parent elements**       | `<xsl:stylesheet`, `<xsl:transform`
| **Child elements**        |           |

**Remarks**
Note that the template need not generate a complete XML document (even the root template, unless using `transformNodeToObject`), but only a fragment of XML. It is possible to include unenclosed text or multiple document elements defined by the template. This facilitates the generation of raw text and XML fragments that can be further processed by an application (for example, HTML fragments inserted into an HTML page).

The value of the `name` attribute is a QName that is expanded. If it has a prefix, it is expanded into a URI reference using the namespace declarations in effect on the attribute in which the name occurs. The expanded-name consisting of the local part of the name and the possibly null URI reference is used as the name of the template. The default namespace is not used for unprefixed names.

If an `<xsl:template>` element has a `name` attribute, it can, but need not, also have a `match` attribute. An `<xsl:call-template>` element invokes a template by name; it has a required `name` attribute that identifies the template to be invoked. Unlike `<xsl:apply-templates>`, `<xsl:call-template>` does not change the current node or the current node list.

An error occurs if a style sheet contains more than one template with the same name.

**Example**

This template rule has a pattern that identifies `<stock>` elements and produces an output `<DIV>` element with the attribute `STYLE="font-weight:bold"`:

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (portfolio.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="templ.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="preserve">
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
```

The value of the `name` attribute is a QName that is expanded. If it has a prefix, it is expanded into a URI reference using the namespace declarations in effect on the attribute in which the name occurs. The expanded-name consisting of the local part of the name and the possibly null URI reference is used as the name of the template. The default namespace is not used for unprefixed names.

If an `<xsl:template>` element has a `name` attribute, it can, but need not, also have a `match` attribute. An `<xsl:call-template>` element invokes a template by name; it has a required `name` attribute that identifies the template to be invoked. Unlike `<xsl:apply-templates>`, `<xsl:call-template>` does not change the current node or the current node list.

An error occurs if a style sheet contains more than one template with the same name.

**Example**

This template rule has a pattern that identifies `<stock>` elements and produces an output `<DIV>` element with the attribute `STYLE="font-weight:bold"`:

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (portfolio.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="templ.xsl"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes" xml:space="preserve">
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
```
XSLT File (templ.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="stock">
    <DIV STYLE="font-weight:bold">
        Symbol: <xsl:value-of select="symbol"/>
        Price: <xsl:value-of select="price"/>
    </DIV>
</xsl:template>

</xsl:stylesheet>
```

Output

This is the formatted output:

**Symbol: ZCXM, Price: 28.875**

**Symbol: ZFFX, Price: 92.250**

**Symbol: ZYSZ, Price: 20.313**

See Also

[transformNodeToObject Method](#)
<xsl:text> Element

Generates text node from a style sheet. White-space-only nodes are preserved in the output.

<xsl:text disable-output-escaping = "yes" | "no">
</xsl:text>

Attributes

disable-output-escaping
Default is "no". If the value is "yes", a text node generated by instantiating the <xsl:text> element will be output without any escaping. For example, the following generates the single character "<".

<xsl:text disable-output-escaping="yes">&lt;</xsl:text>

Note Because disable-output-escaping="yes" can be used to generate non-well-formed documents, it should be used with caution. Output that is not well-formed can generate errors in certain circumstances. For example, transformNodeToObject to an XML document requires that the result be well-formed and thus might not complete if disable-output-escaping has affected the well-formedness of the document. Consider disable-output-escaping="yes" an advanced feature, to be used only when the potential dangers are understood.

Element Information

Number of occurrences Unlimited
Child elements (No child elements)
Remarks

In a style sheet, text can be generated to the literal result tree with or without `<xsl:text>`. However, with this element you can exert some control over the white space created by the style sheet. For example, to make your style sheet more readable, you might want to write one element per line in a template, and indent some lines. Doing so introduces white space as part of the template rule. This might or might not be a desired effect of the transformation.

Sometimes you might want to introduce a white space character to separate two data values. You can use an `<xsl:text>` element to accomplish this. White space enclosed within `<xsl:text>` is output to the result tree. Thus, the following template—

```xml
<xsl:template match="a-node">
  <xsl:text>
</xsl:text>
</xsl:template>
```

—will always output a new line text node in the result tree. However, white-space-only text nodes not enclosed by `<xsl:text>` will be stripped away from the result tree. The example below how to use empty `<xsl:text/>` elements to this.

Example

**XML File (text.xml)**

```xml
<?xml version="1.0"?>
<topic>
  <text>First line.</text>
  <text>Second line.</text>
  <text></text>
</topic>
```

**XSLT File (text.xsl)**

The following style sheet employs empty `<xsl:text/>` elements to strip all white space characters (spaces, newlines, and tabs) that would otherwise have been generated by the template rules. The result is shown in the formatted
output, below.

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="text"/>

   <xsl:template match="/">
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="text">
      <xsl:text>
<xsl:value-of select="."/>
</xsl:text>
   </xsl:template>
</xsl:stylesheet>

HTML File (text.htm)

The following HTML file can be used to try the XSLT transformation and see the results.

<html>
<head>
<title></title>
</head>
<body onload="init()">
<div><input type="text" id="xmlName" value="text.xml"></div>
<div><input type="text" id="xslName" value="text.xsl"></div>
<div><input type=button value="transform" onclick="trans();"></div>
<div id="divErr"></div>
<pre id="preRes" style="background:blue;color:gold"></pre>
</body>
<script language="javascript">
function trans()
{ xmlFile=xmName.value; 
  xslFile=xslName.value; 
  if (xmlFile == "" || xslFile == ")
  {
    divErr.innerHTML = "invalid xml/xsl file names.";
  }
  var xsl = new ActiveXObject("MSXML2.DOMDOCUMENT.5.0"); 
  var xml = new ActiveXObject("MSXML2.DOMDocument.5.0"); 
  xml.validateOnParse = false; 
  xml.async = false; 
  xml.load(xmlFile); 
  if (xml.parseError.errorCode != 0) 
    divErr.innerHTML = "XML Parse Error : " + xml.parseError.reason;
  xsl.async = false; 
  xsl.load(xslFile); 
  if (xsl.parseError.errorCode != 0) 
    divErr.innerHTML = "XSL Parse Error : " + xsl.parseError.reason;
  
  try 
  {
    res = xml.transformNode(xsl.documentElement);
    preRes.innerText = res;
  } 
  catch(err)
Try It!

1. Copy the sample code into appropriate files, and save the files to your local drive.
2. Double-click your HTML file, text.htm.
3. Click the transform button on the Web page that appears.

Output

When the XSLT style sheet above is used as is, all text values are joined, head-to-end, in a single line of output:

"First line." "Second line.""

If you remove both <xsl:text/> elements from the template matching text elements, you get the following output (in 7 lines):

"First line."
"Second line."
"

If you remove only the first <xsl:text/> element from the template, you get the following output (in 4 lines):

"First line."
"Second line."
""
See Also

<xsl:comment> Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:transform> Element


<xsl:transform
  id = "id"
  extension-element-prefixes = "tokens"
  exclude-result-prefixes = "tokens"
  version = "number">
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:transform>

Attributes

id

A unique identifier that facilitates embedding style sheets.

extension-element-prefixes

The namespace to be used as an extension namespace. The value is a white-space-separated list of namespace prefixes. The namespace bound to each of the prefixes is designated as an extension namespace. The default namespace (as declared by xmlns) can be designated as an extension namespace by including #default in the list of namespace prefixes. The designation of a namespace as an extension namespace is effective within the subtree of the style sheet rooted at the element bearing the extension-element-prefixes; a subtree rooted at an <xsl:stylesheet> element does not include any style sheets imported or included by children of that <xsl:stylesheet> element.

exclude-result-prefixes

The namespace URI to be used as an excluded namespace. The value is a white-space-separated list of namespace prefixes. The namespace bound to each of the prefixes is designated as an excluded namespace. The default namespace (as declared by xmlns) may be designated as an excluded namespace by including #default in the list of namespace prefixes. The designation of a namespace as an excluded namespace is effective within the subtree of the style sheet rooted at the element bearing the exclude-result-prefixes; a subtree rooted at an <xsl:stylesheet> element does not include any style sheets imported or included by children of that
<xsl:stylesheet> element.

version
Required. The version of XSLT that the XSLT file requires. The value should be set to "1.0" for this version of XSLT.

Element Information

Number of occurrences One
Parent elements (No parent elements)

Remarks

A synonym for the <xsl:stylesheet> element. This element can have a set of <xsl:template> elements representing different output templates. Processing begins by processing the root template, indicated by the pattern "/".

Example

This style sheet outputs a text string, and begins with <xsl:transform>, rather than <xsl:stylesheet>.

   Note   To test this example in Internet Explorer, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (mymin.xml)

<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="helloxslt.xsl"?>
<myelem/>

XSLT File (helloxslt.xsl)

<?xml version='1.0'?>
<xsl:transform version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
</xsl:template match="/"
<html>
  <body><p>Hello from XSLT!</p></body>
</html>
</xsl:template>
</xsl:transform>

**Output**

This is the formatted output:

Hello from XSLT!

This is the processor output:

<html>
<body><p>Hello from XSLT!</p></body>
</html>

**See Also**

<xsl:stylesheet> Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Reference
<xsl:value-of> Element

Inserts the value of the selected node as text.

```xml
<xsl:value-of
  select = Expression
  disable-output-escaping = "yes" | "no"
</xsl:value-of>
```

Attributes

select

Required. The expression to be evaluated against the current context. The results are converted to a string, as by a call to the string() function. A node-set is converted to a string by inserting the string value of the first node in the set.

disable-output-escaping

Default is "no". If the value is "yes", a text node generated by instantiating the <xsl:value-of> element will be output without any escaping. For example, the following generates the single character "<".

```xml
<xsl:value-of disable-output-escaping="yes" select="string('&lt;')"/>
```

Note Because disable-output-escaping="yes" can be used to generate non-well-formed documents, it should be used with caution. Output that is not well-formed can generate errors in certain circumstances. For example, transformNodeToObject to an XML document requires that the result be well-formed, and thus might not complete if disable-output-escaping has affected the well-formedness of the document. Consider disable-output-escaping="yes" an advanced feature, to be used only when the potential dangers are understood.

Element Information

Number of occurrences Unlimited

The `<xsl:value-of>` element inserts a text string representing the value of the first element (in document order) specified by the `select` attribute.

If the XML Path Language (XPath) expression returns more than a single node, the `<xsl:value-of>` element returns the text of the first node returned (equivalent to the `XMLDOMNode` object's `selectSingleNode` method). If the node returned is an element with substructure, `<xsl:value-of>` returns the concatenated text nodes of that element's subtree with the markup removed.

### Example

The following XSLT file creates a `<p>` element from a `<person>` element with `<given-name>` and `<family-name>` child elements. The `<p>` element will contain the string value of the first `<given-name>` child element of the current node, followed by a space and the string value of the first `<family-name>` child element of the current node.

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

### XML File (family.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="family.xsl"?>
<family>
  <person>
    <given-name age="10">Fred</given-name>
    <family-name>Smith</family-name>
  </person>
  <person>
    <given-name age="13">Jill</given-name>
    <family-name>Jones</family-name>
  </person>
</family>
```
</family>

XSLT File (family.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="person">
        <p>
            <xsl:value-of select="given-name"/>
            <xsl:text>	</xsl:text>
            <xsl:value-of select="family-name"/>
        </p>
    </xsl:template>
</xsl:stylesheet>
```

Output

This is the formatted output:

Fred Smith

Jill Jones

This is the processor output:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<p>Fred Smith</p>
<p>Jill Jones</p>
```

See Also

[Creating and Populating an HTML Template](#) | [selectSingleNode Method](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:variable> Element

Specifies a value bound in an expression.

```xml
<xsl:variable
    name = QName
    select = Expression>
</xsl:variable>
```

**Attributes**

- **name**
  - Required. The name of the variable.

- **select**
  - The value of the attribute is an expression, and the value of the variable is the object that results from evaluating the expression. When this attribute is specified, the content of `<xsl:variable>` must be empty.

**Element Information**

<table>
<thead>
<tr>
<th>Number of occurrences</th>
<th>Unlimited</th>
</tr>
</thead>
</table>

**Remarks**

The value of the variable may be an object of any type that can be returned by an expression. The `<xsl:variable>` element can specify the value of the variable in
three alternative ways:

- If the element has a select attribute, the value of the attribute must be an expression and the value of the variable is the object that results from evaluating the expression. In this case, the content of the element must be empty.

- If the element does not have a select attribute and has non-empty content such as one or more child nodes, the content specifies the value. The content is a template that is instantiated to give the value of the variable. The value is a result tree fragment equivalent to a node-set containing just a single root node having as children the sequence of nodes produced by instantiating the template. The base URI of the nodes in the result tree fragment is the base Uniform Resource Identifier (URI) of the variable-binding element.

  An error occurs if a member of the sequence of nodes created by instantiating the template is an attribute node or a namespace node, because a root node cannot have an attribute node or a namespace node as a child.

- If the content is empty and does not have a select attribute, the value of the variable is an empty string. Thus

  `<xsl:variable name="x"/>`

  is equivalent to

  `<xsl:variable name="x" select="''"/>`

  **Note** When a variable is used to select nodes by position, be careful not to do the following.

  `<xsl:variable name="n">2</xsl:variable> ... <xsl:value-of select="item[$n]"/>`

  This will output the value of the first item element, because the variable "n" will be bound to a result-tree fragment, not a number. Instead, do either

  `<xsl:variable name="n" select="2"/> ... <xsl:value-of select="item[$n]"/>`
—or—

<xsl:variable name="n">2</xsl:variable>
...
<xsl:value-of select="item[number($n)]"/>

**Note** The following is a convenient way to specify the empty node-set as the default value of a parameter.

<xsl:param name="x" select="/.."/>

**Example**

This example shows how to define a variable based on the values of other variables.

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (books.xml)**

Use the Sample XML File (books.xml). Below the line `<?xml version='1.0'?>`, add the following line:

`<?xml-stylesheet type="text/xsl" href="sample.xsl"?>`

**XSLT File (sample.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
<html>
<body>
    <xsl:variable name="bookCount" select="count(//book)"/>
    <xsl:variable name="bookTotal" select="sum(//book/price)"/>
    <xsl:variable name="bookAverage" select="$bookTotal div $bookC">
    <table border="1">
    <tr>
        <td>
            <b>Title</b>
        </td>
        <td>
            <b>Description</b>
        </td>
        <td>
            <b>Price</b>
        </td>
    </tr>
    </table>
</body>
</html>
```
Output

This is the formatted output:
<table>
<thead>
<tr>
<th>Title</th>
<th>Price</th>
<th>Average</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Developer's Guide</td>
<td>44.95</td>
<td>17.87</td>
<td>27.08</td>
</tr>
<tr>
<td>Midnight Rain</td>
<td>5.95</td>
<td>17.87</td>
<td>-11.92</td>
</tr>
<tr>
<td>Maeve Ascendant</td>
<td>5.95</td>
<td>17.87</td>
<td>-11.92</td>
</tr>
<tr>
<td>Oberon's Legacy</td>
<td>5.95</td>
<td>17.87</td>
<td>-11.92</td>
</tr>
<tr>
<td>The Sundered Grail</td>
<td>5.95</td>
<td>17.87</td>
<td>-11.92</td>
</tr>
<tr>
<td>Lover Birds</td>
<td>4.95</td>
<td>17.87</td>
<td>-12.92</td>
</tr>
<tr>
<td>Splish Splash</td>
<td>4.95</td>
<td>17.87</td>
<td>-12.92</td>
</tr>
<tr>
<td>Creepy Crawlies</td>
<td>4.95</td>
<td>17.87</td>
<td>-12.92</td>
</tr>
<tr>
<td>Paradox Lost</td>
<td>6.95</td>
<td>17.87</td>
<td>-10.92</td>
</tr>
<tr>
<td>Microsoft .NET: The Programming Bible</td>
<td>36.95</td>
<td>17.87</td>
<td>19.08</td>
</tr>
<tr>
<td>MSXML3: A Comprehensive Guide</td>
<td>36.95</td>
<td>17.87</td>
<td>19.08</td>
</tr>
<tr>
<td>Visual Studio 7: A Comprehensive Guide</td>
<td>49.95</td>
<td>17.87</td>
<td>32.08</td>
</tr>
</tbody>
</table>

**See Also**

<xsl:param> Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
<xsl:when> Element

Provides multiple conditional testing in conjunction with the <xsl:choose> and <xsl:otherwise> elements.

<xsl:when
    test = boolean-Expression>
</xsl:when>

Attributes

test
    Required. The condition in the source data to test. If the expression in this attribute evaluates to True when cast to a Boolean, the content of <xsl:when> is instantiated and placed in the output. Node-sets are cast to a Boolean True if they contain at least one node.

Element Information

Number of occurrences  Unlimited
Parent elements        xsl:choose
                      xsl:apply-templates, xsl:attribute, xsl:call-template,
                      xsl:choose, xsl:comment, xsl:copy, xsl:copy-of,
                      xsl:element, xsl:for-each, xsl:if, xsl:processing-
                      instruction, xsl:value-of, xsl:variable
Child elements         output
                      elements

Remarks

Describes one of the alternatives to be chosen by the <xsl:choose> element. The default alternative is described by the <xsl:otherwise> element.

For simple conditional testing, use the <xsl:if> element.

Example
See the example in `<xsl:choose>` Element.

**See Also**

- Defining Conditional Templates Using `<xsl:if>` and `<xsl:choose>`
- `<xsl:otherwise>` Element
- `<xsl:if>` Element
**<xsl:with-param> Element**

Passes a parameter to a template.

```xml
<xsl:with-param
    name = QName
    select = Expression>
</xsl:with-param>
```

**Attributes**

- **name**
  Required. The name of the parameter.

- **select**
  An expression to be matched against the current context. There is no default value. An empty string is generated if there is no content.

**Element Information**

- **Number of occurrences**: Unlimited

**Remarks**

The `name` attribute is required. It specifies the name of the parameter. The parameter is the variable the value of whose binding is to be replaced.

The `<xsl:with-param>` element is allowed within both `<xsl:call-template>` and `<xsl:apply-templates>`.

The value of the parameter is specified in the same way as for `<xsl:variable>` and `<xsl:param>`.
The current node and current node-list used for computing the value specified by the \(<\text{xsl:with-param}>\) element is the same as that used for the \(<\text{xsl:apply-templates}>\) or \(<\text{xsl:call-template}>\) element within which it occurs.

If you pass a parameter \(x\) to a template that does not have an \(<\text{xsl:param}>\) element for \(x\), this is not an error; the parameter is simply ignored.

**Example**

A style sheet can use the following approach to call localized message strings.

The messages for a language \textit{somelanguage} are stored in an XML file, \texttt{resources/languageabbreviation.xml}, in the form shown in the sample XML file below.

**Note** To test this example in Internet Explorer, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (app.xml)**

```xml
<?xml version='1.0'?>
<<?xml-stylesheet type="text/xsl" href="showlocmsg.xsl" ?>
<showmsg>
  <msg23/>
</showmsg>
```

**XML Resource File (resources/en.xml)**

```xml
<?xml version='1.0'?>
<messages>
  <message name="msg23">Error 23: The drive is full.</message>
  <message name="msg42">Error 42: The file is not found.</message>
</messages>
```

**XSLT File (showlocmsg.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- These 2 elements effectively assign
       $messages = resources/en.xml/<messages>,
    then $messages is used in the "localized-message" template. -->
```
This is the formatted output:

The XML page cannot be displayed

Cannot view XML input using XSL style sheet. Please correct the error and then click the Refresh button, or try again later.

Error 23: The drive is full.

This is the processor output:

<?xml version="1.0" encoding="UTF-16"?>

See Also

<xsl:call-template> Element | <xsl:variable> Element | <xsl:param> Element | 
<xsl:template> Element

Other Resources

XSLT Samples Viewer
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
XSLT Functions

XSLT functions are used as part of the XPath expressions in an XSLT style sheet to access the current node (current()), merge different XML data files together (document()), maintain version compatibility (element-available() or function-available()), format numbers (format-number()), or check system properties. The following table is a complete list of the XSLT functions as specified in the W3C standard and implemented in MSXML versions 4.0 and later.

Notice that these functions can only be called from within the XSLT processor. Thus, they are not callable from the selectNodes method in the Document Object Model (DOM). This is different from the XPath functions, which can be specified inside the argument to the selectNodes method.

XSLT Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>Returns a node set that has the current node as its only member.</td>
</tr>
<tr>
<td></td>
<td>Provides a way to retrieve other XML resources from within the XSLT style</td>
</tr>
<tr>
<td></td>
<td>sheet beyond the initial data provided by the input stream.</td>
</tr>
<tr>
<td>document</td>
<td>Returns True if and only if the expanded-name is the name of an instruction.</td>
</tr>
<tr>
<td>element-available</td>
<td>Converts the first argument to a string using the format pattern string</td>
</tr>
<tr>
<td></td>
<td>specified by the second argument.</td>
</tr>
<tr>
<td>function-available</td>
<td>Returns True if the function is available in the function library.</td>
</tr>
<tr>
<td>generate-id</td>
<td>Returns a string that uniquely identifies the node in the node-set argument</td>
</tr>
<tr>
<td></td>
<td>that is first in document order.</td>
</tr>
<tr>
<td>key</td>
<td>Retrieves elements previously marked with an <a href="">xsl:key</a> statement.</td>
</tr>
<tr>
<td>node-set</td>
<td>Converts a tree into a node set. The resulting node always contains a single</td>
</tr>
<tr>
<td></td>
<td>node and the root node of the tree.</td>
</tr>
<tr>
<td>system-property</td>
<td>Returns an object representing the value of the system property identified</td>
</tr>
<tr>
<td></td>
<td>by the name.</td>
</tr>
</tbody>
</table>
unparsed-entity-uri Returns declarations of unparsed entities in the document type definition (DTD) of the source document.

The following are node tests, not functions:

- text()
- processing-instruction()
- comment()
- node()

For more information, see Node Tests.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
current Function

Returns a node set that has the current node as its only member.

\[ node-set \ current() \]

Remarks

The function returns a node set that has the current node as its only member. For an outermost expression, an expression not occurring within another expression, the current node is always the same as the context node. Thus,

\[ <xsl:value-of \ select="current()"/> \]

is the same as

\[ <xsl:value-of \ select="."/> \]

However, within square brackets, the current node is usually different from the context node. For example,

\[ <xsl:apply-templates \ select="//glossary/item[@name=current()/@ref]"/> \]

processes all \( \text{item} \) elements that have a \( \text{glossary} \) parent element and a \text{name} attribute with value equal to the value of the current node's \text{ref} attribute. This is different from

\[ <xsl:apply-templates \ select="//glossary/item[@name=./@ref]"/> \]

which means the same as

\[ <xsl:apply-templates \ select="//glossary/item[@name.@ref]"/> \]

and so would process all \( \text{item} \) elements that have a \( \text{glossary} \) parent element and that have a \text{name} attribute and a \text{ref} attribute with the same value.

Example
Note  To test this example, you need to use a script. For more information, see [Initiate XSLT in a Script](#).

**XML File (current.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="current.xsl" ?>
<nodes>
  <node>first</node>
  <node>1</node>
</nodes>
```

**XSLT File (current.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/"
                apply-templates/>
</xsl:stylesheet>
```

**Output**

The following is the formatted output shown in the browser.
The following is the output from the XSLT processor. To obtain this output, right-click on the browser and select the View XSL Output menu item.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<blockquote>
  <blockquote>first</blockquote>
  <blockquote>1</blockquote>
  <blockquote><blockquote>class</blockquote></blockquote>
</blockquote>
```

See Also

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
**document Function**

Provides a way to retrieve other XML resources from within the XSLT style sheet beyond the initial data provided by the input stream.

\[ \text{node-set document(object, node-set?)} \]

**Remarks**

The effects of the \text{document()} function vary, depending on the type and number of arguments supplied.

- If only one argument is provided and that argument is a string, \text{document()} treats the string as a URL and retrieves the document as a set of nodes. See \textbf{Example 1} below for a code illustration.
- If only one argument is provided and that argument is a node set, then each node in that node set is treated as a URL and the function returns the union of all of the documents referenced. See \textbf{Example 2} below for a code illustration.
- If there are two arguments, the first argument can be either a string or a node set while the second argument must be a node set. The second argument, when supplied, serves to indicate the base URL to which the contents of the first argument are relative. See \textbf{Example 3} below for a code illustration.
- If an empty string is passed to the \text{document()} function, the result is the source XML of the XSLT document itself, unless the second argument is given (and is not null). In the latter case, the URL of the document is the base URL of the node contained in the second element.

When an XML document is downloaded from within an ASP page using the \text{document()} function with an absolute URL as its argument, the \text{ServerHTTPRequest} property must be set to \text{true} on the DOM object holding the XSLT style sheet before any transformation is initiated, as shown in the following JScript code fragment:

\[ \text{xsltDom.setProperty("ServerHTTPRequest", true)}; \]
The reason for this is that MXSML uses URLMON.dll as the default to support file download. However, ASP does not support URLMON. Therefore, you need to set this property to choose WinHTTP.dll as the alternative to downloads.

Example 1

In this example, the XSLT style sheet (document.xsl) performs the following steps:

1. Uses the document() function to download two XML documents (hrGroup.xml and myGroup.xml) referenced in the source XML document (document.xml).
2. Extracts the <group> elements from the downloaded documents.
3. Inserts the extracted elements into the resultant XML document.

The effect of the transformation is to resolve the references in document.xml. When a relative path is used to refer the external XML document, the base URL is that of the XSLT style sheet.

When the transformation is initiated from a script in an ASP page (document.asp), the ServerHTTPRequest property is set to true. This is not necessary when the transformation is started in JScript (document.js) and executed within the Windows Scripting Host or an HTML page. Here Windows Script Host is used.

Note To test this example, you need to use a script. For more information, see Initiate XSLT in a Script.

XML File (document.xml)

```xml
<?xml version='1.0'?>
<groups>
  <groupRef href="http://localhost/hr/hrGroup.xml"/>
  <groupRef href="myGroup.xml"/>
</groups>
```

Referenced XML File 1 (hrGroup.xml)

```xml
<?xml version='1.0'?>
<group name="hr">
  <leader>mo</leader>
```
Referenced XML File 2 (myGroup.xml)

<?xml version='1.0'?>
<group name="my">
    <leader>john</leader>
    <member>jane</member>
    <member>jon</member>
    <member>jan</member>
</group>

XSLT File (document.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <groups>
            <xsl:apply-templates select="//groupRef"/>
        </groups>
    </xsl:template>
    <xsl:template match="groupRef">
        <xsl:copy-of select="document(@href)//group"/>
    </xsl:template>
</xsl:stylesheet>

ASP File (document.asp)

<%@ LANGUAGE = JScript %>
<% 
    // Set the source and style sheet locations here.
    var sourceFile = Server.MapPath("document.xml");
    var styleFile = Server.MapPath("document.xsl");

    // Load the XML.
    var source = Server.CreateObject("Msxml2.DOMDocument.5.0");
    source.async = false;
    source.load(sourceFile);

    // Load the XSLT.
    var style = Server.CreateObject("Msxml2.DOMDocument.5.0");
   %>
style.async = false;
style.setProperty("ServerHTTPRequest", true);
style.load(styleFile);

Response.Write(source.transformNode(style));

%%>

**JScript File (document.js)**

```javascript
// Set the source and style sheet locations here.
var sourceFile = "document.xml";
var styleFile = "document.xsl";

// Load the XML.
var source = new ActiveXObject("Msxml2.DOMDocument.5.0");
source.async = false;
source.load(sourceFile);

// Load the XSLT.
var style = new ActiveXObject("Msxml2.DOMDocument.5.0");
style.async = false;
style.load(styleFile);

WScript.Echo(source.transformNode(style));
```

**Try It!**

1. Copy the code samples above, and paste them into files on your local drive.
2. Save the files with the given names and extensions.
3. Create a new virtual directory, aliased as hr, under the default Web site on your local machine. If you use a remote machine as the web server, change `localhost` to the host name of the Web server in the value of the `href` attribute in the `document.xml`. If you use an existing virtual directory, change `hr` in the same `href` attribute value to whatever alias that directory has. Move the `hrGroup.xml` file to the directory that has been made the virtual directory of the Web server.
4. Open the ASP page from an Internet Explorer browser window with the following URL:

   http://localhost/hr/document.asp

5. Run the JScript file using the following Windows Scripting Host command:

   cscript document.js
Output

The following is the processor output, with indentation added for clarity.

```xml
<?xml version="1.0"?>
<groups>
    <group name="hr">
        <leader>mo</leader>
        <member>bo</member>
        <member>ko</member>
        <member>lo</member>
    </group>
    <group name="my">
        <leader>john</leader>
        <member>jane</member>
        <member>jon</member>
        <member>jan</member>
    </group>
</groups>
```

Example 2

This example illustrates the effects of using a node set as the argument for the document function.

XML File (document1.xml)

```xml
<?xml version='1.0' encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="document1.xsl" ?>
<groups>
    <groupRef>hrGroup1.xml</groupRef>
    <groupRef>myGroup1.xml</groupRef>
</groups>
```

XSLT Style Sheet (document1.xsl)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output method="xml"
        version="1.0"
        encoding="UTF-8"
        indent="yes"/>

    <xsl:template match="/"/>
```
<groups>
  <xsl:apply-templates select="/groups/groupRef"/>
</groups>
</xsl:template>

<xsl:template match="groups/groupRef">
  <xsl:copy-of select="document(.)//group"/>
</xsl:template>

</xsl:stylesheet>

Try It!

1. Download msxsl.exe from msdn.microsoft.com, if you have not already done so.
2. From a command prompt, type the following command from the directory where you have saved document1.xml and document1.xsl:

   msxsl document1.xml document1.xsl

Output

The output is similar to the output produced in Example 1 above.

Example 3

This example shows the effect of using two arguments when calling the document function. The second argument, which must be a node set, serves as the base URL for the first argument. When the second argument is absent, the base URL of the first argument is the base URL of the XSLT file.

XSLT Style Sheet (document2.xsl)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="y

<xsl:template match="/">
  <root>
    <xsl:comment>One Argument</xsl:comment>
    <xsl:for-each select="document('b.xml')//a">
      <xsl:copy-of select="."/>
    </xsl:for-each>
  </root>
```
<xsl:comment>Two	Argument	</xsl:comment>
<xsl:for-each	select="document('a.xml',	.)//a">
  <xsl:copy-of	select="."/>
</xsl:for-each>
</root>
</xsl:template>
</xsl:stylesheet>

XML File 1 (b.xml)

<doc>
  <a> one </a>
  <a> two </a>
  <a> three </a>
</doc>

XML File 2 (subDir/a.xml)

<doc>
  <a> I </a>
  <a> II </a>
  <a> III </a>
</doc>

Try It!

1. Save document2.xsl and b.xml to a working directory.
2. Create a subDir directory under the working directory and save a.xml there.
3. From a command prompt type the following command in the working directory:

   msxml.exe subDir\a.xml document2.xsl

Output

<?xml version="1.0" encoding="UTF-8"?>
<root>
  <!-- One Argument -->
  <a> one </a>
  <a> two </a>
  <a> three </a>
  <!-- Two Arguments -->
  <a> I </a>
  <a> II </a>
  <a> III </a>
</root>
See Also

xsl:copy-of Element
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
element-available Function

Returns True only if the expanded-name is the name of an instruction.

boolean element-available(string)

Remarks

The argument must evaluate to a string that is a QName. The QName is expanded into an expanded-name using the namespace declarations in scope for the expression.

This function returns True for all XSLT instruction elements that are supported in this release. The function returns False for all XSLT non-instruction elements, and for those elements that are not supported in this release. Outside of XSLT, the function always returns False.

Example

The following function call returns True (provided the xs1 prefix is mapped to the XSLT namespace).

element-available("xs1:template")

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
format-number Function

Converts numbers into strings. The conversion is the result of formatting the number specified in the first argument (number), using the format specified in the second argument (string), and applying the rules defined in the decimal format named in the third optional argument (string). If the third argument, the decimal format name, is omitted, the default decimal format is used. This function and the <xsl:decimal-format> element are designed to be used together.

\[ \text{string format-number} (\text{number, string, string?}) \]

Remarks

The first argument specifies the \textit{input value}. For example, the number 5351.

The second argument specifies the \textit{format pattern}. For example, the string \texttt{ ###.###,00}.

The third argument is an optional \textit{decimal format name}. This enables any character to be used in the format pattern string; the character is assigned a role in the \texttt{<xsl:decimal-format>} element. A common decimal format name is \texttt{european}, used to reverse the role of comma and period from the default.

In the format pattern, the following characters are supported \\texttt{.}, \\texttt{,}, \\texttt{#}, \\texttt{%}, \\texttt{0}, and the Unicode per-mille character (\#x2030). The \\texttt{#} character is used to describe integers. The \\texttt{.} and \\texttt{,} characters are used to describe the location and type of punctuation. The \\texttt{0} character is used to place a 0 before and after the \\texttt{.} character when the position would otherwise show no numeral, such as 00.3 or 3.00. When truncating digits to the right of the decimal points, the number is rounded. The number to the left of the decimal point is never truncated.

The input value and format pattern string must both be present or an error will be thrown. If the input value cannot be coerced to a number by using the rules in the \texttt{number()} function, the format-number function returns NaN. If the format pattern string is not a valid pattern string, the function returns the input value,
The decimal format name, if present, represents the name of a decimal format as specified in an `<xsl:decimal-format>` element. The decimal format name must be a qualified name. If you specify a decimal format name in the function, there must be an `<xsl:decimal-format>` element in the style sheet with the same name.

The `<xsl:decimal-format>` element can override the default number formatting pattern. The `<xsl:decimal-format>` element follows the formatting specifier patterns of the `java.text.DecimalFormat` class. For examples of using the `format-number` function with `<xsl:decimal-format>`, see `<xsl:decimal-format> Element`.

**Examples**

The following XSLT lines return "24.535,20":

```xml
<xsl:decimal-format name="european" decimal-separator=',' grouping-separator='.'
<xsl:value-of select="format-number(24535.2, '###.###,00', 'european"
```

The following function call returns "5,351":

```java
format-number(5351,"#,###")
```

The following function call returns "5351.00":

```java
format-number(5351, "#.00")
```

The following function call returns "53.5100":

```java
format-number(53.51, "#.0000")
```

The following function call returns "0053.5100":

```java
format-number(53.51, "0000.0000")
```

The following function call returns "0053.51":

```java
format-number(53.51, "0000.####")
```

The following function call returns "53.6":

```java
format-number(53.6, "#.####")
```
format-number(53.56, "0.0")

For a complete example of applying this function to XML data using an XSLT style sheet, see Example of `<xsl:decimal-format>`.

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference | `<xsl:decimal-format>` Element | Initiate XSLT in a Script
function-available Function

Returns True if the function is in the function library.

boolean function-available(string)

Remarks

The argument must evaluate to a string that is a QName. The QName is expanded into an expanded-name using the namespace declarations in scope for the expression. The function-available function returns True if and only if the expanded-name is the name of a function in the function library. If the expanded-name has a non-null namespace Uniform Resource Identifier (URI), it refers to an extension function; otherwise, it refers to a function defined by XML Path Language (XPath) or XSLT.

Example

The following function call returns True.

function-available("format-number")

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference | Initiate XSLT in a Script
generate-id Function

Returns a string that uniquely identifies the node in the node-set argument that is first in document order.

\[
\text{string } \text{generate-id}(\text{node-set})
\]

Remarks

The unique identifier must consist of ASCII alphanumeric characters and must start with an alphabetic character. Thus, the string is syntactically an XML name. There is no guarantee that a generated unique identifier will be distinct from any unique IDs specified in the source document. If the node-set argument is empty, the empty string is returned. If the argument is omitted, it defaults to the context node.

Example

XML File (data.xml)

```xml
<?xml-stylesheet type="text/xsl" href="sample.xsl"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies, a</description>
  </book>
</catalog>
```
XSLT File (sample.xsl)

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <xsl:for-each select="//book">
          <button id="{generate-id(author)}" onclick="alert(this.id)">
            <xsl:value-of select="author"/>
          </button>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Output

This is the formatted output:

| Gambardella, Matthew | Ralls, Kim |

When you click the left button in Internet Explorer, an alert box displays "IDAHAGJD".

When you click the right button in Internet Explorer, an alert box displays "IDAPAGJD".

Notice that the ID value is generate at run-time. Therefore, the specific values may vary in different invocation of the transformation.

This is the processor output:

```html
<html>
<body>
<button id="IDAHAHGD" onclick="alert(this.id)">Gambardella, Matthew</button>
<button id="IDAPAGJD" onclick="alert(this.id)">Ralls, Kim</button></body>
</html>
```

See Also

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#) | [Initiate XSLT in a Script](#)
key Function

Retrieves elements previously marked with an `<xsl:key>` statement.

```
node-set key(name, value)
```

Remarks

The `key()` function retrieves a node set (zero or more nodes) that have the same key name and key value specified in the `<xsl:key>` statement. When the XSLT style sheet is first processed, the keys are stored internally to simplify access. Keys can simplify accessing nodes in the XML document, but may not be faster than using XPath to retrieve the same nodes.

Example

The `key()` function works in conjunction with the `<xsl:key>` element. For an example, see `<xsl:key>` Element.

See Also

Initiate XSLT in a Script
node-set Function

Enables you to convert a tree into a node set. The resulting node always contains a single node and the root node of the tree.

\[ \text{msxsl:node-set(string)} \]

**Remarks**

With earlier versions of Microsoft® XML Core Services (MSXML), it was possible to use expressions such as \(<xsl:for-each select="$var/el">\), where \(var\) is an XSLT variable bound to a result tree. However, this approach does not work with MSXML versions 3.0 and later. To achieve the same results in these more recent version of MSXML, use the node-set function, as shown in the following code sample.

\(<xsl:for-each select="msxsl:node-set($var)/el">\)

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#) | [Initiate XSLT in a Script](#)
system-property Function

Returns an object representing the value of the system property identified by the name.


object system-property(string)

Remarks

The argument must evaluate to a string that is a valid QName. The QName is expanded into a name using the namespace declarations in scope for the expression. The system-property function returns an object representing the value of the system property identified by the name. If there is no such system property, the empty string is returned.

Because these properties are strings representing QNames, a prefix must correspond to a namespace declaration in scope for "http://www.w3.org/1999/XSL/Transform" for the xsl:version, xsl:vendor, and xsl:vendor-url properties. A prefix corresponding to a namespace declaration "urn:schemas-microsoft-com:xslt" must be in scope for the msxsl:version property.

<table>
<thead>
<tr>
<th>System property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsl:version</td>
<td>A number giving the version of XSLT implemented by the processor; for XSLT processors implementing the version of XSLT specified by this document, the number 1.</td>
</tr>
<tr>
<td>xsl:vendor</td>
<td>Returns the string &quot;Microsoft&quot;.</td>
</tr>
<tr>
<td>xsl:vendor-url</td>
<td>A string containing the URL &quot;<a href="http://www.microsoft.com">http://www.microsoft.com</a>&quot;.</td>
</tr>
<tr>
<td>msxsl:version</td>
<td>A number giving the version of MSXML.</td>
</tr>
</tbody>
</table>

Example

The following function call returns 1.

system-property('xsl:version')
See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference | Initiate XSLT in a Script
unparsed-entity-uri Function

Returns the URI of the unparsed entity.

\[ \text{string unparsed-entity-uri(string name)} \]

**Parameter**

**Name**

The name of the entity desired. The entity must be defined in the same document as the context node.

**Return Values**

Returns a URI string of the unparsed entity if the entity is defined. Otherwise, it returns an empty string.

**Example**

This example demonstrates the unparsed-entity-uri() function to retrieve the path of the image file, somepic.jpg, which has been associated with the pic entity.

**XML File (data.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="sample.xsl"?>
<!DOCTYPE catalog [
  <!ELEMENT catalog ANY>
  <!ELEMENT book ANY>
  <!ELEMENT title ANY>
  <!NOTATION JPEG SYSTEM "urn:foo">
  <!ENTITY pic SYSTEM "somepic.jpg" NDATA JPEG>
]>
<catalog>
  <book>
    <title>XML Developer's Guide</title>
  </book>
</catalog>
```
XSLT File (sample.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <body>
                <h3>unparsed-entity-uri()</h3>
                <ul>
                    <li>
                        <b>unparsed-entity-uri('pic')</b>
                        =
                        <xsl:value-of select="unparsed-entity-uri('pic')"/>
                    </li>
                </ul>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>

Output

This is the formatted output:

unparsed-entity-uri()

- unparsed-entity-uri('pic') = file:///C:/path/somepic.jpg

This is the processor output:

<html>
<body>
<h3>unparsed-entity-uri()</h3>
<ul>
<li><b>unparsed-entity-uri('pic')</b> =
    file:///C:/path/somepic.jpg</li>
</ul>
</body>
</html>
See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference | Initiate XSLT in a Script
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
XSLT Syntax

This section covers the following:

- Qualified Names
- NaN Values
- Expressions
- Patterns
Qualified Names

A qualified name (QName) is composed of a prefix and a local part. The prefix provides the namespace prefix part of the qualified name, and must be associated with a namespace Uniform Resource Identifier (URI).

**Note** The prefix functions only as a placeholder for a namespace name. Applications should use the namespace name, not the prefix, in constructing names whose scope extends beyond the containing document.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
**NaN Values**

The term NaN, which stands for "not a number," means a symbolic entity that represents a value not otherwise available in floating-point format.

There are two kinds of NaNs.

**Quiet**
- Represent unknown or unitialized values.

**Signaling**
- Represent symbolic values and values that are too big or too precise for the format. Signaling NaNs raise an invalid operation exception whenever an operation is attempted on them.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Refer
Expressions

Expressions are used in XSL Transformations (XSLT) for the following purposes.

- Selecting nodes for processing.
- Specifying conditions for different ways of processing a node.
- Generating text to be inserted in the result tree.

Expressions occur as the value of certain attributes on XSLT-defined elements and within curly braces in attribute value templates.

Expression evaluation occurs with respect to a context. In XSLT, an outermost expression (that is, an expression that is not part of another expression) gets its context as follows.

- The context node comes from the current node.
- The context position comes from the position of the current node in the current node list; the first position is 1.
- The context size comes from the size of the current node list.
- The variable bindings are the bindings in scope on the element that has the attribute in which the expression occurs.
- The set of namespace declarations are those in scope on the element that has the attribute in which the expression occurs.
- The function library consists of those found at XPath Functions and any functions defined by a binding mechanism such as addObject Method or the msxsl:script Element. Calling a function with no corresponding binding is an error.

See Also

Patterns
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XSLT Reference
Patterns

A *pattern* is a type of expression that returns a set of nodes based on a set of conditions for a node. Nodes that satisfy the conditions match the pattern. The syntax for patterns is a subset of the syntax for expressions. In particular, location paths that meet certain restrictions can be used as patterns. An expression that is also a pattern always evaluates to an object of type node-set. A node matches a pattern if the node is a member of the result of evaluating the pattern as an expression with respect to some possible context; the possible contexts are those whose context node is the node being matched or one of its ancestors.

The following table lists examples of patterns.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Para</td>
<td>any para element</td>
</tr>
<tr>
<td>*</td>
<td>any element</td>
</tr>
<tr>
<td>chapter</td>
<td>appendix</td>
</tr>
<tr>
<td>olist/item</td>
<td>any item element with an olist parent</td>
</tr>
<tr>
<td>appendix//para</td>
<td>any para element with an appendix ancestor element</td>
</tr>
<tr>
<td>/</td>
<td>the root node</td>
</tr>
<tr>
<td>text()</td>
<td>any text node</td>
</tr>
<tr>
<td>processing-instruction()</td>
<td>any processing instruction</td>
</tr>
<tr>
<td>node()</td>
<td>any node other than an attribute node</td>
</tr>
<tr>
<td>id(&quot;W11&quot;)</td>
<td>the element with unique ID w11</td>
</tr>
<tr>
<td>para[1]</td>
<td>any para element that is the first para child element of its parent</td>
</tr>
<tr>
<td>*[position()=1 and self::para]</td>
<td>any para element that is the first child element of its parent</td>
</tr>
<tr>
<td>para[last()=1]</td>
<td>any para element that is the only para child element of its parent</td>
</tr>
</tbody>
</table>
See Also

Location Paths
This guide introduces the Microsoft XML implementation of the XML Path Language (XPath). XPath provides a language for addressing parts of an XML document. It is specified as an open standard in the W3C XPath specification. XPath is used heavily in combination with XSLT for selecting nodes, though it can be used in some instances with XML DOM programming as well.

The XPath Developer's Guide, along with the reference material that accompanies it, describes the Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office implementation of the XPath standard.

The following sections provide more information about creating applications and style sheets that use XPath.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter Kit</td>
<td>Explains the benefits of the XPath, and offers ideas for building XPath applications.</td>
</tr>
<tr>
<td>Concepts</td>
<td>Introduces the fundamental concepts that relate to the XPath.</td>
</tr>
<tr>
<td>How Do I?</td>
<td>Helps you build XSLT style sheets and other XML applications that use XPath. This section provides tutorials for using XPath to select nodes and navigate XML document data.</td>
</tr>
</tbody>
</table>

See Also

XPath Reference
Starter Kit

This starter kit is designed to provide tools and information for working with XML Path Language (XPath).

- What is XPath?
- Introduction to XPath Syntax
- XPath Tutorial Application
What is XPath?

The XML Path Language (XPath) provides a language for addressing parts of an XML document. It is fully described in the W3C XML Path Language (XPath) specification.

XPath was designed to:

- Provide an efficient, compact, logical syntax for referring to path or node tree locations within XML documents.
- Remain an effective language tool for referencing the contents of XML documents separate of any markup used within them.
- Offer a standard library of functions for working with strings, numbers, and Boolean expressions when writing programs that process XML documents.
- Work closely with XSL Transformations (XSLT) and other languages or programming interfaces designed for working with XML.

See Also

XPath Tutorial Application | Concepts | How Do I

Other Resources

W3C XML Path Language (XPath) Recommendation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Introduction to XPath Syntax

XPath enables you to locate any one or more nodes within an XML document, often by using multiple alternate routes. In essence, XPath provides the syntax for performing basic queries upon your XML document data. It works by utilizing the ability to work with XML documents as hierarchically structured data sets.

All XML documents can be represented as a hierarchy or tree of nodes. This aspect of XML shares a similarity to how paths are encoded in file system URLs, which are used in Windows Explorer to produce tree views of files and folders on your computer. The following table compares some of the analogous features of both XPath and file system URLs, both of which are used to pass information.

<table>
<thead>
<tr>
<th>File System URLs</th>
<th>XPath Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy comprised of directories and files in a file system.</td>
<td>Hierarchy comprised of elements and other nodes in an XML document.</td>
</tr>
<tr>
<td>Files at each level have unique names.</td>
<td>Element names at each level might not be unique.</td>
</tr>
<tr>
<td>URLs always identify a single file.</td>
<td>XPath patterns identify a set of all the matching elements that match the pattern, which can be either one or many nodes.</td>
</tr>
<tr>
<td>Evaluated relative to a particular directory, called the &quot;current directory.&quot;</td>
<td>Evaluated relative to a particular node called the &quot;context&quot; for the query.</td>
</tr>
</tbody>
</table>

The following XML sample document shows a simple hierarchy, which can be used to demonstrate some of the facilities of XPath.

```xml
<?xml version='1.0'?>
<authors>
  <author period="modern">
    <name>Eva Corets</name>
    <nationality>British</nationality>
  </author>
  <author>
    <name>Cynthia Randall</name>
    <nationality>Canadian</nationality>
  </author>
</authors>
```
A basic XPath pattern describes a path through the XML hierarchy with a slash-separated list of child element names. For example, starting from the document root of the previous sample, the following pattern traverses down through the hierarchy to the <name> elements.

authors/author/name

The XPath pattern identifies all elements that match the path. In addition to describing an exact path down a known hierarchy, XPath can include wildcards for describing unknown elements and selecting a set of nodes. This provides the basics of an XML query facility. For example, an element of any name can be represented by the "*" wildcard as shown in the following pattern.

authors/*/name

The preceding sample identifies all the <name> elements that are grandchildren (children of any child element for the <authors> element), but without requiring them to be children of an <author> element. Here is another example that can be used to find both <name> and <nationality> elements in our sample document.

authors/author/*

Using additional patterns within square brackets can specify branches on the path. For example, the following query describes a branch on the <author> element, indicating that only the <author> elements with <nationality> children should be considered as a pattern match.

authors/author[nationality]/name

This becomes even more useful for the sample data when comparisons are added. The following query returns the names of Canadian authors.

Note comparisons can be used only within brackets.

authors/author[nationality='Canadian']/name
Attributes are indicated in a query by preceding the name of the attribute with "@". The attribute can be tested as a branch off the main path, or the query can identify attribute nodes. The following examples return authors from the modern period, and just the two period attributes, respectively.

```
authors/author[@period="modern"]
authors/author/@period
```

The above XPath expressions are also presented in the XML Tutorial Application. To practice using XPath, build and run the interactive tutorial to see which XML nodes are selected by various XPath patterns and expressions.

For a complete list of XPath features supported by MSXML, see XPath Syntax.

**Other Resources**

[XML Path Language (XPath) Version 1.0](https://msdn.microsoft.com/en-us/library/ms537333(v=vs.85).aspx)
 XPath Tutorial Application

To help you get started working with XPath, this section shows you how to quickly build an interactive tutorial application that will visually highlights the results of various XPath patterns as you selectively use them to query the data in a brief XML document file.

To run the XPath tutorial, you will need to make five files:

- **Sample XML File (authors.xml)** — A short XML sample data file showing book author information.
- **HTML File (author-patterns.htm)** — The HTML application form used to run the application.
- **XSLT File (hilite-xml.xsl)** — The primary style sheet used by the HTML file to highlight the XPath expression you specify.
- **XSLT File (makeID.xsl)** — Another secondary style sheet called by hilite-xml.xsl and used to generate IDs.
- **Error Reporting Script (reportErrors.js)** — A JScript file used as a helper file for handling and reporting errors when running and debugging the sample application.

**To create and run the XPath tutorial application**

1. Open Notepad.
2. Create the five files: authors.xml, author-patterns.htm, hilite-xml.xsl, makeID.xsl, and reportErrors.js.

   To create the files, copy and paste each of the following samples separately to the Notepad window. Save them all in the same folder, using the names provided.

3. Open the author-patterns.htm file.
4. Try experimenting with various XPath patterns and observing the result in the highlighted document.

   To update the highlighted selection, you can either click an XPath pattern in the list of linked patterns on the left. Alternately, you can type a different
pattern in the **XPath Expression** text box, and then click **Select Nodes** to try out other XPath patterns.
Sample XML File (authors.xml)

The following sample code creates the short XML sample data file containing book author information used by the XPath tutorial application.

```xml
<?xml version='1.0'?>
<authors>
  <author>
    <name>Mike Galos</name>
    <nationality>French</nationality>
  </author>
  <author period="modern">
    <name>Eva Corets</name>
    <nationality>British</nationality>
  </author>
  <author>
    <name>Cynthia Randall</name>
    <nationality>Canadian</nationality>
  </author>
  <author>
    <name>Stefan Knorr</name>
    <nationality>Canadian</nationality>
  </author>
  <author period="modern">
    <name>Paula Thurman</name>
    <nationality>British</nationality>
  </author>
</authors>
```
The following sample code creates the HTML application form used to run the XPath tutorial application.

```html
<html>
<head>
    <title>XPath Tutorial Application</title>
    <style>
        body {font-family: Arial}
        .heading {font-family: "Arial Black"}
        .sample {cursor: hand; font: 9pt Courier; text-decoration: underline}
        .note {font-size: x-small; font-style: italic}
    </style>
</head>
<script src="reportErrors.js"></script>

var makeID, source, stylesheet;

function doQuery(theQuery)
{
    xmlSrc.innerHTML = result;
    var q = source.selectNodes(theQuery);

    for (var e = q.nextNode(); e != null; e = q.nextNode())
    {
        try
        {
            eID = e.transformNode(makeID);
            document.all.item(eID).style.backgroundColor = "yellow";
        }
        catch (exception)
        {
            result = reportRuntimeError(exception);
            alert(result);
        }
    }
}

function loadQuery()
{
```javascript
var q = window.event.srcElement.innerText;
qryIn.value = q;
doQuery(q);
}
</SCRIPT>

<SCRIPT FOR="window" EVENT="onload">
makeID = new ActiveXObject("Msxml2.DOMDocument.5.0");
makeID.async = false;
makeID.load("makeID.xsl");
if (makeID.parseError.errorCode != 0)
{
    result = reportParseError(makeID.parseError);
    alert(result);
}

source = new ActiveXObject("Msxml2.DOMDocument.5.0");
source.async=false;
source.load("authors.xml");
source.setProperty("SelectionLanguage", "XPath");
if (source.parseError.errorCode != 0)
{
    result = reportParseError(source.parseError);
    alert(result);
}

stylesheet = new ActiveXObject("Msxml2.DOMDocument.5.0");
stylesheet.async = false;
stylesheet.load("hilite-xml.xsl");
if (source.parseError.errorCode != 0)
{
    result = reportParseError(source.parseError);
    alert(result);
}

try
{
    result = source.transformNode(stylesheet);
}
catch (e)
{
    result = reportRuntimeError(e);
}
xmlSrc.innerHTML = result;
</SCRIPT>

<BODY>
<DIV class=note>Demonstration of querying against an XML document usi
When this page is opened in the browser, it enables you to type or click an XPath expression and see the result automatically highlighted in the displayed copy of the XML source document, a short XML sample data file containing book author information.
XSLT File (hilite-xml.xsl)

The following sample code creates the primary XSLT style sheet used to run the XPath tutorial application.

```xml
<?xml version="1.0"?>
<!-- Generic stylesheet for viewing XML -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt">

  <xsl:template match="/"
    <DIV STYLE="font-family:Tahoma; font-size:11pt; margin-bottom:2em">
    <xsl:apply-templates/>
  </DIV>
  </xsl:template>

  <xsl:template match="*"
    <DIV STYLE="margin-left:1em; color:red">
    <xsl:attribute name="id">
      <xsl:value-of select="generate-id()"/>
    </xsl:attribute>
    &lt;<xsl:value-of select="name()"/>&gt;<xsl:apply-templates select="@*"/>
  &lt;/DIV>
  </xsl:template>

  <xsl:template match="*[node()]">
    <DIV STYLE="margin-left:1em">
    <SPAN STYLE="color:olive">
      <xsl:attribute name="id">
        <xsl:value-of select="generate-id()"/>
      </xsl:attribute>&lt;<xsl:value-of select="name()"/>&gt;<xsl:apply-templates select="node()"/>
    </SPAN>&lt;/<xsl:value-of select="name()"/>
  </DIV>
  </xsl:template>

  <xsl:template match="@*" xml:space="preserve">
    <SPAN STYLE="color:navy"><xsl:attribute name="ID"><xsl:value-of
      <xsl:value-of select="name()"/>="<SPAN STYLE="color:black"><x
    </SPAN>
  </xsl:template>

  <xsl:template match="processing-instruction()">
    <DIV STYLE="margin-left:1em; color:maroon">
```

XSLT File (hilite-xml.xsl)

The following sample code creates the primary XSLT style sheet used to run the XPath tutorial application.

```xml
<?xml version="1.0"?>
<!-- Generic stylesheet for viewing XML -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt">

  <xsl:template match="/"
    <DIV STYLE="font-family:Tahoma; font-size:11pt; margin-bottom:2em">
    <xsl:apply-templates/>
  </DIV>
  </xsl:template>

  <xsl:template match="*"
    <DIV STYLE="margin-left:1em; color:red">
    <xsl:attribute name="id">
      <xsl:value-of select="generate-id()"/>
    </xsl:attribute>&lt;<xsl:value-of select="name()"/>&gt;<xsl:apply-templates select="@*"/>
  &lt;/DIV>
  </xsl:template>

  <xsl:template match="*[node()]">
    <DIV STYLE="margin-left:1em">
    <SPAN STYLE="color:olive">
      <xsl:attribute name="id">
        <xsl:value-of select="generate-id()"/>
      </xsl:attribute>&lt;<xsl:value-of select="name()"/>&gt;<xsl:apply-templates select="node()"/>
    </SPAN>&lt;/<xsl:value-of select="name()"/>
  </DIV>
  </xsl:template>

  <xsl:template match="@*" xml:space="preserve">
    <SPAN STYLE="color:navy"><xsl:attribute name="ID"><xsl:value-of
      <xsl:value-of select="name()"/>="<SPAN STYLE="color:black"><x
    </SPAN>
  </xsl:template>

  <xsl:template match="processing-instruction()">
    <DIV STYLE="margin-left:1em; color:maroon">
```
The following sample code creates the secondary XSLT style sheet used to generate IDs, which are called and used by the primary XSLT file (hilite-xml.xsl).

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">
  <xsl:output omit-xml-declaration="yes"/>

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:value-of select="generate-id()"/>
  </xsl:template>

  <xsl:template match="*[node()]">
    <xsl:value-of select="generate-id()"/>
  </xsl:template>

  <xsl:template match="@*">
    <xsl:value-of select="generate-id()"/>
  </xsl:template>

  <xsl:template match="processing-instruction()">
    <xsl:value-of select="generate-id()"/>
  </xsl:template>

  <xsl:template match="node()[nodeTypeString=cdatasection]">
    <xsl:value-of select="generate-id()"/>
  </xsl:template>

  <xsl:template match="text()">
    <xsl:value-of select="generate-id()"/>
  </xsl:template>

</xsl:stylesheet>
```
Error Reporting Script (reportErrors.js)

The following sample code creates a helper file for debugging purposes when running the XPath tutorial application.

// JScript File (reportErrors.js)

// Parse error formatting function
function reportParseError(error)
{
    var s = "";
    for (var i=1; i<error.linepos; i++) {
        s += " ";
    }
    r = "<font face=Verdana size=2><font size=4>XML Error loading '" +
        error.url + "'</font><P><B>" + error.reason +
        "</B></P></font>
    if (error.line > 0)
        r += "<font size=3><XMP>" +
        "at line " + error.line + ", character " + error.linepos +
        "\n" + error.srcText +
        "\n" + s + "^" +
        "</XMP></font>";
    return r;
}

// Runtime error formatting function.
function reportRuntimeError(exception)
{
    return "<font face=Verdana size=2><font size=4>XSL Runtime Error</font><P><B>" + exception.description + "</B></P></font>
        "<P>" + exception + "</P>";
}
Concepts

The XML Path Language (XPath) provides a language for addressing parts of an XML document. This section introduces some of the fundamental concepts pertinent to XPath and its use with XSLT and other forms of XML-based programming.

- Introduction to the XPath Tree Model
- Basic Syntax of XPath Expressions
- Understanding XPath Usage in XSLT
- Using XPath with the DOM
- Context and Navigation in XPath
- Navigating XML Data Using XPath Axes
- Filtering XML Data Using XPath Predicates
- Using Functions in XPath Expressions
- Useful Patterns for XPath Expressions
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Introduction to the XPath Tree Model

XPath treats an XML document as a tree of interrelated branches and nodes. A node can be any of the following types of nodes used in any XML document:

- Elements
- Attributes
- Processing Instructions
- Comments
- Textual Content
- Namespace

Additionally, the document itself is also considered a type of node. The XPath tree model is based not on the nodes alone, but on their relationship to one another. For example, how elements relate to one another, attributes to elements, and so on.

The concept of a node is important. XPath—and by extension, XSLT—does not provide direct access to tags, attributes, and other markup. It provides access to the logical nodes established by that markup.

XPath's view of this sample document is explored in the following topics, which are contained in this section:

- Overview of XPath Axes and Node Relationships
- Sample Tree for the Document Root
- Sample Document Root Element Sub-Tree

The topics in this section use this XML file: Sample XML File for XPath Tree Model
Overview of XPath Axes and Node Relationships

In addition to knowing the seven types of nodes, you also need to understand the possible kinds of relationships between and among nodes in a document. Each relationship corresponds to what XPath calls an axis. For more information, see Use XPath Axes to Navigate through XML Data.

The terms used for the axes assume that the document tree represents a set of family relationships. The axes that can be expressed with XPath are:

- Parent
- Child
- Ancestor
- Descendant
- Ancestor-or-self
- Descendant-or-self
- Preceding
- Following
- Preceding-sibling
- Following-sibling
- Self

Two additional relationships apply only to element nodes:

- Attribute
- Namespace
Parent

The parent node of an element, attribute, processing instruction, comment, or text node is the element immediately above it in the tree—the element which contains it. For the root element, and any comments or processing instructions which precede or follow it, the parent is the root node. The root node itself has no parent. The parent of a namespace node is the element which declares it. In the document diagrammed above, for instance:

- The parent of the `<books>` element is the root node.
- The parent of each of the `catnum` attributes is the corresponding `<book>` element.
- The parent of the comment, "Are we sure this guy's name is spelled right???", is the first `<book>` element.
- The parent of both the `<?xsl-stylesheet?>` processing instruction and the comment at the end of the document is the root node.
Child

A node's child is any node immediately below it in the hierarchy of a document's nodes, with some exceptions. The exceptions are that neither attribute nor namespace nodes are considered children of their respective elements. Therefore, if an XPath expression locates all children of a given element, it locates only elements, text nodes, processing instructions, and comments immediately subordinate to that element.

Processing instructions and comments in the document prolog or following the root element are considered children of the root node, as is the root element.

In the document diagrammed above, for instance:

- The children of the root node are the `<?xsl-stylesheet?>` processing instruction, the root `<books>` element, and the comment which follows the root element's end tag.
- The children of the `<books>` element node are the two `<book>` elements. Note that the `catdate` attribute is not considered a child of the `<books>` element node, despite the fact that `<books>` is considered `catdate`'s parent.
- The children of the first `<book>` element are the first `<title>` element, the comment "Are we sure this guy's name is spelled right??" and the first `<author>` element.
- The second `<title>` element has only one child, the text node "For Love of a Toothpick."
Ancestor

An ancestor of any node in the tree is any node at a higher level in the tree than the node in question, including its parent. The root node has no ancestors; on the other hand, the root node is an ancestor of all other nodes in the tree.

For example, in the previous document diagrammed above:

- The catdate attribute's ancestors are the <books> element and the root node.
- The <?xml-stylesheet?> processing instruction, the root <books> element, and the comment following the <books> element each has a single ancestor, the root node.
- Ancestors of the text node "Frey, Jörg" are the second <author> element, the second <book> element, the root <books> element, and the root node.
Descendant

A descendant node is any node (including children) which is subordinate to a given node in the document tree. Everything in the document tree is a descendant of the root node, except the root node itself and any attribute or namespace nodes.

For example, in the document diagrammed earlier in this topic:

- The catdate and catnum attributes have no descendants. (Attributes have no children.)
- Descendants of the first <book> element are the first <title> element and its text node, "Jambing on the Trixies"; the comment, "Are we sure this guy's name is spelled right???"; and the first <author> element and its text node, "Randall, Tristan." Since an attribute is not considered a child of its defining element, the catnum="id2345" attribute is not a descendant of this <book> element (or indeed of anything else in the document).
Ancestor-or-self

Ancestor-or-self nodes of a given node in the tree include all ancestors of that node and the node itself. Therefore, although the root node has no ancestors, an XPath expression which locates the ancestor-or-self nodes of the root node will locate the root node itself.

Some examples from the document diagrammed in this topic are:

- The ancestor-or-self nodes of the concluding comment include the root node and the comment itself.
- The catdate attribute's ancestor-or-self nodes are the `<books>` element which defines it, the root node, and the catdate attribute itself.
Descendant-or-self

If a given node in the tree has any descendants, an XPath expression locating its descendant-or-self nodes will locate all those descendants, and the node itself. If it has no descendants, such an expression will locate only the node itself. Thus, given the document diagrammed in this topic:

- The second <title> element's descendant-or-self nodes are its text node, "For Love of a Toothpick," and that <title> element itself.
- The document's concluding comment has no descendants; therefore it has only a single descendant-or-self node, which is the comment itself.
Preceding

A preceding node, relative to a given node in the document tree, is any node which appears in the document before that node, except ancestors, attribute nodes, or namespace nodes. One way to think of this is that preceding nodes are those whose content occurs in its entirety before the start of the node in question. Since the root node contains all other nodes in the document, it will never be located among a given node's preceding nodes; likewise, the root node itself has no preceding nodes. Given the document diagrammed earlier in this topic, for example:

- The root `<books>` element has only one preceding node, which is the `<?xml-stylesheet?>` processing instruction.
- Everything in the document (except the attributes and the document root node) is a preceding node relative to the concluding comment.
- The first `<author>` element's preceding nodes are the comment, "Are we sure this guy's name is spelled right??", the first `<title>` element and its text node ("Jambing on the Trixles"), and the `<?xml-stylesheet?>` processing instruction. Since they are ancestors of the first `<author>` element, the first `<book>` element, the root `<books>` element, and the root node all terminate after the first `<author>` element closes, therefore they are not among its preceding nodes.
Following

Following nodes are the reverse of preceding ones: They include any nodes (except descendant, attribute, and namespace nodes) which come after a given node. Since the root node contains all other nodes in the document, it is a following node of none of them, and itself has no following nodes.

Based on the document diagrammed in this topic, we can say:

- The root <books> element has only one following node, which is the document's concluding comment.
- The concluding comment has no following nodes.
- The first <author> element's following nodes are the second <book> element (and all of the second <book> element's descendants) and the concluding comment. Because the text node "Randall, Tristan" is a descendant (specifically, a child) of the first <author> element, it is not considered a following node for that element.
Preceding-sibling

Sibling relationships in a family identify children of the same parent, relative to one another. Therefore preceding-sibling nodes are a subset of all siblings, including only those which appear in the document before the node in question. Since attribute and namespace nodes can never be child nodes, they can never be found among a given nodes' preceding-siblings.

Note XPath does not define a simple sibling relationship, only preceding-sibling and following-sibling.

For instance, referring to the document described earlier in this topic:

- Each of the `<author>` elements has one preceding-sibling node, which is the respective `<title>` element. However, the first `<author>` element also has another preceding-sibling, which is the comment, "Are we sure this guy's name is spelled right??"
- The root `<books>` element has one preceding-sibling, the `<?xml-stylesheet?>` processing instruction.
- The concluding comment has two preceding-siblings: the root `<books>` element and the `<?xml-stylesheet?>` processing instruction.
**Following-sibling**

Among a given node's following-siblings, you will find all those nodes (except attribute and namespace nodes, which can never be child nodes) which share that node's parent and appear after the node in question in the document.

**Note** XPath does not define a simple sibling relationship, only preceding-sibling and following-sibling.

Examples from the document diagrammed earlier in this topic include:

- Neither of the `<author>` elements has any following-sibling nodes.
- The root `<books>` element has one following-sibling, which is the concluding comment.
- The concluding comment has no following-siblings.
**Self**

In some cases, you need an XPath expression which locates only the node you're already dealing with. For such cases, XPath provides the "self relationship." Every node in any given document has, of course, one and only one self node.
Attribute

Since attributes can never be found among the parent, child, ancestor, descendant, preceding, following, preceding-sibling, or following-sibling nodes of a given node, how do you locate them with XPath? The answer lies in the special attribute relationship. An element's attribute nodes are any attributes which it declares, and only elements can have attribute nodes.

**Note**  A processing instruction's pseudo-attributes cannot be located this way.

Using the sample document diagrammed earlier in this topic, for instance:

- Each of the `<book>` elements has a single attribute node, the respective `catnum` attribute.
- The root `<books>` element likewise has one attribute node, the `catdate` attribute.
- None of the other elements in this sample document have any attribute nodes.
Namespace

An element has a namespace node for every namespace which is in scope for it. Because namespaces are implicitly inherited by all elements descended from an element which declares a namespace, the following hold true:

- An element may have a namespace node even if the element does not itself declare a namespace.
- An element may have multiple namespace nodes, one corresponding to each namespace that it declares and one corresponding to each namespace declared by an ancestor element.

See Also

Sample XML File for XPath Tree Model
Sample XML File for XPath Tree Model

Topics in this section use this file.

XML File (booksxpath.xml)

```xml
<?xml version="1.0" ?>
<!DOCTYPE books [ 
<!ENTITY oumlaut "&#38;#246;">
]>  
<?xml-stylesheet type="text/xsl" href="catalog.xsl" ?>
<books catdate="2000-12-31">
  <book catnum="id2345">
    <title>Jambing on the Trixles</title>
    <!-- Are we sure this guy's name is spelled right?? -->
    <author>Randall, Tristan</author>
  </book>
  <book catnum="id7823">
    <title>For Love of a Toothpick</title>
    <author>Frey, J&oumlaut;rg</author>
  </book>
</books>
<!-- Reviewed by Catalog Department 2000-12-30 09:11:16 -->
```
Sample Tree for the Document Root

At the most obvious structural level, the tree which XPath can wander through the Sample XML File for XPath Tree Model consists of the document prolog—the XML declaration, the document-type declaration, and the <?xml-stylesheet?> processing instruction—the root element <books>, and the comment which follows the root element's end tag.

Schematically, this tree might be represented like this:

```
  document root
  |<--<?xml ... ?>
  |   \______________
  |   |<--!DOCTYPE ...
  |   |<--<?xml-stylesheet ...
  |   |       \______________
  |   |       |<books>
  |   |<-- Reviewed by... -->
```

Each of these containers is referred to in XPath terms as a node, and a collection of nodes as a node-set. In the above figure, there are four nodes accessible to most XSLT processors, including the MSXML engine:

- The document root node
- The <?xml-stylesheet?> processing-instruction node
- The <books> element node
- The comment node following the <books> element

**Note** The XML declaration and the document-type declaration are not typically accessible. This is because by the time an XSLT processor gets access to the document tree, the document has been fully parsed; information in the XML and document-type declarations is assumed to be useful to the parser but not passed to a downstream application.
See Also

Sample XML File for XPath Tree Model
Sample Document Root Element Sub-Tree

If the diagrams in Sample Tree for the Document Root were all of the document's content that XPath exposed, quite a bit of content would still be inaccessible for linking or transforming purposes. Fortunately, there is also a sub-tree available within the <books> root element highlighted above.

Fully expanded, the <book> element's sub-tree looks like this:
Note  For simplicity, the above diagram disregards the newline characters following each element’s end tag. For more information, see Controlling White Space with XSLT.

Like the full document tree shown in the preceding section, this sub-tree consists of a number of nested containers. While there is no Processing Instruction node type in this sub-tree, there could be; in addition, it shows the following node
types:

- element nodes (outlined in black), such as `<books>` itself, `<book>`, `<title>`, and `<author>`
- attribute nodes (red), `catdate` and `catnum`
- text nodes (blue), such as "Randall, Tristan" and "For Love of a Toothpick"
- comment nodes (green), of which there is only one here

**Note** The text node showing the name of the author of "For Love of a Toothpick." In the tree it appears as "Frey, Jörg" with the entity substitution made. As with the XML and document-type declarations, entity declarations are invisible to XPath and XSLT processors; all that these processors see is the complete document as parsed, including the expanded form of entities.

A seventh node type, namespace nodes, may or may not be present in a document. A namespace node consists of a prefix and a URI to which the prefix is bound, and exists implicitly for any element or attribute node within the scope of the element which declares the namespace prefix. The `xml:` prefix is by default in effect for all XML documents.

**See Also**

[Sample XML File for XPath Tree Model](#)
Basic Syntax of XPath Expressions

This section contains the following topics.

- Overview of XPath Syntax
- Constructing an XPath Location Step
- Constructing the Node-Test Portion of a Location Step
- Specifying the Axis Portion of a Location Step
- Constructing the Predicate Portion of a Location Step
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Overview of XPath Syntax

An XPath expression consists of a series of one or more location steps, separated by "/" delimiters; together, these steps are referred to as the location path. So a location path consisting of two location steps looks like:

locationstep/locationstep

Each location step, from left to right, relocates you up or down in the document's tree of nodes relative to the location step(s) preceding it, if any.

A location step can be absolute or relative. Absolute location steps can never appear in the location path in any but the leftmost position. Relative location steps can appear anywhere in the location path.

Location paths can be compound—that is, you can construct a location path which will select any nodes matching any of a group of constituent location paths. To do so, separate the location paths with a pipe character, |. For example:

locationpath1 | locationpath2

says to locate all nodes in the target document on either locationpath1 or locationpath2.

Some sample absolute location paths and their interpretations are provided in the following table.

<table>
<thead>
<tr>
<th>Location path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Locates the document root</td>
</tr>
<tr>
<td>/books</td>
<td>Locates all &lt;books&gt; children of the document root</td>
</tr>
<tr>
<td>/descendant::title</td>
<td>Locates all &lt;title&gt; elements in the document (i.e., descended from the document root)</td>
</tr>
<tr>
<td>/comment()</td>
<td>/text()</td>
</tr>
<tr>
<td>/processing-instruction(&quot;xml-</td>
<td>Locates any processing instruction</td>
</tr>
</tbody>
</table>
stylesheet"

whose target is "xml-stylesheet".

Absolute Location Steps

An absolute location step in an XPath expression assumes that the content will be located "starting from scratch" within the target document. It always begins with a "/" delimiter, whose presence is interpreted to mean, "Start at the document's root node."
Relative Location Steps

The starting point for a relative location step is always considered to be some previously established context node. This context node may be identified by an absolute location step, or, if the context node within the target document has already been set, by some relative location step. For more information, see Context and Navigation in XPath.
Constructing an XPath Location Step

A location step is constructed as follows:

axis::nodetest[predicate]

Each of the three components may or may not be present in a given expression. The expression as a whole is evaluated as a series of successively finer-grained sieves, resulting in the selection of one or more nodes from the target document:

- The axis (separated from the node-test, if there is one, by a pair of colons, ::) establishes a "direction to look," beginning at each of the nodes selected by the node-test. There is an axis corresponding to each type of relationship defined by XPath's tree model.
- The node-test determines the initial set of nodes to be selected.
- The predicate is set off from the node-test and axis, if they are present, by square bracket characters, [ and ]. If a predicate is present, it defines conditions to be met in order for a given node to be ultimately selected by the XPath expression.

Note  The result of selecting by location step, even if all three components are present, does not necessarily mean that only one node will be located.

Some sample location steps using the complete syntax are shown in the following table.

<table>
<thead>
<tr>
<th>Location step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::*[position()=1]</td>
<td>Locates the first child node of the context node</td>
</tr>
<tr>
<td>ancestor-or-self::*[@catdate=&quot;2000-12-31&quot;]</td>
<td>Locates all ancestors of any &lt;book&gt; child of the context node, as well as the &lt;book&gt; child itself, as long as the element in question has a catdate attribute with the value &quot;2000-12-31&quot;.</td>
</tr>
<tr>
<td>//self::parent[name()=&quot;book&quot;]</td>
<td>Locates any node in the document whose parent node is named &quot;book,&quot; or any node descended from the context</td>
</tr>
</tbody>
</table>
node whose name is "author".

Example

This example does not generate significant output. It demonstrates that no errors are generated by the above XPath expressions.

XML File (booksxpath.xml)

Use the Sample XML File for XPath Tree Model and change its href attribute to point to booksxpath.xsl.

XSLT File (booksxpath.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="//book">
    <p/>
    child::*[position()=1] --
    <xsl:value-of select="child::*[position()=1]"/>
    <p/>
    ancestor-or-self::book[@catdate="2000-12-31"]/author --
    <xsl:value-of select='ancestor-or-self::book[@catdate="2000-12-31"]/author'/>
    <p/>
    //self::parent[name()="book"] | descendant::node[name()="author"]
    <xsl:value-of select='//self::parent[name()="book"] | descendant::node[name()="author"]'/>
</xsl:template>

</xsl:stylesheet>
Constructing the Node-Test Portion of a Location Step

The node-test portion of a location step identifies the type of node(s), or the specific node(s), which make up the baseline collection of nodes from which to gather possible matches. It takes three possible forms:

- Use a node name to select nodes with that name.
- Use a node type, followed by empty parentheses, (), to select nodes of that type. The following node types are available:
  - `comment()` selects element nodes;
  - `text()` selects attribute nodes;
  - `processing-instruction()` selects processing instruction nodes;
  - and `node()` selects nodes of any kind.
- Use `processing-instruction("target")` to select processing instructions with a particular target.
Specifying the Axis Portion of a Location Step

The axis portion of an XPath location step specifies, like the axis of a two-dimensional graph, the direction in which to locate something relative to some starting point. There is an axis corresponding to each of the basic XPath tree model relationships:

- parent::
- child::
- ancestor::
- descendant::
- ancestor-or-self::
- descendant-or-self::
- preceding::
- following::
- preceding-sibling::
- following-sibling::
- self::
- attribute::
- namespace::

See Also

Use XPath Axes to Navigate through XML Data
Constructing the Predicate Portion of a Location Step

The final possible component of an XPath location step, called the \textit{predicate}, provides a further, finer-grained filter to select among the node(s) located by the node-test and axis. The basic form of a predicate is:

\[\text{something operator somevalue}\]

where:

- the square brackets, [ and ], are required;
- \textit{something} may be either a further location step, relative to the nodes matched so far by the node-test and axis, or a built-in XPath function;
- \textit{operator} is one of several Boolean operators, such as an equals sign or \texttt{!="} for a "does not equal" condition; and
- \textit{somevalue} is the value to which you want to compare \textit{something}.

For instance:

\[\text{position() = 1}\]

locates the first matching node in document order, while:

\[\text{attribute::empid != "73519"}\]

locates the subset of the matching node-set at that point which includes any nodes with an \texttt{empid} attribute whose string value does not equal "73519".

\textbf{Note} Be careful when using the \texttt{"<"} and \texttt{">"} operators in your XPath expressions. Because these characters are significant in markup, you will typically have to escape them using the \texttt{&lt;} and \texttt{&gt;} entity references. For example:

\[\text{attribute::empid &lt;= "69999"}\]

says to select all nodes with an \texttt{empid} attribute whose value is \texttt{<=} (less than
or equal to) "69999".

White space between the terms of a predicate is not significant. That is:

\[ \text{[position()} = 1] \]

and

\[ \text{[position()=1]} \]

function identically.

**See Also**

[Filtering XML Data Using XPath Predicates](#)
Understanding XPath Usage in XSLT

The following section covers topics related to using XPath when working with XSLT.

- Differences between XPath Expressions and XSLT Patterns
- How XSLT Defines the Context for XPath Expressions
- How XSLT Template Rules Depend on Context
- Determining the Current Context in an XSLT Application
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Differences between XPath Expressions and XSLT Patterns

An Extensible Stylesheet Language Transformations (XSLT) pattern may appear in a number of places in a style sheet:

- As the value of the match attribute of an `<xsl:template>` element. (This is the most common use of XSLT patterns.)
- As the value of the match attribute of an `<xsl:key>` element.
- As the values of the count and from attributes of an `<xsl:number>` element.

The purpose of a pattern is to restrict the set of candidate nodes in a node-set to just those nodes that meet a particular condition, or set of conditions. For instance:

```xml
<xsl:template match="/attribute::invoice_num[.='X00456']"/>
```

restricts the candidate nodes for the template rule just to the element(s) with an invoice_num attribute whose value is "X00456". This works even though from any given context node, in theory all other nodes in the documents are reachable candidates.

All XSLT patterns are also XML Path Language (XPath) expressions. However, the reverse is not always true: not all XPath expressions are XSLT patterns. For instance:

```xml
round(sales)
```

is a legitimate XPath expression, which returns the value of the `<sales>` element rounded to the nearest integer. But:

```xml
<xsl:template match="round(sales)"/>
```

is not a legitimate way to open an XSLT `<xsl:template>` element. Even though it refers to a node-set (sales) relative to the context node, taken as a whole the expression does nothing to constrain the universe of all possible candidate nodes at that point to a subset of them. Thus, such XPath expressions must always
appear in a pattern in some other form, such as within a predicate.

For example, we could use the round(sales) expression in this way in an XSLT pattern:

```xml
<xsl:template match=".//*[round(sales) &lt; 5000]">
  This says to restrict the universe of possible candidate nodes to those which are descended from the context node (.//), with any name (*), whose <sales> children have a value less than 5000 ([round(sales) &lt; 5000]).
</xsl:template>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
How XSLT Defines the Context for XPath Expressions

At run-time, the XSLT processor associates the templates in an XSLT file with specific source nodes, starting with the template that matches the document root node (match="/"). When the processor begins to process a template, the source node associated with that template becomes the current node. This current node defines a context node for evaluating the remaining XML Path Language (XPath) expressions within the template.

As the XSLT processor evaluates the XPath expressions within a template, the context node often changes based upon the results of the evaluation.

Having defined the current node, the XSLT processor begins evaluating the XPath expressions within the template.

Context for Select Expressions

The <xsl:value-of>, <xsl:for-each>, <xsl:copy-of>, <xsl:param>, <xsl:sort>, <xsl:variable>, <xsl:with-param>, and <xsl:apply-templates> elements have a select attribute. This expression is evaluated relative to the template's current node. In <xsl:for-each> or <xsl:apply-templates>, the select returns a node-set, and each node in this set becomes the current node for further queries within it.

The following example shows <xsl:for-each> and <xsl:value-of> elements are used in a simplified snippet from invoice.xsl (in Sample XSLT File for DOM Context).

```xml
<TABLE BORDER="1">
  <xsl:for-each select="invoices/invoice">
    <TR>
      <TD>Invoice #<xsl:value-of select="@id"/></TD>
    </TR>
    <xsl:for-each select="items/item">
      <TR>
        <TD><xsl:value-of select="qty"/></TD>
        <TD><xsl:value-of select="description"/></TD>
        <TD>$<xsl:value-of select="price"/></TD>
      </TR>
    </xsl:for-each>
  </xsl:for-each>
</TABLE>
```

The first <xsl:for-each> selects a set of <invoice> elements, each of which becomes the context for the "items/item" query. Each <item>, in turn, becomes the context for the various queries in the <xsl:value-of> elements.

Each node selected by <xsl:apply-templates> is associated with a template and therefore, becomes the current node for the template.

Because each expression defines a new context, a set of nested queries, such as those in the preceding example, produces a set or "path" of context nodes active at one time.
Context for Test Boolean Expressions

The conditional elements `<xsl:if>` and `<xsl:when>` do not define a new context for queries within themselves because they do not actually select new nodes. They just test to make sure the nodes are there. In the following example, the `<xsl:for-each select="price">` query is relative to `<item>` elements, despite the intervening conditional that tests the "qty" value:

```xml
<TABLE>
  <xsl:for-each select="items/item">
    <TR>
      <TD><xsl:value-of select="qty"/></TD>
      <TD><xsl:value-of select="description"/></TD>
      <TD>$<xsl:value-of select="price"/></TD>
      <TD><!-- 10% volume discount -->
        <xsl:if test="qty[.>=10]">
          <xsl:for-each select="price">
            <xsl:value-of select="formatNumber(.*.10, '$#,##0.00')"/>
          </xsl:for-each>
        </xsl:if>
        <!-- line total -->
        <xsl:value-of select="formatNumber(user:lineTotal(.), '$#,##0.00')"/>
      </TD>
    </TR>
  </xsl:for-each>
</TABLE>
```
Context for Match Patterns

The `<xsl:template>` element has a `match` attribute that accepts a pattern for the purposes of matching templates to specific elements; there is no fixed context for a query in this case. For more information, see Defining Match Patterns in `<xsl:template>`.

The preceding code listings are from the Invoice sample. For more information, see Sample XML File for DOM Context.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
How XSLT Template Rules Depend on Context

XSLT template rules cannot be relied on to fire in any particular order, each one is evaluated strictly in terms of the context established at that point. This context is strongly influenced both by the built-in XSLT template rules and by <xsl:apply-templates> elements in template rules for higher-level nodes.

Initial Example

The initial example produces some stray data before the desired "Forecast for (Today...)" headings.

XML File (weather.xml)

Use the Sample XML Data File for XPath Context and Navigation and change the href attribute to reference weather1.xsl.

XSLT File (weather1.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">
    <xsl:template match="forecast">
        <h2>Forecast for (Today <xsl:value-of select="@day"/>):</h2>
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="temperature">
        <p><b>Temperature: <xsl:value-of select="."/>
            <xsl:value-of select="@scale"/>
        </b></p>
    </xsl:template>

    <xsl:template match="humidity">
        <p><b>Relative humidity: <xsl:value-of select="."/></b>%
    </xsl:template>
```
The preceding style sheet appears to process three types of elements: <forecast>, <temperature>, and <humidity>.

The built-in template rules for XSLT are also in effect—in particular, the built-in rule for the root and all element nodes, which looks like this if coded explicitly:

```xml
<xsl:template match="/|*" >
  <xsl:apply-templates />
</xsl:template>
```

By default, all element content is output to the result tree—not just the content of the three explicitly transformed elements.

Data items are shown before the heading when the <forecast> template rule is evaluated, the context in which its match expression is evaluated is the context supplied by the default template rule. This works fine for processing the
<forecast> elements in the source document, and the corresponding template rules for the <temperature> and <humidity> children of the <forecast> elements are then invoked by the <xsl:apply-templates/> element.

But the context in the <temperature> and <humidity> templates' match expressions are evaluated is not just the context supplied by the <forecast> elements. There are other <temperature> and <humidity> elements in this document—the children of the <today> element. The <today> element's children are processed by any matching template rules.

Note The <wind> and <precip> elements' contents are passed to the result tree even though our style sheet does not include template rules for them. Again, this is a result of the built-in template rule for the root and element nodes.

You can control the XPath context which matched expressions are evaluated, by manipulating the context in two higher level template rules.

To change this document to display only the desired content from the <forecast> elements' <temperature> and <humidity>, use the following step:

1. Supply a specific template rule for the <today> element, suppressing its contents entirely; and
2. Supply specific child elements to be processed by the <forecast> element's <xsl:apply-templates/> element.

In Step 1, add the following template rule to the style sheet:

```
<xsl:template match="/weather/today" />
```

Note The empty template portion of this template rule. Nothing at all will be instantiated for the <today> element in the source tree.

In Step 2, change the <forecast> element's template rule, as follows:

```
<xsl:template match="forecast">
  <h2>Forecast for (Today <xsl:value-of select="@day"/>):</h2>
  <xsl:apply-templates select="temperature|humidity"/>
</xsl:template>
```

Corrected Example
XML File (weather.xml)

Use the Sample XML Data File for XPath Context and Navigation and change the href attribute to reference weathershort.xsl.

XSLT File (weathershort.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">

    <xsl:template match="/weather/today"/>

    <xsl:template match="forecast">
        <h2>Forecast for (Today <xsl:value-of select="@day"/>):</h2>
        <xsl:apply-templates select="temperature|humidity"/>
    </xsl:template>

    <xsl:template match="temperature">
        <p><b>Temperature: <xsl:value-of select="."/>
xscale</b></p>
    </xsl:template>

    <xsl:template match="humidity">
        <p><b>Relative humidity: <xsl:value-of select="."/>%</b></p>
    </xsl:template>

</xsl:stylesheet>
```

Formatted Output

**Forecast for (Today +1):**

Temperature: 81 F

Relative humidity: 30%

**Forecast for (Today +2):**

Temperature: 72 F

Relative humidity: 60%
Determining the Current Context in an XSLT Application

Any XML Path Language (XPath) expression referencing some nodes in a source document is evaluated according to conditions that are true at the time the evaluation occurs. When XPath is being used for a linking application, the context in most cases will not change; each link carries its own context with it. In XSLT applications, however, understanding the context is critical because a number of factors may cause the context to change at various points in the style sheet.

There are two general kinds factors of: static and dynamic.
Static context

The static context includes two general conditions that are true depending on the XPath expression's placement in the XSLT file.

- **Namespaces:** A given XPath expression found at two different points in an XSLT file may have two completely different meanings if the namespaces in effect at those two points are different.
- **Variable declarations:** An XPath expression, if it refers to a variable, correctly interpreting the expression depends on the declaration of that variable.
Dynamic context

The dynamic context does not depend on the location of the XPath expression in the XSLT file, but on the state of processing at the time the expression is evaluated. Four factors are included in the dynamic context: the context node, context size, context position, and current node.

See Also

Using XSLT Variables and Parameters | Determining the Context Node | Determining the Context Size | Determining the Context Position | Determining the Current Node
Using XPath with the DOM

The following section covers topics related to how use XPath when working in applications that use the XML Document Object Model (DOM).

- [How the DOM Defines the Context for XPath Expressions](#)
- [Sample XML File for DOM Context](#)
- [Sample XSLT File for DOM Context](#)
- [Sample Formatted Output for DOM Context](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
How the DOM Defines the Context for XPath Expressions

The Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office implementation contains methods that accept an XML Path Language (XPath) string to identify nodes within the XML tree. The `selectNodes` method returns a node list pointing to each of the selected nodes, as in the following example.

```javascript
items = xmldoc.selectNodes("invoices/invoice/items/item");
```

It takes a substantial amount of code to retrieve the `<item>` nodes by navigating the tree. With the `selectNodes` method and XPath expressions, nodes can be retrieved through a single line of code.

In most cases, the returned set of nodes are iterated over to perform further manipulations. The following example shows how you can calculate a total price for an `<invoice>` element.

```javascript
function invoiceTotal(invoice)
{
    invoice = invoice.nextNode();
    items = invoice.selectNodes("items/item");
    var sum = 0;
    for (var item = items.nextNode(); item; item = items.nextNode())
    {
        var price = item.selectSingleNode("price").nodeTypedValue;
        var qty = item.selectSingleNode("qty").nodeTypedValue;
        if (qty >= 10)
            price = 0.9*price;
        sum += price * qty;
    }
    return sum;
}
```

This example contains two query contexts. The first query is performed from the context of the `<invoice>` element, and the query string reflects this by beginning with the document element `<items>`. The `for` loop repeat through the returned nodes and uses each as a context for further queries. These query strings are relative to `<item>` elements, and look within them for the `<price>` and `<qty>` children. This illustrates that the node on which the `selectNodes` is performed
defines the context for the query.

Performing queries from the document root versus the document element can be confusing. These two nodes are different in the XML tree and therefore require different query strings. The following three statements yield identical results for the sample document, although only the last two are logically equivalent.

```javascript
items = xmldoc.selectNodes("invoices/invoice");
items = xmldoc.selectNodes("*/invoice");
items = xmldoc.documentElement.selectNodes("invoice");
```

The `selectSingleNode` method returns the first node that matches the query, providing a convenient way to return a node directly without extracting it from a NodeList. It is equivalent to the following:

```javascript
selectNodes(pattern).item(0)
```

The preceding example could simplify the code by using the `selectSingleNode` method.

```javascript
function invoiceTotal(invoice)
{
    invoice = invoice.nextNode();
    items = invoice.selectNodes("items/item");
    var sum = 0;
    for (var item = items.nextNode(); item; item = items.nextNode())
    {
        var price = item.selectSingleNode("price").nodeTypedValue;
        var qty = item.selectSingleNode("qty").nodeTypedValue;
        sum += price * qty;
    }
    return sum;
}
```

In the preceding example, `selectNodes` is used to select a set of `<item>` elements. When only a single node is expected, such as the `<price>` and `<qty>` elements, `selectSingleNode` can be used.

The preceding sample code is part of the Invoice sample. For more information, see Sample XML File for DOM Context.
Sample XML File for DOM Context

This file is part of the Invoice sample.

XML File (invoice.xml)

```xml
<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="invoice.xsl"?>
<invoices xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <customers>
    <customer id="2513">Joe</customer>
    <customer id="4871">Bob</customer>
    <customer id="7806">Jane</customer>
    <customer id="1579">Bruce</customer>
    <customer id="2533">John</customer>
    <customer id="1445">Bill</customer>
  </customers>
  <invoice id="5034">
    <customer ref="2513"/>
    <items>
      <item>
        <product-number>50014371</product-number>
        <description>Windbreaker (teal)</description>
        <qty dt:dt="number">1</qty>
        <price dt:dt="number">18.00</price>
      </item>
    </items>
  </invoice>
  <invoice id="5035">
    <customer ref="4871"/>
    <items>
      <item>
        <product-number>61570910</product-number>
        <description>Arctic Parka</description>
        <qty dt:dt="number">1</qty>
        <price dt:dt="number">160.00</price>
      </item>
      <item>
        <description>Signal Flare 6-pack</description>
        <qty dt:dt="number">3</qty>
        <price dt:dt="number">48.50</price>
      </item>
    </items>
  </invoice>
</invoices>
```
<description>Woolen Mittens</description>
<qty dt:dt="number">1</qty>
<price dt:dt="number">14.50</price>
</item>
</items>
</invoice>

<invoice id="5036">
<customer ref="7806"/>
<items>
  <item>
    <product-number>56812007</product-number>
    <description>Tropical Shade Hat</description>
    <qty dt:dt="number">1</qty>
    <price dt:dt="number">29.90</price>
  </item>
  <item>
    <product-number>74500231</product-number>
    <description>Bug Repellant</description>
    <qty dt:dt="number">3</qty>
    <price dt:dt="number">4.95</price>
  </item>
</items>
</invoice>

<invoice id="5037">
<customer ref="1579"/>
<items>
  <item>
    <product-number>65040097</product-number>
    <description>Arctic Parka</description>
    <qty dt:dt="number">2</qty>
    <price dt:dt="number">160.00</price>
  </item>
  <item>
    <product-number>2310078</product-number>
    <description>Tundra Jacket</description>
    <qty dt:dt="number">1</qty>
    <price dt:dt="number">105.50</price>
  </item>
</items>
</invoice>

<invoice id="5038">
<customer ref="2533"/>
<items>
  <item>
    <product-number>54807910</product-number>
    <description>DVD Player</description>
    <qty dt:dt="number">1</qty>
    <price dt:dt="number">342</price>
  </item>
</items>
</invoice>
<invoice id="5039">
  <customer ref="1445"/>
  <items>
    <item>
      <product-number>65470025</product-number>
      <description>Adventure Rations (Tofu)</description>
      <qty dt="number">20</qty>
      <price dt="number">6.00</price>
    </item>
    <item>
      <product-number>65470025</product-number>
      <description>Adventure Rations (Chicken)</description>
      <qty dt="number">20</qty>
      <price dt="number">6.00</price>
    </item>
    <item>
      <product-number>24500358</product-number>
      <description>Adventure Rations (Beef)</description>
      <qty dt="number">20</qty>
      <price dt="number">6.00</price>
    </item>
  </items>
</invoice>
</invoices>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Sample XSLT File for DOM Context

This file is part of the Invoice sample.

XSLT File (invoice.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt"
    xmlns:user="mynamespace"
    version="1.0">
    <xsl:template match="/">
        <HTML>
            <STYLE>
                TD {font-size:9pt}
            </STYLE>
            <BODY STYLE="font:9pt Verdana">
                <H3>Invoices</H3>
                <TABLE BORDER="1">
                    <TR>
                        <TD><B>Qty</B></TD>
                        <TD><B>Description</B></TD>
                        <TD><B>Price</B></TD>
                        <TD><B>Discount</B></TD>
                        <TD><B>Total</B></TD>
                    </TR>
                    <xsl:for-each select="invoices/invoice">
                        <xsl:variable name="customer-id" select="./customer/@ref"/>
                        <TR>
                            <TD COLSPAN="5" STYLE="background-color: #DDDDDD">
                                Invoice #<xsl:value-of select="@id"/>,
                                for customer: <xsl:value-of select="/invoices/customers/customer[@id=$customer-id]"/>
                            </TD>
                        </TR>
                        <xsl:for-each select="items/item">
                            <TR>
                                <TD>
                                    <xsl:value-of select="qty"/>
                                </TD>
                                <TD>
                                    <xsl:value-of select="description"/>
                                </TD>
                                <TD>
                                    $<xsl:value-of select="format-number(price,'#,##0.'entication, ',###0."
                                </TD>
                            </TR>
                        </xsl:for-each>
                    </xsl:for-each>
                </TABLE>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```
# Sample Formatted Output for DOM Context

## Invoices

<table>
<thead>
<tr>
<th>Qty</th>
<th>Description</th>
<th>Price</th>
<th>Discount</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windbreaker (teal)</td>
<td>$18.00</td>
<td></td>
<td>$18.00</td>
</tr>
<tr>
<td>1</td>
<td>Arctic Parka</td>
<td>$160.00</td>
<td></td>
<td>$160.00</td>
</tr>
<tr>
<td>3</td>
<td>Signal Flare 6-pack</td>
<td>$48.50</td>
<td></td>
<td>$145.50</td>
</tr>
<tr>
<td>1</td>
<td>Woolen Mittens</td>
<td>$14.50</td>
<td></td>
<td>$14.50</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>$320.00</strong></td>
</tr>
<tr>
<td>1</td>
<td>Tropical Shade Hat</td>
<td>$29.90</td>
<td></td>
<td>$29.90</td>
</tr>
<tr>
<td>3</td>
<td>Bug Repellant</td>
<td>$4.95</td>
<td></td>
<td>$14.85</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>$44.75</strong></td>
</tr>
<tr>
<td>2</td>
<td>Arctic Parka</td>
<td>$160.00</td>
<td></td>
<td>$320.00</td>
</tr>
<tr>
<td>1</td>
<td>Tundra Jacket</td>
<td>$105.50</td>
<td></td>
<td>$105.50</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>$425.50</strong></td>
</tr>
<tr>
<td>1</td>
<td>DVD Player</td>
<td>$342.00</td>
<td></td>
<td>$342.00</td>
</tr>
<tr>
<td>1</td>
<td>Odyssey Mousepad</td>
<td>$15.00</td>
<td></td>
<td>$15.00</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>$357.00</strong></td>
</tr>
<tr>
<td>20</td>
<td>Adventure Rations (Tofu)</td>
<td>$5.00</td>
<td>$0.60</td>
<td>$108.00</td>
</tr>
<tr>
<td>20</td>
<td>Adventure Rations (Chicken)</td>
<td>$5.00</td>
<td>$0.60</td>
<td>$108.00</td>
</tr>
<tr>
<td>20</td>
<td>Adventure Rations (Beef)</td>
<td>$5.00</td>
<td>$0.60</td>
<td>$108.00</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>$324.00</strong></td>
</tr>
</tbody>
</table>
Context and Navigation in XPath

The following section covers evaluating, setting, and changing the context in which an XPath expression, or where succeeding an XPath expressions is evaluated.

- Determining the Context Node
- Determining the Context Size
- Determining the Context Position
- Determining the Current Node
- Understanding How the Context Node Changes
- Sample XML Data File for XPath Context and Navigation
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Determining the Context Node

The context node of an XML Path Language (XPath) location path is the node relative to the expression being evaluated.

In lengthy XPath expressions, the context node implicitly changes as the processor hits various steps in moving along the expression from left to right. For example, consider the expression:

\[ \text{attribute::scale/ancestor::weather/forecast[position()=2]/attribute:} \]

- When evaluation of the expression begins, the context node is the parent element of a scale attribute.
- The /ancestor::weather term is evaluated in terms of the scale attribute.
- /forecast[position()=2] assumes the context node to be the result of the previous location step, that is, in the case of the sample document, the second \(<\text{forecast}\>\) child of the \(<\text{weather}\>\) element.
- /attribute::day locates the day attribute of the elements resulting from the previous step.

For XSLT, understanding the context node, and knowing what it is at any given point, is critical to successful use of the standard.

Example

In the following XSLT file, the XPath expressions appear as bold text.

**XSLT File (weathersimple.xsl)**

```xml
&lt;?xml version='1.0'?&gt;
&lt;xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' ver
&lt;!-- suppress text nodes not covered in subsequent template rule --&gt;
&lt;xsl:template match="text()"/&gt;

&lt;xsl:template match="temperature"
   Today&lt;xsl:choose&gt;
      &lt;xsl:when test="name(..)='today'">'s &lt;/xsl:when&gt;
      &lt;xsl:otherwise &gt;&lt;xsl:value-of select="../@day" />'s &lt;/xsl:ot
   &lt;/xsl:choose&gt;
```
What is displayed in the result tree by this template rule depends entirely on the context node. The template rule matches a single element type in the source tree, <temperature>, and the corresponding match pattern (match="temperature") is what establishes the context node within the body of the template rule. However, the <temperature> element type can be a child of either the <today> element or a <forecast> element, and the <xsl:choose> element operates accordingly:

- Is the parent element named <today>? (This is the test="name(.)='today'" attribute in the <xsl:when> element.) If so, just output "Today's temperature:" and the value of the <temperature> element itself.
- If the parent element is not named <today> (this is the <xsl:otherwise> element's role), output the word "Today," the value of that parent element's day attribute (e.g. "+1"), and the string "'s temperature: ".

If the context node did not change each time this template rule fired, it would not be possible to produce different output for each occurrence of the <temperature> element.

**XML File (weather.xml)**

Change the href attribute in weather.xml (shown in Sample XML Data File for XPath Context and Navigation) to reference weathersimple.xsl.

**Formatted Output**

Today's temperature: 76
Today+1's temperature: 81
Today+2's temperature: 72

**Processor Output**

```xml
<?xml version="1.0" encoding="UTF-16"?>
	Today's
temperature: 76<br />
	Today+1's
temperature: 81<br />
```
Today+2's
temperature: 72
Determining the Context Size

Many XPath expressions locate just a single node in the source tree. For example, referring to the sample XML document introduced earlier in this topic, this XPath expression:

today

locates just one element node.

However, in many other cases the XPath expression locates more than one node or rather, an indeterminate number of nodes, ranging from 0 on up—that is, a node-set. For example:

forecast

When applied to the sample document, it locates not just a single node but a node-set, and the size of that node-set—the number of nodes in it, or two in this case—can be critical in determining what your style sheet needs to do at a given point.

The context size is always accessible as the value of the XPath last() and count() functions, each returning an integer. Use last() when you are already working with the node-set in question. Use count() when you need to pass it to another node-set. This can be used for calculating the average of all temperatures in your sample weather-related XML document, when used in concert with the sum() function.

The last() and count() functions are useful in laying out tables whose number of columns, fare fixed but whose number of rows vary depending on the number of occurrences of some element in the source tree.

For more information about last() and count(), see Processing Node-Sets by Using Node-Set Functions. For more information about sum(), see Using Numeric Functions to Perform Math Operations in XPath.

Example
XML File (weather.xml)

Change the href attribute in weather.xml (shown in Sample XML Data File for XPath Context and Navigation) to reference weatherconsize.xsl.

XSLT File (weatherconsize.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="/">
    Average temperature (for <xsl:value-of select="count(.//temperature)"/> days) :
    <xsl:value-of
      select="sum(.//temperature) div count(.//temperature)"/>
    degree
  </xsl:template>

</xsl:stylesheet>
```

Formatted Output

Average temperature (for 3 days) : 76.33333333333333 degrees

Processor Output

```
<?xml version="1.0" encoding="UTF-16"?>
Average temperature (for 3 days) :
76.33333333333333 degrees
```
Determining the Context Position

Similar to context size, context position, where you are in the node-set, is as important in determining what a style sheet needs at any one time. A typical example can be performing an action for every Nth node.

You can use the Sample XML Data File for XPath Context and Navigation, to display information for each day on a separate line, coloring the odd lines white and the even lines yellow. This provides easy legibility when there are a large amount of data to be displayed. Use the following XSLT template rule to display just the temperature values:

The following template rule display just the temperature values:

- The match pattern, match="weather/*", selects all children (*) of all children named "weather" of the context node, that is, of the document root node. Therefore, it selects a node-set consisting of the <today> element and the two <forecast> elements.
- The template rule declares a local variable, called odd_or_even, whose value is either the "odd" or "even" string, depending on the value of the position within the node-set of the node currently being processed.
- The test uses the position() function, which returns an integer whose value is that of the current node's position in the node-set. The test calculates the remainder of this integer when divided by 2. If the result is 0, then the result of the test is true, and $odd_or_even is "even"; otherwise it is "odd".
- The <xsl:choose> element that follows the variable's assignment tests the variable's value. If "odd", the temperature is inserted into a <p> element whose background-color is white; otherwise, the <p> element's background-color will be yellow.

**Note** The use of the position() function in the <xsl:variable> code.

For more information about position(), see Processing Node-Sets by Using Node-Set Functions.

**Example**
XML File (weather.xml)

Change the href attribute in `weather.xml` (shown in Sample XML Data File for XPath Context and Navigation) to reference `weathercontextposn.xsl`.

XSLT File (weathercontextposn.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:variable name="odd_or_even">
        <xsl:choose>
            <xsl:when test="position() mod 2 = 0">even</xsl:when>
            <xsl:otherwise>odd</xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:choose>
        <xsl:when test="$odd_or_even = 'even'">
            <p style="background-color: yellow">even row: <xsl:value-of select="./temperature" /></p>
        </xsl:when>
        <xsl:otherwise>
            <p style="background-color: white">odd row: <xsl:value-of select="./temperature" /></p>
        </xsl:otherwise>
    </xsl:choose>
</xsl:stylesheet>
```

Formatted Output

```
odd row: 76
even row: 81
odd row: 72
```

Processor Output

```xml
<?xml version="1.0" encoding="UTF-16"?><p style="background-color: white">76</p><p style="background-color: yellow">even row: 81</p><p style="background-color: white">odd row: 72
```
72</p>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Determining the Current Node

A node in a node-set becomes the current node as a result of the implicit looping to process all nodes of a certain kind in an `<xsl:apply-templates>` or `<xsl:for-each>` element. You can always access the current node, using the `current()` function. This will be identical to the context node and can be represented in an XPath expression as "." (a single period, which is the abbreviated syntax for `self::node()`).

The one exception may be when a node is referenced in the predicate of an XPath expression. The `context` node remains the node currently being processed in the node-set, while the `current` node temporarily shifts depending on the XPath expression in the predicate.

The two XSLT template rules in the XSLT sample file below recursively walk through the document, displaying the context node, the current node, and the value of all elements in `weather.xml`.

The formatted output below shows that the context node and the current node are the same all the way through the document.

**Example**

**XML File (weather.xml)**

Change the `href` attribute in `weather.xml` (shown in Sample XML Data File for XPath Context and Navigation) to reference `weathercurrnodel.xsl`.

**XSLT File (weathercurrnodel.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <table border="1">
      <tr>
        <th>Context</th>
        <th>Current</th>
        <th>Node value</th>
      </tr>
      <xsl:apply-templates/>
    </table>
  </xsl:template>
</xsl:stylesheet>
```
<?xml version="1.0" encoding="UTF-16"?>
<table border="1">

<table>
<thead>
<tr>
<th>Context</th>
<th>Current</th>
<th>Node value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weather</td>
<td>weather</td>
<td>76 67 5 81 30 10 Rain 72 60 2 Fog</td>
</tr>
<tr>
<td>today</td>
<td>today</td>
<td>76 67 5</td>
</tr>
<tr>
<td>temperature</td>
<td>temperature</td>
<td>76</td>
</tr>
<tr>
<td>humidity</td>
<td>humidity</td>
<td>67</td>
</tr>
<tr>
<td>wind</td>
<td>wind</td>
<td>5</td>
</tr>
<tr>
<td>precip</td>
<td>precip</td>
<td>10</td>
</tr>
<tr>
<td>forecast</td>
<td>forecast</td>
<td>81 30 10 Rain</td>
</tr>
<tr>
<td>temperature</td>
<td>temperature</td>
<td>81</td>
</tr>
<tr>
<td>humidity</td>
<td>humidity</td>
<td>30</td>
</tr>
<tr>
<td>wind</td>
<td>wind</td>
<td>10</td>
</tr>
<tr>
<td>precip</td>
<td>precip</td>
<td>Rain</td>
</tr>
<tr>
<td>forecast</td>
<td>forecast</td>
<td>72 60 2 Fog</td>
</tr>
<tr>
<td>temperature</td>
<td>temperature</td>
<td>72</td>
</tr>
<tr>
<td>humidity</td>
<td>humidity</td>
<td>60</td>
</tr>
<tr>
<td>wind</td>
<td>wind</td>
<td>2</td>
</tr>
<tr>
<td>precip</td>
<td>precip</td>
<td>Fog</td>
</tr>
</tbody>
</table>

</table>
See Also

[Using Abbreviations in XPath Expressions](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Understanding How the Context Node Changes

The context node implicitly changes as the XPath-aware processor works its way through the location steps in a non-important location path.

There is no way to demonstrate this built-in changing of the context node by the processor, because the result of the XPath expression is returned to you as a whole, not as a series of steps. You can simulate the process, by decomposing a complete location path into its parts, and displaying where you are at each point.

Part of the data from the sample file, `weather.xml`, is shown here:

```xml
<weather>
  <today>
    <temperature scale="F">76</temperature>
    <humidity>67</humidity>
    <wind unit="mph">5</wind>
    <!-- No precipitation today -->
    <precip />
  </today>
  <forecast day="+1">
    <temperature scale="F">81</temperature>
    <humidity>30</humidity>
    <wind unit="mph">10</wind>
    <precip>Rain</precip>
  </forecast>
  <forecast day="+2">
    <temperature scale="F">72</temperature>
    <humidity>60</humidity>
    <wind unit="mph">2</wind>
    <precip>Fog</precip>
  </forecast>
</weather>
```

You can trace the steps through the following location path, using the style sheet `weather.xsl`:

```
/weather/forecast/wind/@unit
```

Steps in a location path are delimited by the slash character, `/`. 
The only things you want to see are the results of evaluating each specific one of those steps, one at a time. Emphasizing tight XSLT code is not in this example. Here are the templates from the style sheet:

**Example**

The five template rules in this style sheet follow the same general method, aside from formatting the output for readability:

- Match on the portion of the location path of interest.
- Display the path itself, the name of the context node, and the value of the context node.
- Process the template rule specifically established for the next portion of the location path.

**Note** Using a simple `<xsl:apply-templates />`, without the select attribute, sometimes causes extraneous text to be displayed in the output. For example, omitting a select attribute in the template rule for the `<forecast>` element's `<xsl:apply-templates/>` would also cause the `<temperature>` and `<precipitation>` elements to be processed by the default template rules.

**XML File (weather.xml)**

Use the [Sample XML Data File for XPath Context and Navigation](#) and change the `href` attribute to reference `weather.xsl`.

**XSLT File (weather.xsl)**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40" >

<xsl:template match="/">
    <p>
```
<b>Path: /</b><br />
Name of context node: <b><xsl:value-of select="name()"/></b>
Value: <b><xsl:value-of select="."/></b>
</p>
<xsl:apply-templates select="weather"/>
</xsl:template>
<xsl:template match="/weather">
<p>
<b>Path: /weather</b><br />
Name of context node: <b><xsl:value-of select="name()"/></b>
Value: <b><xsl:value-of select="."/></b>
</p>
<xsl:apply-templates select="forecast"/>
</xsl:template>
<xsl:template match="/weather/forecast">
<p>
<hr width="400" size="1" align="left" />
<b>Path: /weather/forecast</b><br />
Name of context node: <b><xsl:value-of select="name()"/></b>
Value: <b><xsl:value-of select="."/></b>
</p>
<xsl:apply-templates select="wind"/>
</xsl:template>
<xsl:template match="/weather/forecast/wind">
<p>
<b>Path: /weather/forecast/wind</b><br />
Name of context node: <b><xsl:value-of select="name()"/></b>
Value: <b><xsl:value-of select="."/></b>
</p>
<xsl:apply-templates select="@unit"/>
</xsl:template>
<xsl:template match="/weather/forecast/wind/@unit">
<p>
<b>Path: /weather/forecast/wind/@unit</b><br />
Name of context node: <b><xsl:value-of select="name()"/></b>
Value: <b><xsl:value-of select="."/></b>
</p>
</xsl:template>
</xsl:stylesheet>

**Formatted Output**
Path: /
Name of context node: Value: 76 67 5 81 30 10 Rain 72 60 2 Fog

Path: /weather
Name of context node: weather Value: 76 67 5 81 30 10 Rain 72 60 2 Fog

Path: /weather/forecast
Name of context node: forecast Value: 81 30 10 Rain

Path: /weather/forecast/wind
Name of context node: wind Value: 10

Path: /weather/forecast/wind/@unit
Name of context node: unit Value: mph

Path: /weather/forecast
Name of context node: forecast Value: 72 60 2 Fog

Path: /weather/forecast/wind
Name of context node: wind Value: 2

Path: /weather/forecast/wind/@unit
Name of context node: unit Value: mph

Processor Output

```xml
<?xml version="1.0" encoding="UTF-16"?><p xmlns="http://www.w3.org/TR/REC-html40"
  Name of context node: <b>&lt;b&gt;</b>Value: <b>&lt;b&gt; 76 67 5 81 30 10 Rain 72 60 2 Fog &lt;/b&gt;&lt;/p
  Name of context node: &lt;b&gt;weather&lt;/b&gt;Value: &lt;b&gt; 76 67 5 81 30 10 Rain 72 60 2 Fog &lt;/b&gt;&lt;p
  Name of context node: &lt;b&gt;forecast&lt;/b&gt;
...
```
```
Sample XML Data File for XPath Context and Navigation

Topics in this section use the following XML data file. To view this with an XSLT file other than weather.xsl, change the href attribute.

**XML File (weather.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="weather.xsl" ?>
<!-- Readings for 2000-11-01 -->
<weather>
    <today>
        <temperature scale="F">76</temperature>
        <humidity>67</humidity>
        <wind unit="mph">5</wind>
        <!-- No precipitation today -->
        <precip />
    </today>
    <forecast day="+1">
        <temperature scale="F">81</temperature>
        <humidity>30</humidity>
        <wind unit="mph">10</wind>
        <precip>Rain</precip>
    </forecast>
    <forecast day="+2">
        <temperature scale="F">72</temperature>
        <humidity>60</humidity>
        <wind unit="mph">2</wind>
        <precip>Fog</precip>
    </forecast>
</weather>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Navigating XML Data Using XPath Axes

An axis in XPath is similar to an axis in a traditional two or three dimensional chart or graph. It tells the XPath-aware processor which direction to look, up, down, or sideways in a document tree, starting from a given point of origin—the context node. Using an axis as part of a location step is:

\[ \text{axisname}::\text{nodetest} \]

Where \textit{axisname} is one of the built-in axes defined by the XPath specification, selected from the list below, and \textit{nodetest} indicates which node(s) along that axis are of interest. For example:

\[ \text{child}::* \]

indicates to select all children of the context node, while

\[ /\text{child}::* \]

is to select all children of the root node, which is made the context node by virtue of the opening slash.

Using an axis in an XPath expression extends your options in navigating through the document tree. A traditional unidirectional path, such as to a file on a Web server, is similar to the following:

\[ /\text{rootfolder/subfolder/subsubfolder} \]

unlike this sort of path or a traditional charting application with its X, Y, and Z axes, XPath provides you with a rich variety of axes. Each XPath axis consist of a tree-like structure to the context node.

For more information, see Use XPath Axes to Navigate through XML Data.
Filtering XML Data Using XPath Predicates

This section contains the following topics.

- Introduction to Filtering with XPath Predicates
- Testing within a Predicate by Using Boolean Operators
- Testing the Context Position by Using a Numeric Shortcut
- Specifying Multiple Predicates in a Single XPath Step
Introduction to Filtering with XPath Predicates

An XPath location path consists of one or more location steps, delimited by slashes. For example:

/child::book/child::chapter/attribute::date_written

is an XPath location path which returns a node-set consisting of all date_written attributes of those <chapter> elements which themselves are children of the root <book> element.

Note This example could also be expressed using XPath's abbreviated syntax as:

/book/chapter/@date_written

Remaining examples in this topic will use the abbreviated syntax. For more information, see Using Abbreviations in XPath Expressions.

In general, the purpose of a location path is to return a set of one or more nodes—a node-set—meeting specified criteria. Each location step in the location path consists of an axis (explicit or defaulted to child::) relative to the context node, a node test, and an optional predicate. (The above example does not include a predicate in any of the three location steps.) All nodes which meet all criteria specified by the full location path are returned by the expression.

The general format of a predicate in a location step is:

[condition]

where the square brackets, [ and ], are required, and condition represents some test typically resulting in a Boolean true or false value. The condition usually takes the form:

value operator value
If a particular condition itself may be represented as a Boolean true or false value, the predicate can take a simpler form:

\[ \text{value} \]

**Note** This simplified form of the predicate is very important in limiting a location path only to nodes which have other nodes along a particular axis. If there are any nodes of the given name or type along the indicated axis, the result is true; if there are no such nodes, the result is false. Thus:

```
chapter[descendant::figure]
```

selects `<chapter>` elements which are children of the context node *only if* they have any descendant `<figure>` elements.

As an example, the set of nodes returned by the location path at the beginning of this topic could be further limited as follows:

```
/book/chapter[position()&gt;1]/@date_written[."2001-05-12"]
```

This location path can be read, from left to right:

2. Locate all `<chapter>` elements but the first, in document order, which are children of the root `<book>` element located by Step 1 (`chapter[position()&gt;1]`).
   **Note** A predicate, like everything else in an XML document, must adhere to the XML rules for well-formed documents. Therefore, operators such as `>` (in this example), `<`, and `&`, which have special meaning to XML parsers, must be escaped using entity references such as `&gt;`, `&lt;`, and `&amp;` respectively.

3. For the `<chapter>` elements selected by Step 2, locate all of their `date_written` attributes whose values equal the string "2001-05-12` (`@date_written[."2001-05-12"]`).

The result of evaluating this location path is a node-set consisting of the `date_written` attributes (if any) meeting all these criteria. If no `date_written` attributes meet the criteria, the result is an empty node-set.
White space between the value and operator terms in a predicate is not significant. Thus, the following two predicates produce the same result:

[position() = 1]
[position()=1]

See Also

Use XPath Axes to Navigate through XML Data | Determining the Context Node | Constructing the Node-Test Portion of a Location Step
Testing within a Predicate by Using Boolean Operators

As a general rule, the predicate of a location step narrows the range of candidate nodes located by the step to just those that meet the condition specified by the predicate.

The result of such a condition is always a Boolean true or false value. Therefore, the complete syntax for predicates allows you to use a number of standard Boolean operators to compare two values. The table below summarizes these operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>If values to left and right of the &quot;=&quot; are equal, the result is true; otherwise the result is false.</td>
</tr>
<tr>
<td>!=</td>
<td>If values to left and right of the &quot;=&quot; are not equal, the result is true; otherwise the result is false.</td>
</tr>
<tr>
<td>&lt;</td>
<td>If value to the left of the &quot;&lt;&quot; is less than value to the right, the result is true; otherwise the result is false.</td>
</tr>
<tr>
<td>&gt;</td>
<td>If value to the left of the &quot;&gt;&quot; is greater than value to the right, the result is true; otherwise the result is false.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>If value to the left of the &quot;&lt;=&quot; is less than or equal to value to the right, the result is true; otherwise the result is false.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>If value to the left of the &quot;&gt;=&quot; is greater than or equal to value to the right, the result is true; otherwise the result is false.</td>
</tr>
<tr>
<td>test1 and test2</td>
<td>If the results of both test1 and test2 are true, then the result of the test as a whole is true; otherwise the result is false.</td>
</tr>
<tr>
<td>test1 or test2</td>
<td>If the result of either test1 or test2 is true, then the result of the test as a whole is true; otherwise the result is false.</td>
</tr>
</tbody>
</table>

A predicate must adhere to XML’s rules for well-formed documents. Therefore, operators such as >, <, and &, which have special meaning to XML parsers, must be escaped using entity references such as &gt;, &lt;, and &amp; respectively.
Example

This example is used to demonstrate the operators shown below.

**XML File (book_qse.xml)**

See [Sample XML Data File for Boolean Operators in Predicates](#). This document represents a book, titled *Quantum Superstring Electrodynamics for Newbies*, which consists of two chapters. Each chapter contains two sections of text, intermingled with footnote references and pointers to illustrations; following the last `<section>` element in each chapter is a list of the footnotes referred to in the text. Footnotes will be numbered sequentially from the beginning of the book to the end.

**XSLT File (predicate.xsl)**

See [Sample XSLT File for Boolean Operators in Predicates](#).

Portions of this example style sheet which are relevant to the discussion of XPath predicates are reproduced below. The example style sheet includes many match and select patterns which could be replaced with more concise or efficient forms. Some of these patterns are also artificial, such as those which select for paragraphs beginning with certain letters of the alphabet. The style sheet's main purpose is to demonstrate various forms of XPath predicates, not to serve as an example of well-tuned XSLT code.

**Formatted Output**

See [Sample Output for Boolean Operators in Predicates](#).
The = operator

Use an equals sign (=) in a predicate when you need to test whether one value exactly equals another. For instance:

<xsl:when test="parent::*[name()='chapter']">

locates the parent of the context node only when that parent is named <chapter>. 
The != operator

Use the "not-equals" operator (!=) in a predicate when you need to test whether two values are not equal. For instance:

<xsl:template match="title[name(..)!='book']">

locates a <title> child of the context node only if that child's parent (that is, the context node itself) is not named <book>.

**Note** The != operator can sometimes be used interchangeably with the XPath not() function. For instance, these two predicates are interchangeable:

```
[name()!='book']
```

```
[not(name()='book')]
```

In such cases, which form of the test you use is a matter of taste. However, the two forms are not always identical—particularly when one of the values on either side of the operator is a node-set. For more information about the not() function, see [Testing an Expression by Using Boolean Functions](#).
The &lt; operator

Use the less-than operator in a predicate when you need to test whether one value is less than another. For instance:

```xml
<xsl:template match="chapter[position()&lt;2]">
locates a <chapter> child of the context node when that child is at an ordinal position less than 2 (i.e., when it is in the first position) among the current node-set.
The &gt; operator

Use the greater-than operator in a predicate when you need to test whether one value is greater than another. For instance:

```xml
<xsl:template match="chapter[position()&gt;1]">

locates a <chapter> child of the context node only if that child is greater than the first such child (that is, the second child, third child, and so on), in document order.
The \texttt{\&lt;=} operator

Use the less-than-or-equals operator in a predicate when you need to test whether one value is less than or equal to another. For instance:

\begin{verbatim}
<xsl:template match="@copyright[.\&lt;=2000]">

locates all copyright attributes with a value of 2000 or less.
The \&gt;= operator

Use the greater-than-or-equals operator in a predicate when you need to test whether one value is greater than or equal to another. For instance:

<\text{xsl:template \ match="@copyright[.\&gt;=\text{2001}]"}>

locates all copyright attributes with a value of 2001 or greater.
The "and" operator

Use the and operator in a predicate when you need to perform more than one test in order to select from among candidate nodes. All of the tests joined by the and operator must be true in order for the selection to occur. For instance:

```xml
<xsl:template match="text[substring(.,1,1)!='T' and substring(.,1,1)!='B'
and substring(.,1,1)!='A' and substring(.,1,1)!='P']">

This match pattern locates all <text> elements which start with anything except the letters "T," "B," "A," or "P." <text> elements beginning with any of those four letters will not be located.
The "or" operator

Use the or operator in a predicate when you need to perform any of several tests in order to select from among candidate nodes. Any of the tests joined by the or operator can be true in order for the selection to occur. For instance:

```xml
<xsl:template match="text[substring(.,1,1)='T' or substring(.,1,1)='B' or substring(.,1,1)='A' or substring(.,1,1)='P']">

This match pattern locates all `<text>` elements which start with any of the letters "T," "B," "A," or "P." `<text>` elements beginning with any other characters will not be located.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Sample XML Data File for Boolean Operators in Predicates

For more information about this file, see Testing within a Predicate by Using Boolean Operators.

XML File (book_qse.xml)

<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="predicate.xsl"?>
<book copyright="2001">
  <title>Quantum Superstring Electrodynamics for Newbies</title>
  <chapter date_written="2000-11-03">
    <title>QSE Background</title>
    <section>
      <title>History</title>
      <text>The history of quantum superstring...<noteref idref="def_ed"></text>
      <text>On the other hand...<noteref idref="clarif_01"></text>
      <text>Still,...</text>
    </section>
    <section>
      <title>First Principles</title>
      <text>Before proceeding further...<noteref idref="Feyn"></text>
      <text>We will see many examples...</text>
    </section>
  </chapter>
  <chapter date_written="2000-12-16">
    <title>Quanta</title>
    <section>
      <title>Packaging the Lightning</title>
      <text>A "quantum," basically, represents...<noteref idref="exp_01"></text>
      <text>Yes, it is true that we cannot draw conclusions fr</text>
    </section>
    <section>
      <title>Quanta within Quanta</title>
      <text>Picture a set of nested boxes, such as Figure 1:...<figure location="media/figure_01.png" caption="Figure 1: Boxes within boxes within boxes.."/>
What do we mean when we say...? as we shall see over the next five to ten exciting years.

Try it yourself! Take your laser...

Don't let the term "poison" alarm.
Sample XSLT File for Boolean Operators in Predicates

XSLT File (predicate.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40" >

<xsl:output method="html" />

<!-- Override built-in template rules for elements and text nodes (note empty <xsl:template element). -->
<xsl:template match="*"/>

<!-- Process the document root. -->
<xsl:template match="/">
    <html>
        <head><title><xsl:value-of select="book/title"/></title></head>
        <h1><xsl:value-of select="book/title"/></h1>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>

<xsl:template match="book">
    <xsl:apply-templates select="@*|*"/>
</xsl:template>

<xsl:template match="@copyright[.&gt;=2001]">
    <cite>
        [Copyright <xsl:value-of select="."/>
        (21st century)]
    </cite>
</xsl:template>

<xsl:template match="@copyright[.&lt;=2000]">
    <cite>
        [Copyright <xsl:value-of select="."/>
        (Previous century)]
    </cite>
</xsl:template>
```
Sample Output for Boolean Operators in Predicates

Formatted Output

The top portion of the formatted output is shown here.
Quantum Superstring Electrodynamics for Newbies

[Copyright 2001 (21st century)]

---

**History**

The history of quantum superstring... [Note 1]

On the other hand... [Note 2]

Still...

**First Principles**

Before proceeding further... [Note 3]

We will see many examples...

---

Notes for this chapter:

[1] The term "electrodynamics" refers to...

[2] Not to be confused with...

[3] Or, as Feynman once said...

---

**Packaging the Lightning**

A "quantum," basically, represents... [Note 4]

**Processor Output**

<html xmlns="http://www.w3.org/TR/REC-html40">
<head><title>Quantum Superstring Electrodynamics for Newbies</title>
</head>
Quantum Superstring Electrodynamics for Newbies

...as we shall see over the next five to ten exciting years.

Notes for this chapter:

[4] Try it yourself! Take your laser...

[5] Don't let the term "poison" alarm...
Testing the Context Position by Using a Numeric Shortcut

In most cases, an XPath predicate has a Boolean value true or false, resulting from the test contained within the [ and ] characters.

A common test appearing in predicates compares the context position to some value. For instance:

```
chapter[position()=1]
```

selects the first `<chapter>` element which is a child of the context node.

XPath provides a convenient shortcut for this common sort of test: simply use the number of the position you want to test. For example, the following behaves exactly like the above:

```
chapter[1]
```

The value of the predicate need not be a literal number. It can also be a reference to a variable which has a numeric value, for example, or the value of the `number()` function applied to some node or text string. Therefore, both of the forms below also demonstrate acceptable uses of this shortcut:

```
chapter[$xref]
chapter[number("12")]
```

Note that this shortcut is available only for testing whether the context position is equal to some constant value. There is no similar shortcut for testing greater-than, not-equal-to, and similar conditions.

See Also

[Determining the Context Position](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Specifying Multiple Predicates in a Single XPath Step

An XPath location step will usually include only a single predicate. Under certain circumstances, however, it may be necessary to further filter the node-set returned by the first predicate, within a single XPath location step.

For example:

chapter[@author="Niikkonen, Donna"][1]

locates the first <chapter> child of the context node for which the <chapter>'s author attribute has the value "Niikkonen, Donna" (i.e., the first chapter written by Donna Niikkonen).

When using multiple predicates, you must be careful to place them in the right order. The above does not necessarily produce the same result as:

chapter[1][@author="Niikkonen, Donna"]

which always selects the first chapter, then tests whether that was or was not written by Donna Niikkonen and selects or rejects it accordingly.
Using Functions in XPath Expressions

The XPath standard defines functions which can manipulate the content of the source tree in various ways.

This section contains the following topics.

- Introduction to XPath Functions
- Processing Node-Sets by Using Node-Set Functions
- Processing Text Strings by Using String Functions
- Testing an Expression by Using Boolean Functions
- Using Numeric Functions to Perform Math Operations in XPath
- Using Numeric Operators to Perform Math Operations in XPath
- Using XSLT Functions in XPath Expressions
- Sample XML Data File for XPath Functions
Introduction to XPath Functions

A function in XPath, as in most programming languages, is a built-in software routine that performs some operation and returns some value. The returned value effectively replaces the reference (or call) to the function which appears at a given point in the code.

A reference to a function takes the form:

```
functionname(arg1...)
```

where:

- `functionname` is the name of the function;
- the parentheses are required; and
- `arg1...` is a comma-separated list of values, called `arguments`, which are used by the function to perform its work. (Note that not all functions take arguments, and in some cases arguments may or may not be required, depending on the circumstances.)

As an example, the XPath specification defines the `contains()` function. This function takes two arguments, both of them string values. If the first string contains the second string, the function returns the value True; otherwise, it returns False. For more information about `contains()`, see [Processing Text Strings by Using String Functions](#).

Consider the following XML fragment:

```xml
<book>
    <chapter><heading>Help for Lost XPathians</heading></chapter>
    <chapter><heading>Searching and Sorting</heading></chapter>
    <chapter><heading>Where to Find Help</heading></chapter>
</book>
```

and the following XPath location path:

```
//chapter[heading][contains(., "Help")]
```

This location path selects all `<chapter>` elements with a `<heading>` child which
contains the string "Help"—that is, it selects the first and third `<chapter>` elements. The function call, `contains(., "Help")`, is replaced by the value True for `<chapter>` elements which contain the desired text string, and by False for all others.

**Note**  The "." in the call to the `contains()` function here forces the context node to be converted to its string value.
Processing Node-Sets by Using Node-Set Functions

The XPath node-set functions perform operations on (as the name implies) node-sets which have been located at the moment the function call is made.

In the following table, which summarizes these node-set functions, *ns* represents a node-set passed as an argument; *ns?* represents an optional node-set argument; and *obj* represents an object of some arbitrary type, such as node-set or string.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description/Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>last()</code></td>
<td>Returns the number of the last node in the currently selected node-set</td>
</tr>
<tr>
<td><code>position()</code></td>
<td>Returns the number of the current node in the selected node-set. Note: [position()=some number] as an expression's predicate can be abbreviated as [some number]; for example, [position()=3] can be shortened to simply [3].</td>
</tr>
<tr>
<td><code>count(ns)</code></td>
<td>Returns the number of nodes in the node-set passed to the function.</td>
</tr>
<tr>
<td><code>id(obj)</code></td>
<td>Returns the node with the ID-type attribute whose value equals that of <em>obj</em>; if the latter is a string, see the description of the <code>string()</code> function for how the value is converted.</td>
</tr>
<tr>
<td><code>local-name(ns?)</code></td>
<td>Returns the &quot;local name&quot; of the argument. The full name of an element is considered to be its expanded name—that is, including the prefix associated with its namespace, if there is one. The local name is this &quot;full name&quot; with the namespace prefix omitted.</td>
</tr>
<tr>
<td><code>namespace-uri(ns?)</code></td>
<td>Returns the URI associated with the argument's namespace, if there is one.</td>
</tr>
<tr>
<td><code>name(ns?)</code></td>
<td>Returns the &quot;full name&quot; of an element, including its namespace prefix (if any).</td>
</tr>
</tbody>
</table>
last()

This location path:

/sales/region[last()]/units

returns the number of units sold by the last region in the document, or 465.

The last() function is similar to the count() function in that both return the number of nodes in a given node-set. The difference is that last() always operates on the node-set in effect at the time of the function call, while count() requires that a node-set be supplied to it.
**position()**

The location path:

/sales/region[position()=2]/@name

returns the value of the *name* attribute for the second region in the document, or "Southeast".
count(ns)

This XPath expression:

\texttt{count(/sales/region)}

returns the number of \texttt{<region>} elements that are children of the root \texttt{<sales>} element, or 4.

The \texttt{count()} function is similar to the \texttt{last()} function in that both return the number of nodes in a given node-set. The difference is that \texttt{last()} always operates on the node-set in effect at the time of the function call, while \texttt{count()} requires that a node-set be supplied to it.
**id(obj)**

You could determine the amount associated with sales in the Northwest region with the following:

```xml
//*[id("Northwest")]/*/amount
```

This navigates down to the element of any name with an ID-type attribute whose value is "Northwest", and locates the value of the corresponding `<amount>` element, or 12500.26.
The Sample XML Data File for XPath Functions doesn't include any namespace declarations. Even if it did, though, the following XSLT template:

```xml
<xsl:for-each select="//region[2]/*">
    <xsl:value-of select="local-name()"/>
</xsl:for-each>
```

would simply display the local names of all child elements of the second <region> element as a single string, or "unitsamount".

Note that the `local-name()` function always extracts the simple name(s) of a given node-set (the context node, by default), and therefore always works, regardless of the presence or absence of namespace declarations. The `name()` function, on the other hand, always refers to nodes by their fully-qualified names, including any namespace prefixes.
namespace-uri(ns?)

Our sample document doesn't include any namespace declarations. However, consider the case of an XSLT style sheet which begins with the following `<xsl:stylesheet>` element:

```xml
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

If a separate style sheet were applied to this one, containing the following template rule:

```xml
<xsl:template match="/xsl:stylesheet">
  <xsl:value-of select="namespace-uri()"/>
</xsl:template>
```

the result would be the URI in effect for the `<xsl:stylesheet>` element, or "http://www.w3.org/1999/XSL/Transform".
name(ns?)

Our sample document doesn't include any namespace declarations. For this reason, all element names returned by the name() function will be identical to those returned by the local-name() function. However, consider an XSLT style sheet for transforming XML data to HTML; such a style sheet typically opens with this <xsl:stylesheet> element:

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="xmlns:http://www.w3.org/TR/REC-html40">

If you transformed this style sheet using a separate one containing the following template rule:

```xml
<xsl:template match="/xsl:stylesheet">
    <pre>
      <p>
        local-name(): <xsl:value-of select="local-name()"/>
      </p>
      <p>
        name(): <xsl:value-of select="name()"/>
      </p>
    </pre>
</xsl:template>
```

the output, when viewed in Internet Explorer, would be:

```
local-name(): stylesheet
name(): xsl:stylesheet
```

Note that name() returns the full name, including namespace prefix.

See Also

Sample XML Data File for XPath Functions | Processing Text Strings by Using String Functions
Processing Text Strings by Using String Functions

The XPath string functions perform a wide variety of operations on text-string values.

In the following table, which summarizes the string functions, str represents a string passed as an argument; str*, a set of zero or more strings, delimited by commas; obj, an object of some arbitrary type, such as node-set or number; and num, an integer number. The character ? appended to any of these argument types means that the argument is completely optional. If a function has only a single optional argument which is omitted in a given call, the function is assumed to apply to the context node.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description/Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>string(obj?)</td>
<td>Converts the argument to a string value, which is then returned from the function.</td>
</tr>
<tr>
<td>concat(str, str, str*)</td>
<td>Concatenates the various strings passed to it into a single string, which is returned from the function.</td>
</tr>
<tr>
<td>starts-with(str, str)</td>
<td>Returns &quot;true&quot; if the first argument starts with the second, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>contains(str, str)</td>
<td>Returns &quot;true&quot; if the first argument contains the second, otherwise &quot;false&quot;.</td>
</tr>
<tr>
<td>substring(str, num, num?)</td>
<td>Extracts a portion of the first argument, starting with the position supplied by the second argument, for a length of however many characters are in the third argument (if there is one). If the third argument is omitted, the function simply returns all characters in the first argument, starting at the position supplied by the second.</td>
</tr>
<tr>
<td>substring-before(str, str)</td>
<td>Returns the portion of the first argument that precedes the value of the second argument.</td>
</tr>
<tr>
<td>substring-after(str, str)</td>
<td>Returns the portion of the first argument that follows the value of the second argument.</td>
</tr>
<tr>
<td></td>
<td>Returns the number of characters in the argument. If the</td>
</tr>
</tbody>
</table>
string-length(str?)

argument is omitted, returns the number of characters in the current node.

Examines the argument and strips out leading and trailing white space in it; also removes extraneous white space within the argument by replacing two or more occurrences of white space with a single space. The value returned by the function is this "stripped" string.

normalize-space(str?)

Returns the first argument, replacing each occurrence of a character that matches one of the characters in the second argument with the character in the corresponding position in the third argument.
**string(obj?)**

When applied to the root element of our sample document, the following XPath expression:

`string(*)`

returns a single string containing a (space-delimited) list of the entire document's text contents, or "374 12500.26 512 17692 161 8349.72 465 15239.6".
The function \texttt{concat(str, str, str*)}

We could produce a list of the regions in the sample document, including a label which varies by the region's position in the document, using an XSLT template rule such as:

\begin{verbatim}
<xsl:template match="sales">
  <xsl:for-each select="region">
    <h3>
      <xsl:value-of select="concat('Region ', string(position()), ': ', @name)"
    </h3>
  </xsl:for-each>
</xsl:template>
\end{verbatim}

\textbf{Note} The above template rule forces the value of the \texttt{position()} function, normally a number, to be converted to a string. In practice this is not necessary, since numbers are coerced to strings when needed by all these string-handling features. For more information about the \texttt{position()} function, see \textit{Processing Node-Sets by Using Node-Set Functions}.

When viewed in Internet Explorer, this template rule displays:

- **Region 1: Northeast**
- **Region 2: Southeast**
- **Region 3: Southwest**
- **Region 4: Northwest**
**starts-with(str, str)**

This XPath location path:

```xml
//region[starts-with(@name, 'S')]
```

locates all `<region>` elements in the document whose `name` attributes start with the letter "S", or in this case the Southeast and Southwest regions.
contains(str, str)

The location path:

//region[contains(@name, 'east')]

locates all <region> elements in the document whose name attributes contain the string "east", or in this case the Northeast and Southeast regions.
substring(str, num, num?)

We could derive region name *abbreviations* for the four regions in our sample document using the following XSLT template rule:

```xml
<xsl:template match="region">
  <h3><xsl:value-of select="substring(@name, 1, 1)"/>
      <xsl:choose>
          <xsl:when test="contains(@name, 'west')">W</xsl:when>
          <xsl:otherwise>E</xsl:otherwise>
      </xsl:choose></h3>
</xsl:template>
```

The call to the `substring()` function returns the first letter in the `<region>` element's `name` attribute. The `<xsl:choose>` block then displays either a "W" or an "E" depending on whether the name contains the string "west" or not, respectively.

Internet Explorer displays the results of this template rule as:

```
NE
SE
SW
NW
```
substring-before($str, $str)

To display whether a region lies in the northern or southern areas, you could use the substring-before() function as shown in the following sample.

Example

XML File (xpathfuncs.xml)

Change the href attribute in xpathfuncs.xml (shown in Sample XML Data File for XPath Functions) to reference funcsubstringbef.xsl.

XSLT File (funcsubstringbef.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="region">
    <h3>Region "<xsl:value-of select="@name"/>" is in the
        <xsl:choose>
          <xsl:when test="contains(@name, 'west')">Region "<xsl:value-of select="substring-before(@name, 'west')" is in the North
          </xsl:when>
          <xsl:otherwise>Region "<xsl:value-of select="substring-before(@name, 'east')" is in the South
          </xsl:otherwise>
        </xsl:choose>
    </h3>
  </xsl:template>

</xsl:stylesheet>
```

Formatted Output

  Region "Northeast" is in the North
  Region "Southeast" is in the South
  Region "Southwest" is in the South
Region "Northwest" is in the North

Processor Output

<?xml version="1.0" encoding="UTF-16"?><h3>Region "Northeast" is in the North</h3><h3>Region "Southeast" is in the South</h3><h3>Region "Southwest" is in the South</h3><h3>Region "Northwest" is in the North</h3>
substring-after(str, str)

You could invert the example used above, for the substring-before() function, to display whether a region is located in the eastern or western sales districts. To do so, you could use an XSLT template rule such as shown in the sample below.

In this example, unlike the parallel one for substring-before(), the name of the district ("east" or "west") is not capitalized. If you wanted to capitalize the district name, you could use the translate() function for this purpose.

Example

XML File (xpathfuncs.xml)

Change the href attribute in xpathfuncs.xml (shown in Sample XML Data File for XPath Functions) to reference funcsubstringaft.xsl.

XSLT File (funcsubstringaft.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="region">
    <h3>Region "<xsl:value-of select="@name"/" is in the
      <xsl:choose>
        <xsl:when test="contains(@name, 'North')">
          <xsl:value-of select="substring-after(@name, 'North')"
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring-after(@name, 'South')"
        </xsl:otherwise>
      </xsl:choose>ern district
    </h3>
  </xsl:template>

</xsl:stylesheet>
```

Formatted Output

Region "Northeast" is in the eastern district
Region "Southeast" is in the eastern district

Region "Southwest" is in the western district

Region "Northwest" is in the western district

Processor Output

```xml
<?xml version="1.0" encoding="UTF-16"?><h3>Region "Northeast" is in the eastern district</h3><h3>Region "Southeast" is in the eastern district</h3><h3>Region "Southwest" is in the western district</h3><h3>Region "Northwest" is in the western district</h3>
```
**string-length**(str?)

Unlike most XPath string functions, string-length() returns a number instead of a string. This number can be used as-is, or (as with other function references) passed as an argument to some other function.

In our sample XML document, notice that the value of some `<units>` contains a decimal portion, while the others contain integer values. Using this observation and some of the other XPath string functions in conjunction with string-length(), we could select regions which failed to sell an amount of least 10,000 with a template rule as shown in the XSLT sample file below.

First, the template rule defines a variable, `amt_integ`. This variable will contain either the value of the `<amount>` element, or that value less the decimal portion, depending on whether or not the value contains a decimal point. The template rule then tests this resulting "integer amount" to see if its string-length is less than 5—that is, if the value of the `amt_integ` variable is 9999 or less. If so, the region's name and the value of the `<amount>` element are displayed.

In this particular example, you could have used a much simpler and more direct numeric test against the value of the `<amount>` element. For example:

```xml
<xsl:if test="amount &lt; 10000">
```

This direct numeric test would obviate the need to use the various string functions demonstrated in this example.

**Example**

**XML File (xpathfuncs.xml)**

Change the `href` attribute in `xpathfuncs.xml` (shown in Sample XML Data File for XPath Functions) to reference `funcstringlen.xsl`.

**XSLT File (funcstringlen.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
Southwest Region's sales only 8349.72 this quarter.

Processor Output

```xml
<?xml version="1.0" encoding="UTF-16"?><h3>Southwest Region's sales 8349.72 this quarter.</h3>
```
**normalize-space**(str?)

This XPath string function is particularly useful when comparing two strings which possibly contain leading or trailing white space, especially newlines, which you might otherwise overlook in a source document.

For instance, consider the following two `<sample>` elements:

```
<sample>Now is the time...
</sample>
```

and:

```
<sample>Now is the time...</sample>
```

Any template in the following `<xsl:if>` element will *not* be instantiated:

```
<xsl:if test="sample[1]=sample[2]">
```

The trailing newline in the first `<sample>` is part of its content, so its string-value does *not* equal that of the second `<sample>`. For an accurate comparison between the two elements' contents, use this instead:

```
<xsl:if test="normalize-space(sample[1])=normalize-space(sample[2])"
```
translate(str, str, str)

To convert the names of the regions in our sample document to all uppercase, you could use the following:

```xml
translate(//region/@name, "abcdefghijklmnopsqrtuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

The translate() function scans the first argument—the value of the given `<region>` element's name attribute, in this case—looking for any of the individual characters in the second argument. For each matched character, the function substitutes the character from the third argument in the same position as the matched character. So each "a" becomes "A", each "b" becomes "B", and so on. The translated result is returned by the function, giving (in this case) "NORTHEAST", "SOUTHEAST", and so on.

**Note** Any characters which do not appear in the second argument, such as the capital "N" and "S" in the various region names, are retained in the resulting string with no substitution made.

Also note that if the second argument contains more characters than the third, the effect will be to *remove* the missing characters in the returned value. For instance:

```xml
translate("Internet", "nter", "NTE")
```

will substitute a capital "N", "T", and "E" for the corresponding characters in the second string, but replace the lowercase "r" with nothing at all, i.e. remove it from the input string. The result is:

```
INTENET
```

**Example**

**XML File (xpathfuncs.xml)**

Change the `href` attribute in `xpathfuncs.xml` (shown in [Sample XML Data File for XPath Functions](Sample_XML_Data_File_for_XPath_Functions)) to reference `functranslate.xsl`. 
XSLT File (functranslate.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="sales">
    <xsl:value-of select='translate(//region/@name,
        "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZ")'/>
</xsl:template>

</xsl:stylesheet>
```

Formatted Output

NORTHEAST

Processor Output

```xml
<?xml version="1.0" encoding="UTF-16"?>NORTHEAST
```
Testing an Expression by Using Boolean Functions

The XPath Boolean functions return simple true or false values based on the functions' evaluation of arguments (if any) passed to them.

In the following table, which summarizes the Boolean functions, *str* represents a string passed as an argument; *obj*, an object of some arbitrary type, such as node-set or number; and *boolean*, a Boolean condition (with either a true or false value).

<table>
<thead>
<tr>
<th>Format</th>
<th>Description/Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean(<em>obj</em>)</td>
<td>Used primarily to test whether or not something &quot;exists.&quot; If <em>obj</em> is a node-set, the function returns true if and only if the node-set is <em>not</em> empty; if a string, if and only if the string's length is greater than 0; and if a number, if and only if it is non-zero and a valid number. In all other cases it returns false.</td>
</tr>
<tr>
<td>not(<em>boolean</em>)</td>
<td>Returns true if the argument passed to it is false, or false if the argument passed to it is true.</td>
</tr>
<tr>
<td>true()</td>
<td>Simply returns the value true.</td>
</tr>
<tr>
<td>false()</td>
<td>Simply returns the value false.</td>
</tr>
<tr>
<td>lang(<em>str</em>)</td>
<td>Returns true or false, depending on whether or not the context node has the xml:lang value specified in <em>str</em>.</td>
</tr>
</tbody>
</table>

**Note**  
The `true()` and `false()` functions generally make explicit what is otherwise implicit in an XPath expression. This is useful primarily for ensuring that your meaning is clear, as in the following:

```xml
<xsl:if test="(x=y)=true()">
```

which could also be represented, more simply:

```xml
<xsl:if test="x=y">
```

However, these two functions may also be used to assign Boolean values to
XSLT variables. Examples of this use are shown below.
**boolean**(*obj*)

To verify that the value of the `<units>` element is a number before performing some numeric operation on it, you could use the following:

```xml
<xsl:if test="boolean(number(units))">

If the `<units>` element is empty or contains a non-numeric value, the body of the `<xsl:if>` element will not be executed.

**Note**  The preceding is *not* the same as:

```xml
<xsl:if test="boolean(units)">

This simpler test passes the `<units>` element as a *node-set* (not its value as a number) to the function, and thus simply verifies that the context node has at least one `<units>` child.
The following expression determines whether the document has any `<amount>` elements whose values are greater than 20,000:

\[
\text{not(amount [. \text{gt}; 20000])}
\]

If there are any such `<amount>` elements, the function returns `false`, otherwise `true`.

The `not()` function appearing in a predicate or XSLT test can sometimes be used interchangeably with the `!=` Boolean operator. For example:

\[
\text{[not("A"="B")]} \quad \text{and} \quad \text{["A"!="B"]}
\]

are functionally equivalent.

However, when one of the two values being compared is a node-set, using the `not()` function vs. the `!=` operator can produce different results, as shown in the sample below.

Consider the two XSLT `<xsl:if>` elements in the sample XSLT template rule shown below.

The first test outputs "Not March 03" for both invoices. Essentially, it asks the question: "Does the context node have an `invoice_date` attribute, and if it does, does the attribute's value `not` equal '2001-03-04'?"

The second outputs "Not March 03" for invoice #X00456 only, because it asks a slightly different question: "Does the context node `not` have an `invoice_date` attribute which equals '2001-03-04'?" The answer to that question is true for only the `<sale>` element whose invoice number is X00456.

For more information about the `!=` operator, see Testing within a Predicate by Using Boolean Operators.
Example

XML File (funcnot.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="funcnot.xsl" ?>
<sales copyright="2001">
  <sale invoice="X00123" invoice_date='2001-01-05'/>
  <sale invoice="X00456"/>
</sales>
```

XSLT File (funcnot.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="sales">
    <xsl:for-each select="sale">
      <p><b>For invoice #<xsl:value-of select="@invoice"/></b></p>
      <xsl:if test="@invoice_date!='2001-03-04'">
        <p>Test with "!=': Not March 03</p>
      </xsl:if>
      <xsl:if test="not(@invoice_date='2001-03-04')">
        <p>Test with "not()": Not March 03</p>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Formatted Output

For invoice #X00123

Test with "!=': Not March 03

Test with "not()": Not March 03

For invoice #X00456

Test with "not()": Not March 03

Processor Output

The XSLT produces the following stream. Line breaks have been added for
clarity.

<?xml version="1.0" encoding="UTF-16"?>
<p><b>For invoice #X00123</b></p>
<p>Test with "!=": Not March 03</p>
<p>Test with "not()": Not March 03</p>
<p><b>For invoice #X00456</b></p>
<p>Test with "not()": Not March 03</p>
true() and false()

You can define a variable which returns a true or false value depending on some condition in the document being processed, as in the example below.

The explicit test of the string value "true" in the <xsl:if> element above is necessary because the <xsl:value-of> element in the $units_gt_300 variable's definition returns a result tree fragment (RTF), which—regardless of the actual data type of the RTF's value—is always treated as a string. If we hadn't been concerned with demonstrating the use of the true() and false() functions, the variable's definition could be more concisely expressed as:

```xml
<xsl:variable name="units_gt_300" select="number(units) &gt; 300"/>
```

This would have returned a pure Boolean true or false value. So the <xsl:if> element could more simply have been coded:

```xml
<xsl:if test="$units_gt_300">
```

For more information, see Introduction to Result Tree Fragments.

Example

**XML File (xpathfuncs.xml)**

Change the href attribute in xpathfuncs.xml (shown in Sample XML Data File for XPath Functions) to reference the above XSLT file.

**XSLT File (functruefalse.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
```

```xml
<xsl:template match="sales">
  <h2>Regions Selling More than 300 Units:</h2>
  <xsl:for-each select="region">
    <xsl:variable name="units_gt_300">
      <xsl:choose>
        <xsl:when test="number(units) &gt; 300">
          <xsl:value-of select="true()"/>
        </xsl:when>
      </xsl:choose>
    </xsl:variable>
    <xsl:if test="$units_gt_300">
```
<xsl:otherwise>
   <xsl:value-of select="false()"/>
</xsl:otherwise>
</xsl:choose>
</xsl:variable>
<xsl:if test="$units_gt_300='true'">
   <h3><xsl:value-of select="concat(@name, '(', units, ' units')" /></h3>
</xsl:if>
</xsl:for-each>
</xsl:template>

</xsl:stylesheet>

**Formatted Output**

**Regions Selling More than 300 Units:**

Northeast (374 units)

Southeast (512 units)

Northwest (465 units)

**Processor Output**

```xml
<?xml version="1.0" encoding="UTF-16"?>
<h2>Regions Selling More than 300 Units:</h2>
<h3>Northeast (374 units)</h3>
<h3>Southeast (512 units)</h3>
<h3>Northwest (465 units)</h3>
```
**lang(str)**

Consider an XML document such as the following:

```xml
<?xml version='1.0'?>
<dict>
  <translate>
    <word xml:lang="DE">Ma&amp;#xDF;einheiten</word>
    <word xml:lang="EN">units</word>
    <word xml:lang="ES">unidades</word>
    <word xml:lang="FR">unit&amp;#xE9;s</word>
    <word xml:lang="IT">unit&amp;#xE8;</word>
    <word xml:lang="PT">unidades</word>
  </translate>
  <translate>
    <word xml:lang="DE">Verk&amp;#xE4;ufe</word>
    <word xml:lang="EN">sales</word>
    <word xml:lang="ES">ventas</word>
    <word xml:lang="FR">ventes</word>
    <word xml:lang="IT">vendite</word>
    <word xml:lang="PT">vendas</word>
  </translate>
</dict>
```

An XSLT style sheet to demonstrate a general-purpose translation feature might accept two parameters from the user: a word to be translated (held in the parameter called `$xlate_word`), and a two-character code for the language into which the word is to be translated (the parameter `$xlate_lang`). Such a style sheet might contain the template rule shown in the sample below.

The template rule's `<xsl:for-each>` element extracts the `<word>` element which matches the input word. It then displays the following:

- The input word
- The language code which corresponds to the input word
- The input language code
- The translated value

Note that the translated value is derived (in the final `<xsl:select>`) first by locating the parent of the `<word>` element whose value matches the input word, then by locating the child of that parent whose `xml:lang` attribute matches the input language code.
With the word "units" and the language code "DE" passed as parameters into the style sheet, Internet Explorer displays this template rule's results as follows:

Input word is: units
Its language code is: EN

Translating to DE...
Translated value is: Maßeinheiten

Example

XML File (funclang.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="funclang.xsl" ?>
<dict>
  <translate>
    <word xml:lang="DE">Maßeinheiten</word>
    <word xml:lang="EN">units</word>
    <word xml:lang="ES">unidades</word>
    <word xml:lang="FR">unités</word>
    <word xml:lang="IT">unità</word>
    <word xml:lang="PT">unidades</word>
  </translate>
  <translate>
    <word xml:lang="DE">Verkäufe</word>
    <word xml:lang="EN">sales</word>
    <word xml:lang="ES">ventas</word>
    <word xml:lang="FR">ventes</word>
    <word xml:lang="IT">vendite</word>
    <word xml:lang="PT">vendas</word>
  </translate>
</dict>
```

XSLT File (funclang.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- input parameters for testing -->
<xsl:param name="xlate_word">units</xsl:param>
<xsl:param name="xlate_lang">DE</xsl:param>

<xsl:template match="dict/translate">
```

Input word is: units
Its language code is: EN

Translating to DE...
Translated value is: Maßeinheiten

Formatted Output

Processor Output
Using Numeric Functions to Perform Math Operations in XPath

XPath's numeric functions allow you to perform certain basic mathematical operations on given numeric values or node-sets.

Note that simple arithmetic—addition, subtraction, and so on—can be performed using straightforward numeric operations. For more information, see Using Numeric Operators to Perform Math Operations in XPath.

In the following table, which summarizes the numeric functions, obj? is an optional object of some arbitrary type, such as node-set or number, passed into the function; ns, a node-set; and num, an integer number.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description/Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>number(obj?)</td>
<td>Converts the argument to a number and returns the result. If the argument can't be converted, the function returns the string &quot;NaN&quot; (for &quot;not a number&quot;). If no argument is passed, the function operates on the context node.</td>
</tr>
<tr>
<td>sum(ns)</td>
<td>Returns the sum of all nodes in the argument. If you want to simply add numeric values that aren't associated with a node-set, use the + sign as described in the Numeric operations topic.</td>
</tr>
<tr>
<td>floor(num)</td>
<td>Returns the largest integer that is less than or equal to the argument.</td>
</tr>
<tr>
<td>ceiling(num)</td>
<td>Returns the smallest integer that is greater than or equal to the argument.</td>
</tr>
<tr>
<td>round(num)</td>
<td>Rounds the argument's value up or down to the nearest integer. 1.5 always rounds up to 2, and -1.5 always rounds up to -1. Accurate rounding is dependent on hardware and operating-system considerations; you should test a wide range of possible values to be sure that this function returns the results you want in your application.</td>
</tr>
</tbody>
</table>
number(obj?)

The following XPath expression:

\[
\text{number}(/\text{region[@name="Northwest"]}/\text{units})
\]

simply returns the numeric value 465.

On the other hand, this XPath expression:

\[
\text{number}(/\text{region/@name})
\]

returns the string "NaN". This can be useful when you want to perform basic arithmetic operations. If you add two elements' supposedly numeric contents, but one of them is empty or non-numeric, your results will not be what you expected. Thus you could use XSLT logic such as the following to substitute a value of 0 for any such non-numeric values:

```xml
<xsl:template match="/sales">
  <xsl:for-each select="region">
    <xsl:variable name="tax_pct">.07</xsl:variable>
    <xsl:variable name="num_sales">
      <xsl:choose>
        <xsl:when test="string(number(amount))='NaN'">0</xsl:when>
        <xsl:otherwise><xsl:value-of select="amount"/></xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <p>
      Region - <xsl:value-of select="@name"/>:
      <xsl:value-of select="format-number($tax_pct * 100, '#0.00')"/>
      tax on amount of
      <xsl:value-of select="format-number($num_sales, '#,##0.00')"/>
      is: <xsl:value-of select="format-number($num_sales * $tax_pct,'#', '##0.00')"/>
    </p>
  </xsl:for-each>
</xsl:template>
```

**Note** In order to test the result of the \texttt{number()} function call for a value of "NaN," you must convert that result (as in this template rule) to a string, using the \texttt{string()} function. For more information, see Processing Text Strings by Using String Functions.

If you go back to our sample XML document and replace the value 12500.26
(the value of the Northeast region's <amount> element) with either a non-numeric value or with nothing at all, the above template rule displays in Internet Explorer as:

Region - Northeast:
7.00% tax on amount of 0.00 is: 0.00

Region - Southeast:
7.00% tax on amount of 17,692.00 is: 1,238.44

Region - Southwest:
7.00% tax on amount of 8,349.72 is: 584.48

Region - Northwest:
7.00% tax on amount of 15,239.60 is: 1,066.77

Without the use of the number() function to test for "NaN," the lines for the Northeast region look like this:

Region - Northeast:
7.00% tax on amount of NaN is: NaN
To display the sum of all regions' sales amounts and units, you could use an XSLT template rule such as:

```xml
<xsl:template match="sales">
  <h4>Quarter beginning: <xsl:value-of select="@quarter"/></h4>
  <p>Total units sold by all regions:
    <xsl:value-of select="sum(./units)"/>
  </p>
  <p>Total sales amounts, all regions:
    <xsl:value-of select="sum(./amount)"/>
  </p>
</xsl:template>
```

The result, viewed in Internet Explorer, is:

**Quarter beginning: 2001-01**

- Total units sold by all regions: 1512
- Total sales amounts, all regions: 53781.58
**floor(num) and ceiling(num)**

The number of units sold by the four regions in our sample XML document are 374, 512, 161, and 425. If we wanted to construct a bar graph for these figures in some XML application, such as Scalable Vector Graphics, we may not want to set the Y-axis origin at 0, because it would diminish the (fairly slight) differences among the four regions. For example, in this case, we might want the Y-axis origin to 100—that is, the highest multiple of 100 which is less than or equal to our lowest set of values to be graphed.

Each region's "closest-and-lower multiple of 100" value can be obtained using an XPath expression such as the following:

```
floor(.//units div 100) * 100
```

Similarly, the top of the Y axis (for each region) can be established using:

```
ceiling(.//units div 100) * 100
```
**round(num)**

To display the values of the `<amount>` elements rounded to the nearest integer value, use `round()`.

**Example**

This example demonstrates the `round()` function.

**XML File (xpathfunacs.xml)**

Change the `href` attribute in `xpathfunacs.xml` (shown in [Sample XML Data File for XPath Functions](#)) to reference `round.xsl`.

**XSLT File (round.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="sales">
    <table border="1">
        <tr>
            <th>Region</th>
            <th>Approx Sales</th>
        </tr>
        <xsl:for-each select="region">
            <tr>
                <td><xsl:value-of select="@name"/></td>
                <td align="right"><xsl:value-of select="round(amount)"/></td>
            </tr>
        </xsl:for-each>
    </table>
</xsl:template>
</xsl:stylesheet>
```

**Formatted Output**

<table>
<thead>
<tr>
<th>Region</th>
<th>Approx Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Northeast</td>
<td>12500</td>
</tr>
<tr>
<td>Region</td>
<td>Approx Sales</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>Northeast</td>
<td>12500</td>
</tr>
<tr>
<td>Southeast</td>
<td>17692</td>
</tr>
<tr>
<td>Southwest</td>
<td>8350</td>
</tr>
<tr>
<td>Northwest</td>
<td>15240</td>
</tr>
</tbody>
</table>

**Processor Output**

```xml
<?xml version="1.0" encoding="UTF-16"?>
<table border="1"><tr><th>Region</th><th>Approx Sales</th></tr><tr><td>Northeast</td><td align="right">12500</td></tr><tr><td>Southeast</td><td align="right">17692</td></tr><tr><td>Southwest</td><td align="right">8350</td></tr><tr><td>Northwest</td><td align="right">15240</td></tr></table>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Using Numeric Operators to Perform Math Operations in XPath

Simple arithmetic operations, such as addition and subtraction, can be used in XPath expressions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description/Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>num1 + num2</td>
<td>Adds two numbers, returning their sum.</td>
</tr>
<tr>
<td>num1 - num2</td>
<td>Subtracts num2 from num1, returning the difference.</td>
</tr>
<tr>
<td>num1 * num2</td>
<td>Multiplies num1 times num2, returning the product.</td>
</tr>
<tr>
<td>num1 div num2</td>
<td>Divides num1 by num2, returning the quotient. (This is a floating-point division, which means that the value returned may include a fractional part. 10 div 4 equals 2.5, not 2 or 3.)</td>
</tr>
<tr>
<td>num1 mod num2</td>
<td>Returns the modulus—that is, divides num1 by num2, returning the remainder.</td>
</tr>
</tbody>
</table>

**Note** Division cannot be performed using a simple slash, /, because that character is used as a separator between the steps in a location path. You must use the div operator to represent division.
+ operator

Add the Northeast and Southeast's sales volumes using an XPath expression such as:

\texttt{units[./@name="Northeast"] + units[./@name="Southeast"]}
- **operator**

You could calculate the total sales amounts of all regions, *except* the Northwest, using an XPath expression such as the following:

\[
\text{sum(amount) - amount[./@name="Northwest"]}
\]
* operator

To calculate the sales tax on a region's sales amount, multiply the tax rate times the amount. For instance:

\[ \text{amount} \times 0.07 \]

(assuming a seven-percent sales tax).
To calculate and display the average number of units sold across all regions in this quarter, you could use an XSLT template rule such as:

```xml
<xsl:template match="sales">
    Average units sold for all regions:
    <xsl:value-of select="sum(./units) \texttt{div} count(./units)"/>
</xsl:template>
```

The result, displayed in Internet Explorer, is:

Average units sold for all regions: 378
mod operator

The mod operator is commonly used for enabling a style sheet's template rules to perform a given action every $N$ times, such as doing one thing on an even-numbered table row and something else on an odd-numbered row.

Example

This example uses the mod operator to display the sales figures for each region in alternating bold and normal text.

XML File (xpathfuncs.xml)

Change the href attribute in xpathfuncs.xml (shown in Sample XML Data File for XPath Functions) to reference mod.xsl.

XSLT File (mod.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="sales">
    <table border="1">
        <tr>
            <th>Region</th>
            <th>Approx Sales</th>
            <th>Units</th>
        </tr>
        <xsl:for-each select="region">
            <xsl:choose>
                <xsl:when test="position() mod 2 = 0">
                    <tr>
                        <td><b><xsl:value-of select="@name"/></b></td>
                        <td align="right"><b><xsl:value-of select="round(amount)"/></b></td>
                        <td align="right"><b><xsl:value-of select="units"/></b></td>
                    </tr>
                </xsl:when>
            </xsl:choose>
        </xsl:for-each>
    </table>
</xsl:template>
</xsl:stylesheet>
```
## Formatted Output

<table>
<thead>
<tr>
<th>Region</th>
<th>Approx Sales</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Northeast</td>
<td>12500</td>
<td>374</td>
</tr>
<tr>
<td>Southeast</td>
<td>17692</td>
<td>512</td>
</tr>
<tr>
<td>Southwest</td>
<td>8350</td>
<td>161</td>
</tr>
<tr>
<td>Northwest</td>
<td>15240</td>
<td>465</td>
</tr>
</tbody>
</table>

## Processor Output

```xml
<?xml version="1.0" encoding="UTF-16"?><table border="1"><tr><th>Region</th><th>Approx Sales</th><th>Units</th></tr><tr><td>Northeast</td><td align="right">12500</td><td align="right">374</td></tr><tr><td><b>Southeast</b></td><td align="right"><b>17692</b></td><td align="right"><b>512</b></td></tr><tr><td>Southwest</td><td align="right">8350</td><td align="right">161</td></tr><tr><td><b>Northwest</b></td><td align="right"><b>15240</b></td><td align="right"><b>465</b></td></tr></table>```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
# Using XSLT Functions in XPath Expressions

The XPath specification defines many functions. XSLT style sheets have several additional functions available to them. These include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document(obj, ns?)</td>
<td>Allows processing of more than one source document</td>
</tr>
<tr>
<td>key(str, obj)</td>
<td>Retrieves a key assigned by the <code>&lt;xs1:key&gt;</code> element</td>
</tr>
<tr>
<td>format-number(num, str, str?)</td>
<td>Formats a numeric value in any of a variety of ways</td>
</tr>
<tr>
<td>current()</td>
<td>Locates the current node in a node-set</td>
</tr>
<tr>
<td>unparsed-entity-uri(str)</td>
<td>Returns the URI of a specified unparsed entity</td>
</tr>
<tr>
<td>generate-id(ns?)</td>
<td>Assigns a unique identifier to the first node in a given node-set</td>
</tr>
<tr>
<td>system-property(str)</td>
<td>Returns the value of a specified system property, such as the name or version of the XSLT processor</td>
</tr>
</tbody>
</table>

For more information about these functions, including examples, see the following topics in the XSLT Reference:

- [document Function](#)
- [key Function](#)
- [format-number Function](#)
- [current Function](#)
- [unparsed-entity-uri Function](#)
- [generate-id Function](#)
- [system-property Function](#)
Sample XML Data File for XPath Functions

Topics and XPath expressions in this section use the following XML file. Note that the name attributes on the <region> elements are declared in the DTD to be of type ID.

To use an XSLT file to display this XML file in Internet Explorer, insert the XSLT filename in the href attribute.

**XML File (xpathfuncs.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="myfile.xsl" ?>
<!DOCTYPE sales [ 
<!ELEMENT sales (region*)> 
<!ATTLIST sales quarter CDATA #IMPLIED> 
<!ELEMENT region (units, amount)> 
<!ATTLIST region name ID #IMPLIED> 
<!ELEMENT units (#PCDATA) > 
<!ELEMENT amount (#PCDATA) > ]>
<sales quarter="2001-01">
  <region name="Northeast">
    <units>374</units>
    <amount>12500.26</amount>
  </region>
  <region name="Southeast">
    <units>512</units>
    <amount>17692</amount>
  </region>
  <region name="Southwest">
    <units>161</units>
    <amount>8349.72</amount>
  </region>
  <region name="Northwest">
    <units>465</units>
    <amount>15239.6</amount>
  </region>
</sales>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Useful Patterns for XPath Expressions

This section contains the following topics.

- Useful Patterns for XPath Wildcards and Axes
- Useful Boolean Predicates in XPath Expressions
- Common Use of Predicate Filtering in XPath Expressions
Useful Patterns for XPath Wildcards and Axes

Try to select an efficient approach. The use of axes and wildcard characters can radically simplify your code, but that come at the expense of decreased efficiency. For example, the pattern "//*" will match every node in the tree, which is useful if you're trying to search for a given element or attribute but which can be disastrous if your XML structure has 100,000 nodes.
//* Linearizes Tree

The //* pattern will put all of the nodes in a tree into a single node-set in regular traversal order.
//* Is Least Efficient

Using universal match for searches is the least efficient way of finding nodes, as it traverses every point in the tree.
<!--[@attrName] Gets Elements With a Specific Attribute

The \textit{id()} function has been removed from the XSLT specification, but you can search for all elements with given attributes with the XPath expression //[@id='myID'].
ancestor-or-self::* Gets Ancestor List

This pattern can be used to retrieve a list of all direct ancestors of a given node, useful for identifying paths.
Use Axes and Wildcards for Recursion

Both Axes and Wildcards are perfect for making recursive calls.
following-sibling::* is a node set

This pattern returns ALL sibling nodes after the current node. To get the next node, use:

following::sibling::*[1]
Useful Boolean Predicates in XPath Expressions

A predicate expression (that is, an XPath expression within square brackets) takes the current context and applies a test to it. If the test fails for the context, then it is eliminated from the node-set that the context belongs to, otherwise it stays in. You can use this technique for performing very fine grain analysis of the XML tree.
Predicates return Booleans

For each node, predicates return either true or false, indicating that the given node either satisfies or doesn't satisfy the predicate. Numeric expressions will always evaluate to true unless the result is 0, while an empty string is considered to be a false value within a predicate.
Use OR for conjoining: [a or b]

will match either a or b.
Use AND for filtering: [a and b]

will match only if both a and b nodes exist.
Multiple predicates

You can have a predicate filter of the form path[A][B], or path1[A]/path2[B]. XPath expressions work from left to right, so that with the expression path1[A]/path2[B], path1[A] will be determined first and only those elements that remain have path2[B] applied.
AND is faster than sequential predicates

The MSXML parser can handle Boolean evaluations faster than it can node-set creation or reduction. As a consequence, while both path[A][B] and path[A and B] will generate the same results, the use of 'and' will improve the performance of your XPath query significantly.
AND short-circuits expressions

One reason for the efficiency when using AND is that if the first part of an intersection (an AND clause) is false, the second part will not be evaluated. As a consequence, when building AND XPath expressions, place the most efficient clause first to cut down on the number of searches, especially if one clause is especially complex. For example, the following XPath expression selects all elements that have the same name as the parameter $price.type before applying the filtering algorithm to make sure the price is in between the low and high price:

```xml
//*[name(.)=$price.type and (number(.) > $price.low and number(.) < $price.high)]
```
Common Use of Predicate Filtering in XPath Expressions

There are a number of predicate functions, but you can use a few fairly simple XPath expressions to get surprisingly sophisticated results.

The following expression matches a given element by name without having to specify the name explicitly. This is useful for parameterization.

```
*[name(.)='myName'] or *[name(.)=$myNameVar]
```

The following expression returns even elements only. This is useful for creating alternating schemes.

```
*[position() mod 2 = 0]
```

The following expression matches all elements in the tree for which the id attribute corresponds to the idref attribute of the current node. This is useful when you have multiple types of information in an XML document (such as employees and offices), and need to relate one to the other.

```
//*[id=./idref]
```
How Do I?

The following tutorials demonstrate common XPath tasks.

- Work with Lists of Items Using XPath
- Use XPath Axes to Navigate Through XML Data
- Construct Complex XPath Searches
Work with Lists of Items Using XPath

This section covers the following:

- [Placing a List of Items into a Table](#)
- [Creating a Comma-Separated List of Items](#)
Placing a List of Items into a Table

A common transformation is to place a list of items into a grid or table. Given a list of product names, you might want to present them two-across in a table, as follows.

product 1 product 2
product 3 product 4

Items must be grouped in pairs into table rows. XSLT currently does not have a built-in mechanism to handle arbitrary grouping. A combination of the `<xsl:if>` element with the `test` attribute and the `following-sibling` axis can be used to approximate this behavior. The following template turns a list of "product" elements into a two-column table.

```
<TABLE BORDER="1">
  <xsl:for-each select="products/product">
    <xsl:if test="(position() mod 2) = 1">
      <TR>
        <TD><xsl:value-of select="."/></TD>
        <TD><xsl:value-of select="following-sibling::product[position()=1]"/></TD>
      </TR>
    </xsl:if>
  </xsl:for-each>
</TABLE>
```

Repeat through each product element, but use the `<xsl:if>` element with the expression "position() mod 2 = 1" to isolate only those products that should start a new row, every other product. The name of this product also is placed in the first cell in the row. Then make another cell and place within it the next child — the one filtered out using the `<xsl:if>`. The query to select the next element navigates to the parent and obtains a set of following siblings.

Three-column tables can be created by adding another cell to the row to obtain another sibling element, and adjusting the expression to allow only every third element to create a new row. You can extrapolate this mechanism to create tables with even more columns.

```
<TABLE BORDER="1">
```
Example

This example demonstrates the table code snippets that are shown above.

XML File (product-list.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="product-list.xsl"?>
<products>
  <product>ActiMates Arthur</product>
  <product>ActiMates Barney</product>
  <product>ActiMates DW</product>
  <product>ActiMates PC Pack</product>
  <product>ActiMates TV Pack</product>
  <product>Arthur's Brainteasers</product>
  <product>Arthur's Math Carnival</product>
  <product>Arthur's Reading Roundup</product>
  <product>Barney fun on the Farm</product>
  <product>Barney goes to the Circus</product>
  <product>Barney under the Sea</product>
</products>
```

XSLT File (product-list.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <HTML>
    <BODY STYLE="font:bold 10pt Arial">
      <DIV>Two column table:</DIV>
      <TABLE BORDER="1">
        <xsl:for-each select="products/product">
          <xsl:if test="(position() mod 3) = 1">
            <TR>
              <TD><xsl:value-of select="."/></TD>
              <TD><xsl:value-of select="following-sibling::product[position()=1]"/></TD>
              <TD><xsl:value-of select="following-sibling::product[position()=2]"/></TD>
            </TR>
          </xsl:if>
        </xsl:for-each>
      </TABLE>
    </BODY>
  </HTML>
</xsl:template>
```

Three column table:

<table>
<thead>
<tr>
<th>ActiMates Arthur</th>
<th>ActiMates Barney</th>
<th>ActiMates DW</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiMates DW</td>
<td>ActiMates PC Pack</td>
<td></td>
</tr>
<tr>
<td>ActiMates TV Pack</td>
<td>Arthur's Brainteasers</td>
<td></td>
</tr>
<tr>
<td>Arthur's Math Carnival</td>
<td>Arthur's Reading Roundup</td>
<td></td>
</tr>
<tr>
<td>Barney fun on the Farm</td>
<td>Barney goes to the Circus</td>
<td></td>
</tr>
<tr>
<td>Barney under the Sea</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Processor Output
A portion of the output stream is shown here.

<HTML>
<BODY STYLE="font:bold 10pt Arial">
<DIV>Two column table:</DIV>
<TABLE BORDER="1">
<TR>
<TD>ActiMates Arthur</TD>
<TD>ActiMates Barney</TD>
</TR>
<TR>
<TD>ActiMates DW</TD>
<TD>ActiMates PC Pack</TD>
</TR>
...
</TABLE><BR>
<DIV>Three column table:</DIV>
<TABLE BORDER="1">
<TR>
<TD>ActiMates Arthur</TD>
<TD>ActiMates Barney</TD>
<TD>ActiMates DW</TD>
</TR>
<TR>
<TD>ActiMates PC Pack</TD>
...
</TR>
</TABLE>
</BODY>
</HTML>
Creating a Comma-Separated List of Items

The `last()` function can be used to determine if the element is the last one in the query.

```xml
<DIV>
  <xsl:for-each select="products/product">
    <xsl:value-of select="."/>
    <xsl:if test="position() != last()">,
    </xsl:if>
  </xsl:for-each>
</DIV>
```

Because the comma is being inserted based on the position in the query and not the source document, the list can be sorted without creating errors in the results. The following template shows how to add commas to a sorted product list.

```xml
<DIV>
  <xsl:for-each select="products/product">
    <xsl:sort select="product" order="descending"/>
    <xsl:value-of select="."/>
    <xsl:if test="position() != last()">,
    </xsl:if>
  </xsl:for-each>
</DIV>
```

The `<xsl:sort>` `order` attribute is given the value "descending" to indicate a descending sort by product name. The preceding two code snippets are shown in the following example.

You can separate the comma logic from outputting the element value. In the following template rule, the names in a group of names are formatted as a comma-separated list.

```xml
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test="position() != last()">,
  </xsl:if>
</xsl:template>
```

Another way to separate the comma and node printing is by reversing the logic to verify whether this name is the first. In some cases, this performs better than the preceding example because `last()` requires that the entire set of names be
found and counted, while this one does not.

```xml
<xsl:template match="namelist/name">
    <xsl:if test="position() != 1">', '</xsl:if>
    <xsl:apply-templates/>
</xsl:template>
```

The preceding two snippets are used with another template rule to display the name value itself:

```xml
<xsl:template match="name">
    <xsl:value-of select="."/>
</xsl:template>
```

Example

**XML File (product-list2.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="product-list2.xsl"?>
<products>
    <product>ActiMates Arthur</product>
    <product>ActiMates Barney</product>
    <product>ActiMates DW</product>
    <product>ActiMates PC Pack</product>
    <product>ActiMates TV Pack</product>
    <product>Arthur's Brainteasers</product>
    <product>Arthur's Math Carnival</product>
    <product>Arthur's Reading Roundup</product>
    <product>Barney fun on the Farm</product>
    <product>Barney goes to the Circus</product>
    <product>Barney under the Sea</product>
</products>
```

**XSLT File (product-list2.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <HTML>
            <BODY STYLE="font:10pt Arial">
                <DIV>B>Comma separated list:</B></DIV>
                <DIV>
                    <xsl:for-each select="products/product">
```
Comma separated list:


Comma separated and reverse-sorted list:


Processor Output

<html>
<body style="font:10pt Arial">
<div><b>Comma separated list:</b></div>


<div><b>Comma separated and reverse-sorted list:</b></div>

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Use XPath Axes to Navigate through XML Data

The following topics describe each XPath navigation axis. Each topic presents an Extensible Stylesheet Language Transformation (XSLT) file that works with the Sample XML File for Navigating XPath Axes. The XSLT template rules are also presented as a long file, Sample XSLT File for Navigating XPath Axes.

- Navigating Along the child Axis
- Navigating Along the descendant Axis
- Navigating Along the parent Axis
- Navigating Along the ancestor Axis
- Navigating Along the following Axis
- Navigating Along the preceding Axis
- Navigating Along the following-sibling Axis
- Navigating Along the preceding-sibling Axis
- Navigating Along the attribute Axis
- Navigating Along the namespace Axis
- Navigating Along the self Axis
- Navigating Along the descendant-or-self Axis
- Navigating Along the ancestor-or-self Axis
- Using Abbreviations in XPath Expressions

See Also

Constructing the Node-Test Portion of a Location Step | Determining the Context Node
Navigating Along the child Axis

The child axis selects nodes that are immediate children of the context node. For example, if the context node is the root `<chairman>` element:

```xml
child::president
```

locates a node set consisting of two `<president>` elements:

- `<president empID="emp4390" emppdate="1980-06-18">`, and
- `<president empID="emp4238" emppdate="1984-08-12">`

Assuming the same context node, changing the location step to:

```xml
child::president | child::name
```

will locate a node-set consisting of the above two elements and the `<name>Kim Akers</name>` element.

Only the root node and element nodes in a document have nodes along the child::* axis. This means that, comments, attributes, processing instructions (PIs), and namespaces do not have children. Similarly, only element nodes may be located along the child::* axis if you use a name node test, such as a specific element name, or * for all elements. If you need to select comments and PIs as well, use the node() or other node-type test. For example:

```xml
child::node()
```

locates all comments and PIs which are children of the context node, as well as all element children, while:

```xml
child::comment()
```

locates all comments which are children of the context node, while excluding all PI and element nodes.

The child::* axis is the default, if no axis is supplied in a location step. Therefore, the following are equivalent:
child::name
name

See Also

Sample XML File for Navigating XPath Axes | Constructing the Node-Test Portion of a Location Step
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Navigating Along the descendant Axis

The descendant:: axis of a given context node locates all element nodes which the context node contains, that is, all children, all the children's children, and so on. The location step:

/descendant::*

locates all elements descended from the root node, and therefore locates all elements in a given document.

Since the descendant:: axis extends the child:: axis recursively through the context node's tree, the same restrictions apply as for the child:: axis. In particular, using a name node test, specific element names or the * for all elements, do not select comments and PIs descended from the context node. If you need to select comments and PIs as well as elements, use:

descendant::node()

attributes and namespaces can never be located on the child:: axis, and therefore cannot be located on the descendant:: axis.

See Also

Sample XML File for Navigating XPath Axes | Navigating Along the child Axis
Navigating Along the parent Axis

The parent:: axis is the inverse of the child:: axis. It selects the node in the document tree that is immediately above the context node, in hierarchical order.

All nodes except the root node have one and only one parent node; the root node has no parent. The parent of a namespace or attribute node is the element node declaring it. The parent of a comment or PI is the element that contains it, or, if the comment or PI precedes or follows the root element, the parent is the root node.

In the XSLT template rule shown below, the match pattern for the template rule sets the context node to a particular <director> element, the one whose <name> element equals "Michelle Votava". The template portion of the rule builds a table with several rows: one for column headings, and one for each attribute of the requested <director> element. For each such attribute, the template constructs a table cell for the name of the attribute, its value, and using the parent:: axis, the name and value of the attribute's parent.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-parent.xsl.

XSLT File (orgchart-parent.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- suppress text nodes not covered in subsequent template rule: -->
  <xsl:template match="text()"/>

  <xsl:template match="director[name='Michelle Votava']">
    <table border="1">
      <tr>
        <th>Attribute name</th>
        <th>Attribute value</th>
        <th>Parent name</th>
      </tr>
    </table>
  </xsl:template>
</xsl:stylesheet>
```
<th>Parent value</th>
</tr>
<xsl:for-each select="attribute::*">
  <tr>
    <td><xsl:value-of select="name()"/></td>
    <td><xsl:value-of select="."/></td>
    <td><xsl:value-of select="name(parent::*)"/></td>
    <td><xsl:value-of select="parent::*"/></td>
  </tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>

Formatted Output

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Attribute value</th>
<th>Parent name</th>
<th>Parent value</th>
</tr>
</thead>
<tbody>
<tr>
<td>empID</td>
<td>emp2348</td>
<td>director</td>
<td>Michelle Votava Southeast</td>
</tr>
<tr>
<td>empdate</td>
<td>1990-06-02</td>
<td>director</td>
<td>Michelle Votava Southeast</td>
</tr>
</tbody>
</table>

Note  The parent of an attribute is considered to be the element in whose start tag the attribute is assigned a value.

See Also

Navigating Along the child Axis
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Navigating Along the ancestor Axis

The ancestor:: axis locates all nodes in the document hierarchy above the context node. It locates the parent of the context node, the parent's parent, and so on up to the root node. The root node is an ancestor of all other nodes in the document, and itself has no ancestors.

Because ancestor:: extends the parent:: axis all the way back through the document tree, all limitations and effects of the parent:: axis apply to the ancestor:: axis as well.

If you wanted to show the names of all employees along the ancestor axis of the employee whose name is Peter Porzuczek—that is, all of Peter Porzuczek's supervisors—you could use the sample XSLT template rule below.

The ancestors are retrieved in reverse document order. That is, the ancestor closest to context node is displayed first, followed by that ancestor's parent, followed by its parent's parent.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-ancestor.xsl.

XSLT File (orgchart-ancestor.xsl)

```xml
<?xml version='1.0'?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
!--- suppress text nodes not covered in subsequent template rule -->
<xsl:template match="text()"/>

<xsl:template match="*[name='Peter Porzuczek']">
  <h2>Peter Porzuczek's superiors:</h2>
  <xsl:for-each select="ancestor::*">
    <h4><xsl:value-of select="name"/></h4>
  </xsl:for-each>
</xsl:template>
```
Formatted Output

Peter Porzuczek's superiors:

Amy Egert

Katie McAskill-White

Neil Charney

See Also

Navigating Along the parent Axis
Navigating Along the following Axis

All nodes that follow the context node, in document order, lie along the following:: axis. This specifically excludes attribute and namespace nodes, as well as nodes that are descendants of the context node.

Given the sample XML document of Lucerne Publishing's organization chart, you could locate, all employees that follow the employee named Katie McAskill-White with the XSLT template rule shown below.

Because the context node established by the template rule's match pattern is the <name> element whose string value is Katie McAskill-White (match="name[.='Katie McAskill-White']"), the <xsl:for-each> element's select pattern, "following::*/name", locates the names of all employees that follow the context node, in document order.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-follow.xsl.

XSLT File (orgchart-follow.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="name[.= 'Katie McAskill-White']">  
    <h2>Katie McAskill-White's followers:</h2>
    <xsl:for-each select="following::*/name">
      <h4><xsl:value-of select="."/></h4>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Formatted Output
Katie McAskill-White's followers:

Neil Charney
Beth Silverberg
Lani Ota
Peter Ponzuczek
Marea Angela Castaneda
Navigating Along the preceding Axis

The preceding:: axis is the inverse of the following:: axis. It locates all nodes that come before the context node in document order. Ancestors, attribute nodes, and namespace nodes cannot be located along the context node's preceding:: axis.

In the sample organization chart document, if the context node is the <name> element containing "John Tippett," you can locate and display all preceding element nodes using the template rule shown below. This template rule creates a table in the result tree. Aside from the table headings, each row consists of the name of a preceding node, and that node's value. If the node name is "name", the row is colored yellow, otherwise white.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-prec.xsl.

XSLT File (orgchart-prec.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="name[.='John Tippett']">
    <table border="1">
      <tr>
        <th>Node name</th>
        <th>Node value</th>
      </tr>
      <xsl:for-each select="preceding::*">
        <xsl:variable name="bgcolor">
          <xsl:choose>
            <xsl:when test="name()='name'">yellow</xsl:when>
            <xsl:otherwise>white</xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
        <tr bgcolor="{$bgcolor}'">
          <xsl:if test="name()='name'">
            <th>Node name</th>
          </xsl:if>
          <xsl:if test="name()!='name'">
            <th>Node value</th>
          </xsl:if>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```
### Formatted Output

<table>
<thead>
<tr>
<th>Node name</th>
<th>Node value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Amy Egert</td>
</tr>
<tr>
<td>name</td>
<td>Steve Masters</td>
</tr>
<tr>
<td>division</td>
<td>Domestic</td>
</tr>
<tr>
<td>name</td>
<td>Shelly Szymanski</td>
</tr>
<tr>
<td>department</td>
<td>Sales</td>
</tr>
<tr>
<td>director</td>
<td>Cindy Durkin Northeast</td>
</tr>
<tr>
<td>name</td>
<td>Cindy Durkin</td>
</tr>
<tr>
<td>region</td>
<td>Northeast</td>
</tr>
<tr>
<td>director</td>
<td>Michelle Votava Southeast</td>
</tr>
<tr>
<td>name</td>
<td>Michelle Votava</td>
</tr>
<tr>
<td>region</td>
<td>Southeast</td>
</tr>
</tbody>
</table>

### See Also

Navigating Along the following Axis
Navigating Along the following-sibling Axis

The following-sibling:: XPath axis locates all nodes, elements only, by default which share the same parent as the context node itself, and which appear in their entirety after the context node, in document order. Since the root node has no parent, it also has no following-siblings. If the context node is an attribute or namespace node, the following-sibling:: node locates an empty node-set.

In the sample XML document, each <name> element is followed in the document tree by information about the given employee’s organization, including employees at the same level of the hierarchy. This information can be captured by the XSLT template rule for the employee named Shelly Szymanski, shown in the sample XSLT file.

The template rule in the sample XSLT file contains two references to the following-sibling:: axis. The first, used as the value of the <xsl:if> element's test attribute, ensures that the template will be created in the result tree only if the selected <name> element has following-sibling nodes.

This example also demonstrates that the string-value of an element node is a combination of all text nodes contained by that element. In our sample document, the <director> elements do not contain any text nodes of their own. The only text nodes belong to the <name> and <region> elements subordinate to a given <director> element. Thus, the string-value of each <director> element is the combined string-value of all subordinate <name> and <region> elements.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-followsib.xsl.

XSLT File (orgchart-followsib.xsl)
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="name[.= 'Shelly Szymanski']">
    <xsl:if test="following-sibling::*">
      <h2>following-siblings of &lt;name&gt; 'Shelly Szymanski'&lt;/name&gt;'</h2>
      <table border="1">
        <tr>
          <th>Node name</th>
          <th>Node value</th>
        </tr>
        <xsl:for-each select="following-sibling::*">
          <tr>
            <td><xsl:value-of select="name()"/></td>
            <td><xsl:value-of select="."/></td>
          </tr>
        </xsl:for-each>
      </table>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>

Formatted Output

**following-siblings of** `<name>` 'Shelly Szymanski'

<table>
<thead>
<tr>
<th>Node name</th>
<th>Node value</th>
</tr>
</thead>
<tbody>
<tr>
<td>department</td>
<td>Sales</td>
</tr>
<tr>
<td>director</td>
<td>Cindy Durkin Northeast</td>
</tr>
<tr>
<td>director</td>
<td>Michelle Votava Southeast</td>
</tr>
<tr>
<td>director</td>
<td>John Tippett Southwest</td>
</tr>
<tr>
<td>director</td>
<td>Alan Steiner Northwest</td>
</tr>
</tbody>
</table>
Navigating Along the preceding-sibling Axis

Similar to the following-sibling:: axis, the preceding-sibling:: axis locates all nodes, elements, by default, which share the same parent node and which appear before the context node, in document order. Since the root node has no parent, it also has no preceding-siblings. If the context node is an attribute or namespace node, the preceding-sibling:: node locates an empty node-set.

You can override the default elements only selection of preceding-siblings by using the node() node test in the location step. For example, you can locate all of the root <chairman> element's preceding-sibling nodes of type element, node, or processing instruction (PI) using an XSLT template such as the one in the sample XSLT file.

The template rule in the sample XSLT file contains two references to the preceding-sibling:: axis. The first appears as the value of the <xsl:if> element's test attribute. It ensures that the template is instantiated in the result tree only if the context node has at least one node along the preceding-sibling:: axis.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-presibl.xsl.

XSLT File (orgchart-presibl.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
<!-- suppress text nodes not covered in subsequent template rule -->
<xsl:template match="text()"/>

<xsl:template match="chairman">
    <xsl:if test="preceding-sibling::node()">
```
<h2>preceding-siblings of 'Amy Egert'</h2>

<table>
<thead>
<tr>
<th>Node name</th>
<th>Node value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml:stylesheet</td>
<td>type=&quot;text/xsl&quot; href=&quot;/org:hart-persib/xsl&quot;</td>
</tr>
</tbody>
</table>

See Also

Navigating Along the following-sibling Axis
Navigating Along the attribute Axis

When the context node is an element, use the attribute:: axis to locate that element's attributes. The attribute:: axis can be used only with element-type context nodes, and it locates attributes only.

To display a table of all employees' IDs and dates of employment, you could use three XSLT template rules, one that employs the attribute:: axis. This is shown in the XSLT sample file below.

The first template rule sets up the body of the result tree, including a table and a row of table headers. The second template rule suppresses the output of all element and text nodes, note the empty <xsl:template> element.

The third template rule makes an exception for the second, which is to process all <name> elements. The template placed into the result tree by this rule consists of a table row with four cells:

- The value of this <name> element
- The value of the parent element's empID attribute
- The value of the parent element's empdate attribute

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-attr.xsl.

XSLT File (orgchart-attr.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
</xsl:stylesheet>
```
Formatted Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Emp ID</th>
<th>Emp Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy Eger</td>
<td>emp3456</td>
<td>1976-03-21</td>
</tr>
<tr>
<td>Steve Masters</td>
<td>emp4290</td>
<td>1980-06-18</td>
</tr>
<tr>
<td>Shelly Szymanski</td>
<td>emp9301</td>
<td>1981-09-01</td>
</tr>
<tr>
<td>Cindy Dukin</td>
<td>emp5443</td>
<td>1993-04-19</td>
</tr>
<tr>
<td>Michelle Votava</td>
<td>emp2348</td>
<td>1990-06-02</td>
</tr>
<tr>
<td>John Tippett</td>
<td>emp4322</td>
<td>1995-05-26</td>
</tr>
<tr>
<td>Alan Steiner</td>
<td>emp4537</td>
<td>1990-12-20</td>
</tr>
<tr>
<td>John Thosson</td>
<td>emp4320</td>
<td>1978-11-09</td>
</tr>
<tr>
<td>Josh Barnhill</td>
<td>emp3961</td>
<td>1990-01-01</td>
</tr>
<tr>
<td>Kate McAnhill-White</td>
<td>emp4238</td>
<td>1984-08-12</td>
</tr>
<tr>
<td>Neil Charney</td>
<td>emp7216</td>
<td>1987-04-13</td>
</tr>
<tr>
<td>Beth Silverberg</td>
<td>emp9032</td>
<td>1982-08-31</td>
</tr>
<tr>
<td>Ian Osa</td>
<td>emp7634</td>
<td>1994-02-12</td>
</tr>
<tr>
<td>Peter Forzuczek</td>
<td>emp9332</td>
<td>1991-10-03</td>
</tr>
<tr>
<td>Maria Angela Castaneda</td>
<td>emp3832</td>
<td>1991-05-15</td>
</tr>
</tbody>
</table>
Navigating Along the namespace Axis

When the context node is an element, the namespace:: axis will locate all namespace nodes which apply to that element. The namespace:: axis can be used only with elements, and locates only namespace nodes.

A typical XSLT style sheet for use with Microsoft Internet Explorer declares two namespaces: one for elements in the XSLT namespace itself, and one for elements in the HTML 4.0 namespace. Although our sample orgchart.xml document contains no namespace declarations, the following XSLT style sheet, namespace.xsl, can be used with any source XML document which does contain them.

Note The <?xml-stylesheet?> processing instruction refers to this style sheet itself. Therefore you can even open this style sheet directly in Internet Explorer.

The namespace.xsl style sheet contains three template rules:

- The template rule <xsl:template match="/"/> sets up basic HTML elements in the result tree.
- The template rule <xsl:template match="text()"/> suppresses all text nodes.
- The template rule <xsl:template match="*"/> matches all element nodes. For each element:
  - A heading is instantiated in the result tree, Namespaces in effect for <elementname> element:
  - A table is created consisting of two rows. Row 1 is the table column headings: Local name, Namespace URI, and Namespace node(s). Row 2 shows, the value of the local-name() function for the context node; the value of the namespace-uri() function for the context node; and the value(s) of any namespace node(s) in scope for the context node, each on a separate line.

Example
Regardless of the namespace URI, which applies to a given element name (shown in the second column), each element has a namespace node (third column, the nodes along the namespace:: axis) for every namespace that it or any of its ancestors declares.

**XML File (nsdemo.xml)**

```xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="namespace.xsl" ?>
<nsdemo
 xmlns:ns="http://uri.for.ns-prefix"
 xmlns="http://uri.for.no-prefix">
 <branch1>
   <ns:sub_branch1.1/>
   <sub_branch1.2/>
 </branch1>
 <branch2 xmlns:ns2="http://uri.for.ns2-prefix">
   <ns:sub_branch2.1/>
   <sub_branch2.2/>
   <ns2:sub_branch2.3/>
 </branch2>
</nsdemo>
```

**XSLT File (namespace.xsl)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="namespace.xsl" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/TR/REC-html40">
 <xsl:template match="/">
   <html>
     <head><title>Namespaces in XSLT Style Sheet</title></head>
     <body>
       <xsl:apply-templates />
     </body>
   </html>
 </xsl:template>

<xsl:template match="text()"/>

<xsl:template match="*">
  <h3>Namespaces in effect for &lt;xsl:value-of select="name(</h3>
  <table border="1">
    <tr>
      <th>Local-name</th>
      </tr>
  </table>
</xsl:template>
```
<table>
<thead>
<tr>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Namespaces in effect for `<nsdemo>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>

Namespaces in effect for `<branch1>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>

Namespaces in effect for `<sub_branch1.1>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>

Namespaces in effect for `<sub_branch1.2>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>

Namespaces in effect for `<branch2>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>

Namespaces in effect for `<sub_branch2.1>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>

### Namespaces in effect for `<sub_branch2.2>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>
| sub_branch2.2 | http://uri.for.no-prefix | http://www.w3.org/XML/1998/namespace  
| | | http://uri.for.ns-prefix  
| | | http://uri.for.no-prefix  
| | | http://uri.for.ns2-prefix |

### Namespaces in effect for `<ns2:sub_branch2.3>` element:

<table>
<thead>
<tr>
<th>Local-name</th>
<th>Namespace-uri</th>
<th>Namespace node(s)</th>
</tr>
</thead>
</table>
| | | http://uri.for.ns-prefix  
| | | http://uri.for.no-prefix  
| | | http://uri.for.ns2-prefix |

### See Also

[Sample XML File for Navigating XPath Axes](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Navigating Along the self Axis

The self:: axis always locates the context node. It is often used to retrieve the value of the current node, particularly as the select attribute of the <xsl:value-of> element.

To display the name of each employee in the international division identified in the sample XML document as a level-2 heading, you could use a simple XSL template rule such as the XSLT file shown below.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-self.xsl.

XSLT File (orgchart-self.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
</xsl:template match="text()"/>

<xsl:template match="president[division='International']">
    <xsl:for-each select="descendant::name">
        <h2><xsl:value-of select="self::*"/></h2>
    </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

Formatted Output

Katie McAskill-White

Neil Charney

Beth Silverberg
Lani Ota

Peter Porzuczek

Marea Angela Castaneda
Navigating Along the descendant-or-self Axis

The descendant-or-self:: axis, like the descendant:: axis, locates all nodes which are descended from the context node, its children, its children's children, and so on. Unlike descendant::, however, descendant-or-self:: also selects the context node itself.

Attribute and namespace nodes cannot be located using the descendant-or-self:: axis. Since attribute and namespace nodes have no descendants, using descendant-or-self:: when the context node is an attribute or namespace node always returns an empty node-set. To locate descendant comment and processing instruction (PI) nodes as well as elements, use the node(), comment(), or processing-instruction() node-type test with this axis.

To display information about the employee named Steve Masters and the employees whose elements descend from his in orgchart.xml, you can use an XSLT template rule such as orgchart-descself.xsl, below.

Once the template rule's match pattern ("name[.='Steve Masters']") has established the branch of the family tree to be processed, the template rule sets up a level-2 heading and a table for each <name> element node along the descendant-or-self:: axis. Each row of the table, other than column headers, contains the value of that <name> element, and the value of its parent's empID and empdate attributes.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-descself.xsl.

XSLT File (orgchart-descself.xsl)

<?xml version='1.0'?>
Steve Masters and descendant elements:

<table>
<thead>
<tr>
<th>Name</th>
<th>Emp ID</th>
<th>Empl Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steve Masters</td>
<td>emp4390</td>
<td>1980-06-18</td>
</tr>
<tr>
<td>Shelly Szymanski</td>
<td>emp9801</td>
<td>1981-09-01</td>
</tr>
<tr>
<td>Cindy Durkin</td>
<td>emp5443</td>
<td>1993-04-10</td>
</tr>
<tr>
<td>Michelle Votava</td>
<td>emp2348</td>
<td>1990-06-02</td>
</tr>
<tr>
<td>John Tippet</td>
<td>emp4322</td>
<td>1995-05-26</td>
</tr>
<tr>
<td>Alan Steiner</td>
<td>emp4387</td>
<td>1990-12-20</td>
</tr>
<tr>
<td>John Thosson</td>
<td>emp4320</td>
<td>1978-11-09</td>
</tr>
<tr>
<td>Josh Barnhill</td>
<td>emp3961</td>
<td>1990-01-01</td>
</tr>
</tbody>
</table>

Note  Kim Akers (Steve Masters's "parent") and Katie McAskill-White (Masters's "sibling") do not appear in the list, as neither of those relationships are ever found on the descendant-or-self:: axis.

See Also
Navigating Along the descendant Axis
Navigating Along the ancestor-or-self Axis

The ancestor-or-self:: axis, like the ancestor:: axis, locates all nodes in the document hierarchy above the context node. It locates the parent of the context node, the parent's parent, and so on up to the root node. Unlike ancestor::, however, ancestor-or-self:: also selects the context node itself.

Attribute and namespace nodes cannot be located along the ancestor-or-self:: axis, unless the context node is itself an attribute or namespace node. In the latter case, the context node is located because of the or-self component of this axis.

Again referring to our sample XML document, who is in the "chain of command" responsible for setting Josh Barnhill's goals and objectives? You can answer this question with a template rule such as the XSLT file shown below.

The position() and last() functions are used to calculate the number of steps above "Josh Barnhill". For more information, see Using Functions in XPath Expressions.

Example

XML File (orgchart.xml)

Use orgchart.xml (in Sample XML File for Navigating XPath Axes) and edit its href attribute to refer to orgchart-ancself.xsl.

XSLT File (orgchart-ancself.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>

  <xsl:template match="name[.= 'Josh Barnhill']">
    <h2>Josh Barnhill's reporting structure:</h2>
  </xsl:template>
</xsl:stylesheet>
```
Formatted Output

**Josh Barnhill's reporting structure:**

<table>
<thead>
<tr>
<th>Generation</th>
<th>Name</th>
<th>Emp ID</th>
<th>Empl Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Amy Egert</td>
<td>emp5456</td>
<td>1976-03-21</td>
</tr>
<tr>
<td>-1</td>
<td>Steve Masters</td>
<td>emp390</td>
<td>1980-06-18</td>
</tr>
<tr>
<td>0</td>
<td>Josh Barnhill</td>
<td>emp3961</td>
<td>1990-01-01</td>
</tr>
</tbody>
</table>

See Also

Navigating Along the ancestor Axis
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Using Abbreviations in XPath Expressions

Several shortcuts are available to be used in place of certain axes:

- The default axis if none is provided in a given location step is `child::`. Therefore, the following two location steps are equivalent, both locate a `<name>` child of the context node:

```
child::name
name
```

- The "at" sign, `@`, is a shortcut for the `attribute::` axis. Therefore, the following two location steps are equivalent, both locate the `empdate` attribute of a `<director>` child of the context node:

```
director/attribute::empdate
director/@empdate
```

- Two succeeding slashes, `//`, is a shortcut for `/descendant-or-self::node()`. Therefore, the following two location steps are equivalent, locating both the node whose name is "name" and whose string-value is "Kim Akers," and all its descendants, except attribute and namespace nodes:

```
name[.="Kim Akers"]/descendant-or-self::node()/name
name[.="Kim Akers"]//name
```

- A single period, `.`, is a shortcut for `self::node()`. Therefore, the following two location steps, which locate an attribute of the context node, are equivalent:

```
self::node()/attribute::empdate
./attribute::empdate
```

This XPath location step can be further abbreviated using the shortcut for the `attribute::` axis, for example:

```
./@empdate
```
• Two succeeding periods, , are a shortcut for parent::node(). Therefore, the following two location steps are equivalent, both select the parent of the context node, when the parent's value is "Josh Barnett":

    parent::node()[self::node()="Josh Barnett"]
    ..[self::node()="Josh Barnett"]

This location step could be abbreviated even further, using the shortcut for self::node(), as follows:

    ..[.="Josh Barnett"]

• The asterisk (*) is a wild-card character, used in a name node test to indicate that selected nodes may have any names at all. Therefore, given that various elements in orgchart.xml have the same two attributes, empID and empdate, the following two location steps are equivalent, assuming the context node to be one of these elements:

    attribute::empID | attribute::empdate
    attribute::*

This location step could be abbreviated even further, using the shortcut for the attribute:: axis, as follows:

    @*

• While not really a shortcut, a single leading slash, representing the document root in an XPath expression always makes it an absolute path, relative to the document root. Therefore, the following location steps are not equivalent:

    name
    /name

The first selects all <name> elements which are children of the context node; the second, all <name> elements which are children of the root node, that is, it selects an empty node-set, since the root node's only child is the <chairman> element.

Example

This example doesn't generate significant output. It demonstrates that no errors are generated by the above XPath expressions.
XML File (booksxpath.xml)

Use the orgchart.xml (in Sample XML File for Navigating XPath Axes) and change its href attribute to point to xpabbr.xsl.

XSLT File (xpabbr.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <!-- suppress text nodes not covered in subsequent template rule -->
 <xsl:template match="text()"/>

 <xsl:template match="president">
  <p/>
  <xsl:apply-templates select="child::*"/>
 </xsl:template>

 <xsl:template match="name">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>

 <xsl:template match="@empdate">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>

 <xsl:template match="attribute::empdate">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>

 <xsl:template match="attribute::*">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>

 <xsl:template match="attribute::empID | attribute::empdate">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>

 <xsl:template match="@*">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>

 <xsl:template match="name">
  <p/>
  <xsl:value-of select="."/>
 </xsl:template>
</xsl:stylesheet>
```
See Also

Sample XML File for Navigating XPath Axes
Sample XML File for Navigating XPath Axes

An XML file is structured as a hierarchical tree. This tree structuring matches the structure of family trees and organization charts.

For example, a portion of Lucerne Publishing's corporate structure might be represented by the following XML document.

To use this XML file to test the various XSLT files that contain axis templates later in this section, modify the href attribute here to reference the correct XSLT file.

XML File (orgchart.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="orgchart.xsl"?>
<chairman empID="emp3456" empdate="1976-03-21">
  <name>Kim Akers</name>
  <president empID="emp4390" empdate="1980-06-18">
    <name>Steve Masters</name>
    <division>Domestic</division>
    <groupvp empID="emp9801" empdate="1981-09-01">
      <name>Shelly Szymanski</name>
      <department>Sales</department>
      <director empID="emp5443" empdate="1993-04-19">
        <name>Cindy Durkin</name>
        <region>Northeast</region>
      </director>
      <director empID="emp2348" empdate="1990-06-02">
        <name>Michelle Votava</name>
        <region>Southeast</region>
      </director>
      <director empID="emp4322" empdate="1995-05-26">
        <name>John Tippett</name>
        <region>Southwest</region>
      </director>
      <director empID="emp4587" empdate="1990-12-20">
        <name>Alan Steiner</name>
        <region>Northwest</region>
      </director>
    </groupvp>
  </president>
</chairman>
```
<groupvp empID="emp4320" empdate="1978-11-09">
    <name>John Thorson</name>
    <department>Marketing</department>
</groupvp>

<groupvp empID="emp3961" empdate="1990-01-01">
    <name>Josh Barnhill</name>
    <department>Production</department>
</groupvp>

<president>
</president>

<president empID="emp4238" empdate="1984-08-12">
    <name>Katie McAskill-White</name>
    <division>International</division>
    <groupvp empID="emp7216" empdate="1987-04-13">
        <name>Neil Charney</name>
        <department>Sales</department>
        <director empID="emp9032" empdate="1982-08-31">
            <name>Beth Silverberg</name>
            <region>Africa</region>
        </director>
        <director empID="emp7634" empdate="1994-02-12">
            <name>Lani Ota</name>
            <region>Asia</region>
        </director>
        <director empID="emp9032" empdate="1991-10-03">
            <name>Peter Porzuczek</name>
            <region>Europe</region>
        </director>
    </groupvp>
</president>

<president empID="emp3833" empdate="1991-05-15">
    <name>Marea Angela Castaneda</name>
    <department>Marketing</department>
</president>

</chairman>
Sample XSLT File for Navigating XPath Axes

A subset of this file is used in each topic for XPath navigation axes.

XSLT File (orgchart.xsl)

```xml
<?xml version="1.0"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">

<!-- Common template rule (suppresses all text nodes not explicitly covered in succeeding template rules) -->
<xsl:template match="text()"/>

<!-- Parent axis -->
<xsl:template match="director[name='Michelle Votava']">
<table border="1">
<tr>
<th>Attribute name</th>
<th>Attribute value</th>
<th>Parent name</th>
<th>Parent value</th>
</tr>
<xsl:for-each select="attribute::*">
<tr>
<td><xsl:value-of select="name()"/></td>
<td><xsl:value-of select="."/></td>
<td><xsl:value-of select="name(parent::*)"/></td>
<td><xsl:value-of select="parent::*"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>

<!-- Ancestor axis -->
<xsl:template match="*[name='Peter Porzuczek']">
<h2>Peter Porzuczek's superiors:</h2>
<xsl:for-each select="ancestor::*">
<h4><xsl:value-of select="name()"/></h4>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```
<!- Following axis -->
<xsl:template match="name[.= 'Katie McAskill-White']">
  <h2>Katie McAskill-White's followers:</h2>
  <xsl:for-each select="following::*/name">
    <h4><xsl:value-of select="."/></h4>
  </xsl:for-each>
</xsl:template>

<!- Preceding axis -->
<xsl:template match="name[.= 'John Tippett']">
  <table border="1">
    <tr>
      <th>Node name</th>
      <th>Node value</th>
    </tr>
    <xsl:for-each select="preceding::*">
      <xsl:variable name="bgcolor">
        <xsl:choose>
          <xsl:when test="name()='name'">yellow</xsl:when>
          <xsl:otherwise>white</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <tr>
        <xsl:attribute name="style">background-color: <xsl:value-of select="$bgcolor"></xsl:attribute>
        <td><xsl:value-of select="name()"/></td>
        <td><xsl:value-of select="."/></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>

<!- Following-sibling axis -->
<xsl:template match="name[.= 'Shelly Szymanski']">
  <h2>following-siblings of &lt;name&gt;'<xsl:value-of select=".">'&lt;/h2&gt;
  <table border="1">
    <tr>
      <th>Node name</th>
      <th>Node value</th>
    </tr>
    <xsl:for-each select="following-sibling::*">
      <tr>
        <td><xsl:value-of select="name()"/></td>
        <td><xsl:value-of select="."/></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
</xsl:template>
</xsl:template>

<!-- Self axis -->
<xsl:template match="president[division='International']">
  <xsl:for-each select="descendant::name">
    <h2><xsl:value-of select="self::*"/></h2>
  </xsl:for-each>
</xsl:template>

<!-- Descendant-or-self axis -->
<xsl:template match="name[.='Steve Masters']">
  <h2>Steve Masters and descendant elements:</h2>
  <table border="1">
    <tr>
      <th>Name</th>
      <th>Emp ID</th>
      <th>Empl Date</th>
    </tr>
    <xsl:for-each select="parent::*[name()!='name']">
      <tr>
        <td><xsl:value-of select="."/></td>
        <td><xsl:value-of select="parent::*[@empID]"/></td>
        <td><xsl:value-of select="parent::*[@empdate]"/></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>

<!-- Ancestor-or-self axis -->
<xsl:template match="name[.='Josh Barnhill']">
  <h2>Josh Barnhill's reporting structure:</h2>
  <table border="1">
    <tr>
      <th>Generation</th>
      <th>Name</th>
      <th>Emp ID</th>
      <th>Empl Date</th>
    </tr>
    <xsl:for-each select="ancestor-or-self::*[name()!='name']">
      <tr>
        <td align="center">
          <xsl:value-of select="position()-last()"/>
        </td>
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="@empID"/></td>
        <td><xsl:value-of select="@empdate"/></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
Construct Complex XPath Searches

Once you have learned the rudiments of XPath syntax, creating a complex search of an XML document's contents is effectively a matter of combining various location steps into a complex location path or combining multiple location paths to form a compound location path.

This section covers the following:

- [Constructing an XPath with Axis, Node Test, and Predicate](#)
- [Constructing a Multi-step XPath Expression](#)
- [Constructing a Compound Location Path Using Union](#)

See Also

[Introduction to XPath Syntax](#) | [Context and Navigation in XPath](#) | [Use XPath Axes to Navigate through XML Data](#) | [Filtering XML Data Using XPath Predicates](#)
Constructing an XPath with Axis, Node Test, and Predicate

An individual location step in an XPath location path consists of up to three components: an axis, a node test, and a predicate. If all three components are present, the form of a location step is:

```
axis::nodetest[predicate]
```

For example:

```
child::person[attribute::birthdate &lt;= "1951-06-18"]
```

says to start at the context node, locate all its children named "person", and from among those child nodes select only those with a `birthdate` attribute which is less than or equal to "1951-06-18".

Note that the predicate portion of this location step itself contains a location step, `attribute::birthdate`, but that the predicate as a whole does not constitute a location step. For more information, see Differences between XPath Expressions and XSLT Patterns.

**Note** In the remainder of this topic, we will use the XPath abbreviated syntax for axes and node tests. For instance, with the abbreviated syntax, the above location step can be more concisely represented as:

```
person[@birthdate &lt;= "1951-06-18"]
```

For more information, see Using Abbreviations in XPath Expressions.
Constructing a Multi-step XPath Expression

Location paths can contain more than one location step, separated from one another by slashes. Each succeeding location step, reading the pattern from left to right, is evaluated in the context of the nodes selected by all the location steps which precede it.

Consider the following location path:

```
person[@birthdate &lt;= "1951-06-18"]/surname[substring(., 1, 1)="S"
```

This location path consists of two location steps. We have already seen in this section the first step (preceding the slash). It locates a node-set consisting of only one node, or none, or several. If there is at least one node in the resulting node-set, the second step (following the slash) kicks in, and restricts the list of matching nodes even further. Specifically, of the node(s) selected by the first step, the location path as a whole selects only those (if any) with a child named `<surname>` whose content begins with the letter "S".

Note that the above location path moves you down the document tree, from the context node, to its `<person>` child(ren), to the `<surname>` child(ren) of each of the matching `<person>` elements. Because of the wide variety of XPath axes, however, this downward direction is not the only option available to you. For example, a location path might be constructed as follows:

```
person[@birthdate &lt;= "1951-06-18"]/surname[substring(., 1, 1)="S"
```

The two addition location steps in this path move you from the node(s) selected in the earlier example up to their parents (..), and then sideways to include only the nodes which are preceding siblings of those parents (`preceding-sibling::*`).

See Also

[Use XPath Axes to Navigate through XML Data](#) | [Constructing the Node-Test](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Deve
Constructing a Compound Location Path Using Union

Simple XPath location paths consist of one or more location steps, any or all of which can be quite complex.

The XPath specification allows an even broader reach through any document, by providing for compound location paths — multiple location paths, separated from one another by a vertical bar or pipe symbol (|). The resulting node-set represents the union of all nodes retrieved by all of its constituent location paths.

We see simple examples of compound location paths in the built-in template rules defined by the XSLT specification. For instance:

```
<xsl:template match="comment() | processing-instruction()" />
```

matches the set of all nodes in the document which are of either the comment or the processing instruction node types. For more information about the built-in template rules, see How XSLT Template Rules Depend on Context.

Most often, compound location paths are used to simplify locating document content which comes from different branches of the document tree. For example, consider the example XML document shown below.

Although this document is structurally simple, using a single location path to obtain content from different branches of its tree of nodes can be extremely complicated.

You could, for example, locate the <primate> element corresponding to "gibbon", the second <marsupial> element (whatever it is), and all the <insect> elements using a single location path consisting of multiple steps, utilizing various axes such as ancestor::, parent::, preceding-sibling::, and child::—perhaps together with the position() or other functions. Such a location path would be difficult to construct in the first place, difficult to maintain, and quite possibly useless if intervening elements are later added to the document (or existing ones dropped).
It is much simpler to achieve this with a compound location path such as the match pattern in the example XSLT template rule below.

**Example**

**XML File (animals.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="animals.xsl" ?>
<animals>
    <insects>
        <insect>bee</insect>
        <insect>fly</insect>
        <insect>beetle</insect>
        <insect>ant</insect>
    </insects>
    <mammals>
        <marsupials>
            <marsupial>kangaroo</marsupial>
            <marsupial>opossum</marsupial>
        </marsupials>
        <primates>
            <primate>gibbon</primate>
            <primate>lemur</primate>
            <primate>gorilla</primate>
            <primate>chimpanzee</primate>
        </primates>
    </mammals>
</animals>
```

**XSLT File (animals.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- suppress text nodes not covered in subsequent template rule -->
    <xsl:template match="text()"/>

        <p><xsl:value-of select="."/></p>
    </xsl:template>
</xsl:stylesheet>
```

**Formatted Output**
bee
fly
beetle
ant
opossum
gibbon

**Processor Output**

```xml
<?xml version="1.0" encoding="UTF-16"?><p>bee</p><p>fly</p><p>beetle</p>
```

For more information about the `position()` function, see [Processing Node-Sets by Using Node-Set Functions](#).
XPath Reference

XML Path Language (XPath) is a general-purpose query notation for addressing and filtering the elements and text of XML documents. XPath expressions can address parts of an XML document; can manipulate strings, numbers, and Booleans; and can match a set of nodes in the document.

MSXML versions 4.0 and later support XPath within Extensible Stylesheet Language Transformations (XSLT), and through the DOM extensions, selectNodes and selectSingleNode. Starting with version 4.0, MSXML implements XPath as described in the XML Path Language (XPath) Version 1.0 Recommendation by the Worldwide Web Consortium (W3C). For information about the parser's conformance, see Supported XSLT Features.

XPath models an XML document as a tree of different type nodes. XPath expressions identify these nodes in the XML document based on their type, name, and values, as well as the relationship of a node to other nodes in the document. An XPath expression yields one of the following basic objects.

- node set
- Boolean
- number
- string

For example, the query "find the <author> elements that have a period attribute with the value of 'classical', and that are contained in the <authors> element at the document root" can be expressed as in XPath expression as "/authors/author[@period='classical']".

XPath collections are 1-based.

This section covers the following:

- XPath Syntax
- XPath Functions

See Also
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
XPath Syntax

An XPath expression uses a path notation, like those used in URLs, for addressing parts of an XML document. The expression is evaluated to yield an object of the node-set, Boolean, number, or string type. For example, the expression `book/author` will return a node-set of the `<author>` elements contained in the `<book>` elements, if such elements are declared in the source XML document. In addition, an XPath expression can have predicates (filter expressions) or function calls. For example, the expression `book[@type="Fiction"]` refers to the `<book>` elements whose type attribute is set to "Fiction".

The following table summarizes some of the analogous features between URLs and XPath expressions.

<table>
<thead>
<tr>
<th>URLs</th>
<th>XPath expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy comprised of folders and files in a file system.</td>
<td>Hierarchy comprised of elements and other nodes in an XML document.</td>
</tr>
<tr>
<td>Files at each level have unique names. URLs always identify a single file.</td>
<td>Element names at each level might not be unique. XPath expressions identify a set of all the matching elements.</td>
</tr>
<tr>
<td>Evaluated relative to a particular folder, called the &quot;current folder.&quot;</td>
<td>Evaluated relative to a particular node called the &quot;context&quot; for the expression.</td>
</tr>
</tbody>
</table>

This section covers the syntax of XML Path Language (XPath) expressions, including the following:

- [Context for XPath Expressions](#)
- [Operators and Special Characters](#)
- [Collections](#)
- [Filters and Filter Patterns](#)
- [Boolean, Comparison, and Set Expressions](#)
- [Comparisons](#)
- [Set Operations](#)
- [Location Paths](#)
- [XPath Examples](#)
Topics in this section use the Sample XML File for XPath Syntax (inventory.xml).

XPath also support namespaces and data types. Namespace prefixes can be included in expressions so that the matching operations can check for specific namespace prefixes.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Context for XPath Expressions

The evaluation of an XPath expression depends on the context against which the expression operates. The context consists of the node against which the expression is evaluated and its associated environment, which includes the following:

- The position of the context node in the document order, relative to its siblings.
- The size of the context—that is, the number of siblings of the context node plus one.
- Variable bindings with which references to a variable can be resolved.
- A function library.
- The namespace declarations in scope for the expression.

To better appreciate the concept of context, consider a tree containing nodes. Asking for all nodes named X from the root of the tree returns one set of results, while asking for those nodes from a branch in the tree returns a different set of results. Thus, the result of an expression depends upon the context against which it executes.

XPath expressions can match specific patterns at one particular context, return the results, and perform additional operations relative to the context of the returned nodes. This gives XPath expressions extraordinary flexibility in searching throughout the document tree.
Basic XPath Expressions

The following are basic types of XPath expressions. Each type is described below.

- Current context
- Document root
- Root element
- Recursive descent
- Specific element

Examples

The following examples show some basic XPath expressions. More complex expressions are possible by combining these simple expressions together and by using the various XPath operators and special characters.

**Current context**
An expression prefixed with a period and forward slash (./) explicitly uses the current context as the context. For example, the following expression refers to all `<author>` elements within the current context:

```
./author
```

Note that this is equivalent to the following:

```
author
```

**Document root**
An expression prefixed with a forward slash (/) uses the root of the document tree as the context. For example, the following expression refers to the `<bookstore>` element at the root of this document:

```
/bookstore
```

**Root element**
An expression that uses a forward slash followed by an asterisk (/*) uses the root element as the context. For example, the following expression finds the root element of the document:
/*

Recursive descent

An expression that uses the double forward slash (//) indicates a search that can include zero or more levels of hierarchy. When this operator appears at the beginning of the pattern, the context is relative to the root of the document. For example, the following expression refers to all <author> elements anywhere within the current document:

//author

The .// prefix indicates that the context starts at the level in the hierarchy indicated by the current context.

Specific elements

An expression that starts with an element name refers to a query of the specific element, starting from the current context node. For example, the following expression refers to the <background.jpg> element within the <images> element in the current context node:

images/background.jpg

The following expression refers to the collection of <book> elements within the <bookstore> elements in the current context node:

bookstore/book

The following expression refers to all <first.name> elements in the current context node:

first.name

Note Element names can include the period character (.). These names can be used just like any other name.
Context in the DOM

When using XPath expressions with the Microsoft XML DOM, the context is the Node object whose `selectNodes` method or `selectSingleNode` method is called.

When using XPath directly from the DOM, you define the context from a particular node. For more information, see How the DOM Defines the Context for XPath Expressions.
Context in XSLT

When using XPath directly from the XSLT, you define the context by the current node. For more information about how a context is defined, see How XSLT Defines the Context for XPath Expressions.

Other Resources

XML Path Language (XPath) Version 1.0
Operators and Special Characters

XPath expressions are constructed using the operators and special characters shown in the following table.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Child operator; selects immediate children of the left-side collection. When this path operator appears at the start of the pattern, it indicates that children should be selected from the root node. Recursive descent; searches for the specified element at any depth. When this path operator appears at the start of the pattern, it indicates recursive descent from the root node.</td>
</tr>
<tr>
<td>//</td>
<td>Indicates the current context.</td>
</tr>
<tr>
<td>.</td>
<td>Wildcard; selects all elements regardless of the element name.</td>
</tr>
<tr>
<td>@</td>
<td>Attribute; prefix for an attribute name.</td>
</tr>
<tr>
<td>@*</td>
<td>Attribute wildcard; selects all attributes regardless of name.</td>
</tr>
<tr>
<td>:</td>
<td>Namespace separator; separates the namespace prefix from the element or attribute name.</td>
</tr>
<tr>
<td>( )</td>
<td>Groups operations to explicitly establish precedence.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Applies a filter pattern.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Subscript operator; used for indexing within a collection.</td>
</tr>
<tr>
<td>+</td>
<td>Performs addition.</td>
</tr>
<tr>
<td>-</td>
<td>Performs subtraction.</td>
</tr>
<tr>
<td>div</td>
<td>Performs floating-point division according to IEEE 754.</td>
</tr>
<tr>
<td>*</td>
<td>Performs multiplication.</td>
</tr>
<tr>
<td>mod</td>
<td>Returns the remainder from a truncating division.</td>
</tr>
</tbody>
</table>

This table does not include Boolean and set operators, which are listed in [Boolean, Comparison, and Set Expressions](link) or [Set Operations](link).

Precedence order (from highest precedence to lowest) is defined as indicated in the following table.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Character</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( )</td>
<td>Grouping</td>
</tr>
</tbody>
</table>
The group operator, ( ), is applicable only at the top-level path expression. For example, (author/degree | author/name) is a valid grouping operation, but author/(degree | name) is not.

The filter pattern operators ([ ]) have a higher precedence than the path operators (/ and //). For example, the expression //comment()[3] selects all comments with an index equal to 3 relative to the comment's parent anywhere in the document. This differs from the expression //comment()[3], which selects the third comment from the set of all comments relative to the parent. The first expression can return more than one comment, while the latter can return only one comment.

These operators and special characters are described in detail throughout this reference.
Path Operators

The collection of elements of a certain type can be determined using the path operators (/ and //). These operators take as their arguments a "left side" collection on which to perform the selection and a "right side" collection indicating which elements to select. The child operator (/) selects from immediate children of the left-side collection, while the descendant operator (//) selects from arbitrary descendants of the left-side collection. In effect, // can be considered a substitute for one or more levels of hierarchy.

Note that the path operators change the context as the query is performed. By stringing path operators together, users can traverse the document tree.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>author/first-name</td>
<td>All &lt;first-name&gt; elements within an &lt;author&gt; element of the current context node.</td>
</tr>
<tr>
<td>bookstore/title</td>
<td>All &lt;title&gt; elements one or more levels deep in the &lt;bookstore&gt; element (arbitrary descendants). Note that this is different from the following pattern, bookstore/*/title.</td>
</tr>
<tr>
<td>bookstore/*/title</td>
<td>All &lt;title&gt; elements that are grandchildren of &lt;bookstore&gt; elements.</td>
</tr>
<tr>
<td>bookstore//book/excerpt//emph</td>
<td>All &lt;emph&gt; elements anywhere inside &lt;excerpt&gt; children of &lt;book&gt; elements, anywhere inside the &lt;bookstore&gt; element:</td>
</tr>
<tr>
<td>.//title</td>
<td>All &lt;title&gt; elements one or more levels deep in the current context. Note that this situation is essentially the only one in which the period notation is required.</td>
</tr>
</tbody>
</table>
**Wildcard Character**

An element can be referenced without using its name by substituting the wildcard (*) collection. The * collection refers to all elements that are children of the current context, regardless of the tag name.

### Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>author/*</td>
<td>All element children of <code>&lt;author&gt;</code> elements.</td>
</tr>
<tr>
<td>book/*/last-name</td>
<td>All <code>&lt;last-name&gt;</code> elements that are grandchildren of <code>&lt;book&gt;</code> elements.</td>
</tr>
<tr>
<td>*/</td>
<td>All grandchildren elements of the current context.</td>
</tr>
<tr>
<td>my:*</td>
<td>All elements from the <code>my</code> namespace.</td>
</tr>
</tbody>
</table>

Note that the pattern `*:book` is not supported.
Attributes

XPath denotes attribute names with the @ symbol. Attributes and child elements are treated impartially, and capabilities are equivalent between the two types wherever possible.

**Note** Attributes cannot contain child elements, so syntax errors occur when path operators are applied to attributes. In addition, you cannot apply an index to attributes because, by definition, no order is defined for attributes.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>@style</td>
<td>The style attribute of the current element context.</td>
</tr>
<tr>
<td>price/@exchange</td>
<td>The exchange attribute of &lt;price&gt; elements within the current context.</td>
</tr>
<tr>
<td>book/@style</td>
<td>The style attribute of all &lt;book&gt; elements.</td>
</tr>
</tbody>
</table>

Note that the following example is not valid, because an attribute cannot have any children.

price/@exchange/total
Finding Multiple Attributes

All attributes of an element can be returned using @*. This is potentially useful for applications that treat attributes as fields in a record.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>@*</td>
<td>All attributes of the current context node.</td>
</tr>
<tr>
<td>@my:*</td>
<td>All attributes from the my namespace. This does not include unqualified attributes on elements from the my namespace.</td>
</tr>
</tbody>
</table>

Note that the pattern @*:title is not supported.

Try It!

The following is an interactive demonstration to illustrate the features of XPath expressions discussed above. When you select a context node and an XPath expression from the drop-down menu, you will see them displayed in colors in the XML source document that follows.

To proceed, click here.

See Also

[Set Operations](#) | [Sample XML File for XPath Syntax (inventory.xml)](#) | [XPath Examples](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Collections

Collections returned by XPath queries preserve document order, hierarchy, and identity, to the extent that these are defined. That is, a collection of elements is returned in document order without repeated elements. Because by definition attributes are unordered, there is no implicit order to attributes returned for a specific element.

The collection of all elements with a certain tag name is expressed using the tag name itself. This can be qualified by showing that the elements are selected from the current context by using a period and forward slash (./), but the current context is used by default and does not have to be noted explicitly.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>./first-name</td>
<td>All &lt;first-name&gt; elements. Note that this expression is equivalent to the expression that follows.</td>
</tr>
<tr>
<td>first-name</td>
<td>All &lt;first-name&gt; elements.</td>
</tr>
</tbody>
</table>
Indexing into a Collection

XPath expressions make it easy to query a specific node within a set of nodes. Simply enclose the index ordinal within square brackets. The ordinal is 1-based (the first element is number 1).

The square bracket characters ([ ]) have higher precedence than the slash characters (/ and //). For more information see Operators and Special Characters.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>author[first-name][3]</td>
<td>The third &lt;author&gt; element that has a &lt;first-name&gt; child element.</td>
</tr>
</tbody>
</table>

Note that indexes are relative to the set being filtered. Consider, for example, the following data.

```
<x>
  <y/>
  <y/>
</x>
<x>
  <y/>
  <y/>
</x>
```

The following table shows how to select specific <x> and <y> elements.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>x/y[1]</td>
<td>The first &lt;y&gt; inside each &lt;x&gt;.</td>
</tr>
<tr>
<td>(x/y)[1]</td>
<td>The first &lt;y&gt; from the entire set of &lt;y&gt; elements within &lt;x&gt; elements.</td>
</tr>
<tr>
<td>x[1]/y[1]</td>
<td>The first &lt;y&gt; inside the first &lt;x&gt;.</td>
</tr>
</tbody>
</table>

The examples above are simple references to XPath collections that use implied
defaults, such as the child:: axis. For this axis, the collection of child nodes is indexed in forward document order.

For other axes, such as ancestor::, use the axis name explicitly in your XPath expression. For this axis, the collection of ancestors is indexed in reverse document order. Consider this example from the previous table:

\[ x/y[1] \]

This expression is equivalent to this one:

\[ x/child::y[1] \]

Both expressions mean "for each <x> element, select the first child element named <y>".

The following example uses the same syntax.

\[ x/ancestor::y[1] \]

This example translates to "for each <x> element, select the first ancestor element (in reverse-document order) named <y>". The syntax is the same, but the order is reversed.
Finding the Last Element in a Collection

The `last()` function returns True for the last element in a collection. Note that `last` is relative to the parent node.

**Examples**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>book[last()]</td>
<td>The last <code>&lt;book&gt;</code> element.</td>
</tr>
<tr>
<td>book/author[last()]</td>
<td>The last <code>&lt;author&gt;</code> element inside each <code>&lt;book&gt;</code> element.</td>
</tr>
<tr>
<td>(book/author)[last()]</td>
<td>The last <code>&lt;author&gt;</code> element from the entire set of <code>&lt;author&gt;</code> elements inside <code>&lt;book&gt;</code> elements.</td>
</tr>
</tbody>
</table>
Grouping

Parentheses can be used to group collection operators for clarity or where the normal precedence is inadequate to express an operation. Grouping operators can be used in any filter expressions (predicates), such as `author[(degree or award) and publication]`. They can also be used in the top-level step expression, such as `(book|magazine)` or `(author/degree | book/award)`. They cannot be applied to lower-level step expressions. For example, `author/(degree | award)` is not valid.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(book/author)</td>
<td>All <code>&lt;author&gt;</code> elements that are child elements of any <code>&lt;book&gt;</code> element from the current context node.</td>
</tr>
<tr>
<td><code>author[(degree or award) and publication]</code></td>
<td>All <code>&lt;author&gt;</code> elements that contain at least one <code>&lt;degree&gt;</code> or <code>&lt;award&gt;</code> element and at least one <code>&lt;publication&gt;</code> element.</td>
</tr>
</tbody>
</table>

See Also

Sample XML File for XPath Syntax (inventory.xml) | XPath Examples
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Filters and Filter Patterns

Constraints and branching can be applied to any collection by adding a filter clause, \([\text{pattern}]\), to the collection. The filter is analogous to the SQL WHERE clause. The filter contains a pattern within it called the filter pattern. The filter pattern evaluates to a Boolean value and is tested for each element in the collection. Any elements in the collection failing the filter pattern test are omitted from the result collection.

For convenience, if a collection is placed within the filter, a Boolean TRUE is generated if the collection contains any members and a FALSE is generated if the collection is empty. An expression such as \(\text{author/degree}\) implies a collection-to-Boolean conversion function that evaluates to TRUE if there exists an \(<\text{author}>\) element with a child element named \(<\text{degree}>\).

Note that any number of filters can appear at a given level of an expression. Empty filters are not allowed.

Filters are always evaluated with respect to a context. In other words, the expression \(\text{book[author]}\) means that for every \(<\text{book}>\) element that is found, test whether it has an \(<\text{author}>\) child element. Likewise, \(\text{book[author = 'Bob']}\) means that for every \(<\text{book}>\) element that is found, test whether it has an \(<\text{author}>\) child element with the value Bob. One can examine the value of the context as well by using the period (\(\cdot\)) character. For example, \(\text{book[. = 'Trenton']}\) means that for every book that is found in the current context, test whether its value is Trenton.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>book[excerpt]</td>
<td>All (&lt;\text{book}&gt;) elements that contain at least one (&lt;\text{excerpt}&gt;) element.</td>
</tr>
<tr>
<td>book[excerpt]/title</td>
<td>All (&lt;\text{title}&gt;) elements inside (&lt;\text{book}&gt;) elements that contain at least one (&lt;\text{excerpt}&gt;) element.</td>
</tr>
<tr>
<td></td>
<td>All (&lt;\text{author}&gt;) elements that contain at least one (&lt;\text{excerpt}&gt;) element.</td>
</tr>
</tbody>
</table>
book[excerpt]/author[degree] least one <degree> element, and are inside of <book> elements that contain at least one <excerpt> element.

book[author/degree] All <book> elements that contain at least one <author> element with at least one <degree> child element.

book[excerpt][title] All <book> elements that contain at least one <excerpt> element and at least one <title> element.

See Also

Sample XML File for XPath Syntax (inventory.xml) | XPath Examples
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Boolean, Comparison, and Set Expressions

Filter patterns can contain Boolean expressions, comparison expressions, and set expressions. Shortcuts listed in the following table represent alternative symbols that are provided in this XSL Transformations (XSLT) implementation. This documentation discusses these expression operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>Logical-and</td>
</tr>
<tr>
<td>or</td>
<td>Logical-or</td>
</tr>
<tr>
<td>not()</td>
<td>Negation</td>
</tr>
<tr>
<td>=</td>
<td>Equality</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal</td>
</tr>
<tr>
<td>&lt; *</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;= *</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>&gt; *</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;= *</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td></td>
<td>Set operation; returns the union of two sets of nodes</td>
</tr>
</tbody>
</table>

* Extended XPath method

The World Wide Web Consortium (W3C) syntax for operator keywords uses white space and other separators rather than the dollar sign character ($) used in version 2.5. In the W3C syntax, a binary keyword of the form $xxx$ can be expressed as wsxxxws, where ws refers to a token terminator that can be white space, single quote characters ('), or double quote characters ("). Unary operators such as not() use functional notation. Although the Microsoft implementation supports both syntaxes, it is recommended that the W3C syntax be used for future compatibility.

Precedence order (from highest to lowest) for comparison operators and Boolean operators is shown in the following table.
1 ( ) Grouping
2 [ ] Filters
3 // Path operations

4 < or &lt;

5 <= or &lt;=

6 > or &gt;

7 >= or &gt;=

8 =

9 != Comparisons

10 | Union
11 not() Boolean not
12 And Boolean and
13 Or Boolean or

When the operators are used in an XML document, such as an XSLT style sheet, the < and > tokens must be escaped as &lt; and &gt;, respectively. For example, the following XSLT instruction invokes an XSLT template rule on all <book> elements whose <price> element has a numeric value less than or equal to 10.

<xsl:apply-templates select="book[price &lt;= 10]"/>

When an XPath expression is used with DOM, the < and > operators need not to be escaped. For example, the following JScript statement selects all <book> elements whose <price> element has a numeric value less than or equal to 10.

var cheap_books = dom.selectNodes("book[price <= 10]");

Boolean expressions can match all nodes of a particular value or all nodes with nodes in particular ranges. The following is an example of a Boolean expression that returns false.
1 \geq 2

Operators are case-sensitive.
Logical-and and Logical-or

The Boolean operators and and or perform logical-and and logical-or operations, respectively. These operators, in conjunction with grouping parentheses, can be used to build sophisticated logical expressions.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>author[degree and award]</td>
<td>All &lt;author&gt; elements that contain at least one &lt;degree&gt; element and at least one &lt;award&gt; element.</td>
</tr>
<tr>
<td>author[(degree or award) and publication]</td>
<td>All &lt;author&gt; elements that contain at least one &lt;degree&gt; or &lt;award&gt; element, and at least one &lt;publication&gt; element.</td>
</tr>
</tbody>
</table>
Boolean not

The Boolean not operator negates the value of an expression within a filter pattern.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>author[degree and not(publication)]</td>
<td>All &lt;author&gt; elements that contain at least one &lt;degree&gt; element, but contain no &lt;publication&gt; elements</td>
</tr>
<tr>
<td>author[not(degree or award) and publication]</td>
<td>All &lt;author&gt; elements that contain at least one &lt;publication&gt; element, but do not contain any &lt;degree&gt; elements or &lt;award&gt; elements.</td>
</tr>
</tbody>
</table>

Example

XML File (test.xml)

```xml
<?xml version="1.0"?>
<test>
  <x a="1">
    <x a="2" b="B">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
  <x a="2">
    <y>y2</y>
  </x>
  <x a="3">
    <y>y3</y>
  </x>
</test>
```
XSLT File (test.xsl)

The following XSLT stylesheet selects all the <x> elements without any attributes.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>
<!-- suppress text nodes not covered in subsequent template rule -->
  <xsl:template match="text()"/>
  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:apply-templates select="*|@*"/>
      <xsl:if test="text()">
        <xsl:value-of select="."/>
      </xsl:if>
    </xsl:element>
  </xsl:template>
  <xsl:template match="@*>">
    <xsl:attribute name="{name()}">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>
  <xsl:template match="/test">
    <xsl:apply-templates select="//x[not(@*)]"/>
  </xsl:template>
</xsl:stylesheet>
```

Output

The transformation, when applied to the XML file given above yields the following result:

```xml
<x>
    <y>y31</y>
    <y>y32</y>
</x>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Comparisons

To compare two objects in XPath, use the = sign to test for equality, or use != to test for inequality.

For a comparison operation, exactly two operands must be supplied. Comparisons are then made by evaluating each operand, and converting them as needed, so they are of the same type. This is done according to the process described below, in *Order of Precedence For Comparisons*.

All elements and attributes are strings, but are automatically cast as integer values for numeric comparisons. Literal numeric values are cast to long or double types during comparison operations, as shown in the following table.

For information about &lt; and other binary comparison operators, see *Binary Comparison Operators*, below.

<table>
<thead>
<tr>
<th>Literal type</th>
<th>Comparison</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>text(lvalue) op text(rvalue)</td>
<td>a &lt; GGG</td>
</tr>
<tr>
<td>Integer</td>
<td>(long) lvalue op (long) rvalue</td>
<td>a &lt; 3</td>
</tr>
<tr>
<td>Real</td>
<td>(double) lvalue op (double) rvalue</td>
<td>a &lt; 3.1</td>
</tr>
</tbody>
</table>

Single or double quotation marks can be used for string delimiters in expressions. This makes it easier to construct and pass patterns from within scripting languages.

For more information about how comparisons are performed using XPath, see section 3.4 ("Booleans") of the *W3C XML Path Language (XPath) Recommendation*.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>All &lt;author&gt; elements that contain at</td>
<td></td>
</tr>
</tbody>
</table>
author[last-name = "Bob"]
least one <last-name> element with the value Bob.

author[last-name[1] = "Bob"]
All <author> elements whose first <last-name> child element has the value Bob.

author/degree[@from != "Harvard"]
All <author> elements that contain <degree> elements with a from attribute that is not equal to "Harvard".

author[last-name = /editor/last-name]
All <author> elements that contain a <last-name> element that is the same as the <last-name> element inside the <editor> element under the root element.

author[. = "Matthew Bob"]
All <author> elements whose string value is Matthew Bob.
Order of Precedence for Comparisons

Comparisons with regard to data types obey the order of precedence.

- If at least one operand is a Boolean, each operand is first converted to a Boolean.
- Otherwise, if at least one operand is a number, each operand is first converted to a number.
- Otherwise, if at least one operand is a date, each operand is first converted to a date.
- Otherwise, both operands are first converted to strings.
Binary Comparison Operators

A set of binary comparison operators compares numbers and strings and returns Boolean results. The &lt;, &lt;=, &gt;, and &gt;= operators are used for less than, less than or equal, greater than, and greater than or equal, respectively. Single or double quotation marks can be used for string delimiters in expressions. This makes it easier to construct and pass patterns within scripting languages.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>author[last-name = &quot;Bob&quot; and price &gt; 50]</td>
<td>All &lt;author&gt; elements that contain a &lt;last-name&gt; element with the value Bob and a &lt;price&gt; element with a value greater than 50.</td>
</tr>
<tr>
<td>degree[@from != &quot;Harvard&quot;]</td>
<td>All &lt;degree&gt; elements with a from attribute that is not equal to &quot;Harvard&quot;.</td>
</tr>
<tr>
<td>author[last-name &gt;= &quot;M&quot;]</td>
<td>All &lt;author&gt; elements that contain a &lt;last-name&gt; element with a value that begins with the letter M or greater.</td>
</tr>
</tbody>
</table>

Example

**XML File (test.xml)**

```xml
<?xml version="1.0"?>
<test>

  <x a="1">
    <x a="2" b="B">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
</test>
```
The following XSLT style sheet selects all the <x> elements that are the first of their siblings in the document order.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>

  <!-- Suppress text nodes not covered in subsequent template rule. -->
  <xsl:template match="text()"/>

  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:apply-templates select="*|@*"/>
      <xsl:if test="text()">
        <xsl:value-of select="."/>
      </xsl:if>
    </xsl:element>
  </xsl:template>

  <xsl:template match="@*">
    <xsl:attribute name="{name()}">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>

  <xsl:template match="/test">
    <xsl:apply-templates select="//x[position() = 1 ] "/>
  </xsl:template>
</xsl:stylesheet>
```

**Formatted Output**
The transformation applied to the XML file above yields the following result:

```xml
<x a="1">
  <x a="2" b="B">
    <x>
      <y>y31</y>
      <y>y32</y>
    </x>
  </x>
</x>
<x a="2" b="B">
  <x>
    <y>y31</y>
    <y>y32</y>
  </x>
</x>
<x>
  <y>y31</y>
  <y>y32</y>
</x>
```

**See Also**

[Sample XML File for XPath Syntax (inventory.xml)](inventory.xml) | [XPath Examples]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Set Operations

XML Path Language (XPath) supports the set operation |.
Union (|) Operator

The |, or union, operator returns the union of its two operands, which must be node-sets. For example, //author | //publisher returns a node-set that combines all the //author nodes and all the //publisher nodes. Multiple union operators can be chained together to combine multiple node-sets. For example, //author | //publisher | //editor | //book-seller returns a node-set containing all //author, //publisher, //editor, and //book-seller elements. The union operator preserves document order and does not return duplicates.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-name</td>
<td>last-name</td>
</tr>
<tr>
<td>(book</td>
<td>magazine)/price</td>
</tr>
</tbody>
</table>

Example

The following example illustrates the effect of the union operator.

XML File (test.xml)

```xml
<?xml version="1.0"?>
<test>
  <x a="1">
```
The following XSLT style sheet selects all the $\langle x \rangle$ elements whose $a$ attribute is equal to 2, plus those $\langle x \rangle$ elements that have no attributes.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>
   <!-- Suppress text nodes not covered in subsequent template rule. -->
   <xsl:template match="text()"/>
   <!-- Handles a generic element node. -->
   <xsl:template match="*">
     <xsl:element name="{name()}">
       <xsl:apply-templates select="*|@*"/>
       <xsl:if test="text()">
         <xsl:value-of select="."/>
       </xsl:if>
     </xsl:element>
   </xsl:template>
   <!-- Handles a generic attribute node. -->
   <xsl:template match="@*">
     <xsl:attribute name="{name()}">
       <xsl:value-of select="."/>
     </xsl:attribute>
   </xsl:template>
   <xsl:template match="/test">
     <xsl:apply-templates select="//x[@a=2 | //x[not(@*)]]"/>
   </xsl:template>
</xsl:stylesheet>
```

The transformation yields the following result:

```xml
<x a="2" b="B">
  <x>
    <y>y31</y>
    <y>y32</y>
  </x>
</x>
```
<x>
  <y>31</y>
  <y>y32</y>
</x>
</x>
<x>
  <y>y31</y>
  <y>y32</y>
</x>


**Precedence**

Precedence order (from highest precedence to lowest) between Boolean and comparison operators is shown in the following table.

1 ( ) Grouping
2 [ ] Filters
3 / Path operations
   
   &lt;
   &lt;=
4 &gt; Comparisons
   &gt;=
   =
5 != Comparisons
6 | Union
7 not() Boolean not
8 and Boolean and
9 or Boolean or

**Example**

The following example illustrates the effect of the operator precedence listed above.

**XML File (test.xml)**

```xml
<?xml version="1.0"?>
<test>
   <x a="1">
```
Basic XSLT File (test.xsl)

We will use this basic XSLT file as a starting point for the series of illustrations that follow.

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>

  <!-- Suppress text nodes not covered in subsequent template rule. -->
  <xsl:template match="text()"/>

  <!-- Handles a generic element node. -->
  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:apply-templates select="*|@*"/>
      <xsl:if test="text()">
        <xsl:value-of select="."/>
      </xsl:if>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```
Case 0. Test run

You can add the following template-rule to the XSLT style sheet.

```xml
<xsl:template match="/test">
    <xsl:apply-templates select="*|@*/>
</xsl:template>
```

This will produce an XML document identical to the original one, without the `<? xml version="1.0"?>` processing instruction.

The following cases show different ways of writing this template rule. The point is to show the order in which the XPath operators bind to an element.

**Case 1: () binds tighter than []**

The following template rule selects the first `<y>` element in the document order, from all the `<y>` elements in the source document.

```xml
<xsl:template match="/test">
    <xsl:apply-templates select="//(y)[1]"/>
</xsl:template>
```

The result is as follows:

`<y>y31</y>`

**Case 2: [] binds tighter than / or //**

The following template rule selects all the `<y>` elements that are the first among their siblings.

```xml
<xsl:template match="/test">
```
The result is as follows:

<y>y31</y>
<y>y21</y>
<y>y11</y>
<y>y03</y>

**Case 3: and, not**

The following template rule selects all the <x> elements that have no <x> child elements, that have an <x> parent element, and that do not have any attributes.

<xsl:template match="/test">
    <xsl:apply-templates select="//x[./ancestor::*[name()='x'] and *[name()!='x'] and not(@*)]"/>
</xsl:template>

The result is a single <x> element, listed below with its children:

<x>
    <y>y31</y>
    <y>y32</y>
</x>

**Case 4: or, and, not**

The following template rule selects each <x> elements that is a child of an <x> element; or, that is not a parent of an <x> element and has no attributes.

<xsl:template match="/test">
    <xsl:apply-templates select="//x[./ancestor::*[name()='x'] or *[name()!='x'] and not(@*)]"/>
</xsl:template>

The result is a node set containing the following <x> elements, listed below with its children:

<x a="2" b="B">
    <x>
        <y>y31</y>
        <y>y32</y>
    </x>
</x>
Case 5: and, or, not

The following template rule selects each `<x>` element that is a child of an `<x>` element but not a parent of an `<x>` element; or, that has no attributes.

```xml
<xsl:template match="/test">
  <xsl:apply-templates select="//x[./ancestor::*[name()='x'] and *[name()!='x'] or not(@*)]"/>
</xsl:template>
```

The result is a node set containing the following `<x>` elements, listed below with its children:

```xml
<x>
  <y>y31</y>
  <y>y32</y>
</x>
<x a="2">
  <y>y21</y>
  <y>y22</y>
</x>
<x>
  <y>y03</y>
  <y>y04</y>
</x>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Location Paths

A location path is an XPath expression used for selecting a set of nodes relative to the context node. The evaluation of a location path expression results in a node-set containing the nodes specified by the location path. A location path can recursively contain expressions used to filter sets of nodes.

Syntactically, a location path consists of one or more location steps, each separated by a forward slash (/):

locationstep/locationstep/locationstep

Each location step in turn selects a set of nodes relative to the context node—that is, to the node selected by the preceding location step. A location path expressed this way is a relative location path. An absolute location path starts from the root element:

/locatedstep/locationstep/locationstep

In a location path, location steps are evaluated from left to right. The leftmost location step selects a set of nodes relative to the context node. These nodes then become the context used to process the next location step. This processing of steps and updating of the context node repeats itself until all the location steps have been processed.

A location path can be unabbreviated or abbreviated.

In an unabbreviated location path, a location step has the following syntax:

axis::node-test[predicate]

In this syntax, axis specifies how the nodes selected by the location step are related to the context node; node-test specifies the node type and expanded name of the nodes selected by the location step; and predicate is a filter expression to further refine the selection of nodes in the location step. Predicates are optional. In this case, a location step consists of axis:: and node-test only. The following table provides some examples.
### Unabbreviated Location Path

<table>
<thead>
<tr>
<th>Location Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>child::para[last()]</code></td>
<td>Selects the last <code>&lt;para&gt;</code> element of the context node.</td>
</tr>
<tr>
<td><code>parent::para</code></td>
<td>Selects the <code>&lt;para&gt;</code> element that is the parent of the context node.</td>
</tr>
<tr>
<td><code>child::text()</code></td>
<td>Selects all text node children of the context node.</td>
</tr>
<tr>
<td><code>child::div/child::para</code></td>
<td>Selects the <code>&lt;para&gt;</code> child elements of the <code>&lt;div&gt;</code> element that is a child of the context node.</td>
</tr>
</tbody>
</table>

In an abbreviated location path, the axis specifier, `axis::`, is not expressed explicitly in a location step, but implied by a set of shot cuts instead. The following table provides some examples.

### Abbreviated Location Path

<table>
<thead>
<tr>
<th>Location Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>para</code></td>
<td>Selects the <code>&lt;para&gt;</code> elements of the context node.</td>
</tr>
<tr>
<td><code>../para</code></td>
<td>Selects the <code>&lt;para&gt;</code> element that is the parent of the context node.</td>
</tr>
<tr>
<td><code>text()</code></td>
<td>Selects all text node children of the context node.</td>
</tr>
<tr>
<td><code>.div/para</code></td>
<td>Selects the <code>&lt;para&gt;</code> element children of the <code>&lt;div&gt;</code> element children of the context node.</td>
</tr>
</tbody>
</table>

The following is a summary of some of the abbreviations:

### Unabbreviated          Abbreviated

<table>
<thead>
<tr>
<th>Location Path</th>
<th>Abbreviated</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>child::*</code></td>
<td><code>*</code></td>
</tr>
<tr>
<td><code>attribute::*</code></td>
<td><code>@*</code></td>
</tr>
<tr>
<td><code>/descendant-or-self::*node()</code></td>
<td><code>//</code></td>
</tr>
</tbody>
</table>
In This Section

The following topics discuss in detail the syntax and usage of the pieces of location paths.

- Location Steps
- Axes
- Node Tests
- Predicates
- Location Path Examples
Location Steps

A location step selects a set of nodes (a node-set) relative to the context node.

There are three parts to a location step: an optional axis, a node test, and an optional predicate. The syntax for a location step is the axis name followed by a double colon, then the node test, and finally zero or more predicates, each in square brackets. The most basic form of this syntax is as follows:

\[
\text{axis}::\text{nodetest}[[\text{predicate}]]
\]

\textit{axis}

Specifies the tree relationship between the context node and the nodes to be selected by the location step. In other words, the axis indicates the general direction that the location step proceeds from the context node. In a location step, the axis is optional. If omitted, the axis defaults to \texttt{child:}.

Furthermore, several axes have shortcut forms; for instance, the ampersand (@) character is a shortcut for the attribute axis.

\textit{nodetest}

Specifies the node type or expanded name of the nodes to be initially selected by the location step. Basically, the node test indicates which node(s), from among all nodes on the indicated axis, to consider as candidates, that is, potential, matches for the location step.

\textit{predicate}

Uses an XPath expression (condition to be met) to further refine the set of nodes selected by the location step. The predicate is a filter, specifying a selection criterion for further refining the list of candidate nodes. The predicate is optional. If there is no predicate, there are no square brackets ([ and ]) in the location step.
Selecting Nodes

The node-set selected by a location step results from generating an initial node-set based on the relationship between the axis and node test, and then filtering that initial node-set by each of the predicates in turn.

The initial node-set consists of those nodes that meet the following two criteria:

- The nodes have the relationship to the context node specified by the axis.
- The nodes have the node type and expanded name specified by the node test.

XPath then uses the first predicate in the location step to filter the initial node-set to generate a new node-set. XPath then uses the second predicate to filter the node-set resulting from the first predicate. This filtering process repeats until XPath has evaluated all the predicates. The node-set that results after applying all the predicates is the node-set selected by the location step.

**Note** Because the axis affects the evaluation of the expression in each predicate, the semantics of a predicate is defined with respect to the specified axis.

Some sample location steps using the complete syntax are shown in the following table.

<table>
<thead>
<tr>
<th>Location step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::*[position()=1]</td>
<td>Locates the first child node of the context node</td>
</tr>
<tr>
<td>ancestor-or-self::*[@catdate=&quot;2000-12-31&quot;]</td>
<td>Locates all ancestors of any &lt;book&gt; child of the context node, as well as the &lt;book&gt; child itself, as long as the element in question has a catdate attribute with the value &quot;2000-12-31&quot;.</td>
</tr>
<tr>
<td>//self::parent[name()=&quot;book&quot;]</td>
<td>Locates any node in the document whose parent node is named &quot;book&quot;, or any node descended from the context node whose name is &quot;author&quot;.</td>
</tr>
</tbody>
</table>
See Also

Aaxes | Node Tests | Predicates
Axes

A location path uses an axis to specify the relationship between the nodes selected by the location step and the context node.

<table>
<thead>
<tr>
<th>Axes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor::</td>
<td>The ancestors of the context node. The ancestors of the context node consist of the parent of the context node and the parent's parent and so on; thus the ancestor:: axis always includes the root node unless the context node is the root node.</td>
</tr>
<tr>
<td>ancestor-or-self::</td>
<td>The context node and its ancestors. The ancestor-or-self:: axis always includes the root node.</td>
</tr>
<tr>
<td>attribute::</td>
<td>The attributes of the context node. This axis will be empty unless the context node is an element.</td>
</tr>
<tr>
<td>child::</td>
<td>The children of the context node. A child is any node immediately below the context node in the tree. However, neither attribute or namespace nodes are considered children of the context node.</td>
</tr>
<tr>
<td>descendant::</td>
<td>The descendants of the context node. A descendant is a child or a child of a child and so on; thus the descendant:: axis never contains attribute or namespace nodes.</td>
</tr>
<tr>
<td>descendant-or-self::</td>
<td>The context node and its descendants.</td>
</tr>
</tbody>
</table>
following::
All nodes that are after the context node in the tree, excluding any descendants, attribute nodes, and namespace nodes.
All the following siblings of the context node.

following-sibling::
The following-sibling:: axis identifies just those children of a parent node who appear in the tree after the context node. This axis excludes all other children that appear before the context node.
If the context node is an attribute node or namespace node, the following-sibling:: axis is empty.

namespace::
The namespace nodes of the context node.
There is a namespace node for every namespace which is in scope for the context node.
This axis will be empty unless the context node is an element.

parent::
The parent of the context node, if there is one.
The parent is the node immediately above the context node in the tree.
All nodes that are before the context node in the tree, excluding any ancestors, attribute nodes, and namespace nodes.

preceding::
One way to think of the preceding axis is all nodes whose content occurs in their entirety before the start of the context node.
All the preceding siblings of the context node.
The preceding-sibling:: axis identifies just those children of a parent node who appear in the tree before the context node. This axis excludes all other children that
sibling::

appear after the context node.

If the context node is an attribute node or namespace node, the preceding-sibling:: axis is empty.

self::

Just the context node itself.

Remarks

namespace

Each element has an associated set of namespace nodes, one for each distinct namespace prefix that is in scope for the element (including the xml prefix, which is implicitly declared by the XML Namespaces Recommendation) and one for the default namespace if one is in scope for the element. The element is the parent of each of these namespace nodes; however, a namespace node is not a child of its parent element. Elements never share namespace nodes: if one element node is not the same node as another element node, then none of the namespace nodes of the one element node will be the same node as the namespace nodes of another element node. This means that an element will have a namespace node:

- For every attribute on the element whose name starts with xmlns:
- For every attribute on an ancestor element whose name starts with xmlns: —unless the element itself or a nearer ancestor redeclares the prefix.
- For an xmlns attribute, if the element or some ancestor has an xmlns attribute, and the value of the xmlns attribute for the nearest such element is non-empty.

The <sample> element in the following instance:

```xml
<A xmlns="urn:A">
  <B xmlns:test="sample">
    <sample xmlns:sampletest="sampletest"/>
  </B>
</A>
```

contains the following namespace nodes:

```xml
xmlns="urn:A"
```
The `<sample>` element in the following instance:

```xml
<A xmlns="urn:A">
  <B xmlns="" xmlns:sample="sample">
    <sample xmlns:sampletest="sampletest" xmlns:bar="test"/>
  </B>
</A>
```

contains the following namespace nodes:

```xml
xmlns:sampletest="sampletest"
xmlns:test="test"
```

**Note** An attribute `xmlns=""` undeclares the default namespace.

A namespace node has an expanded-name: the local part is the namespace prefix (this is empty if the namespace node is for the default namespace); the namespace Uniform Resource Identifier (URI) is always null.

The string-value of a namespace node is the namespace URI that is being bound to the namespace prefix; if it is relative, it must be resolved just like a namespace URI in an expanded-name.

**Examples**

**following::**

The examples for the `following::` axis refer to this instance document:

```xml
<A>
  <B>
    <C>sample</C>
    <C>sample2</C>
  </B>
  <B>
    <C>sample</C>
    <C>sample2</C>
    <D>sample3</D>
  </B>
</A>
```
<table>
<thead>
<tr>
<th>Query</th>
<th>Returned nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/B[1]/following::*</td>
<td><code>&lt;B&gt;</code>&lt;br&gt;<code>&lt;C&gt;sample&lt;/C&gt;</code>&lt;br&gt;<code>&lt;C&gt;sample2&lt;/C&gt;</code>&lt;br&gt;<code>&lt;D&gt;sample3&lt;/D&gt;</code>&lt;br&gt;<code>&lt;/B&gt;</code>&lt;br&gt;<code>&lt;C&gt;sample&lt;/C&gt;</code>&lt;br&gt;<code>&lt;C&gt;sample2&lt;/C&gt;</code>&lt;br&gt;<code>&lt;D&gt;sample3&lt;/D&gt;</code>&lt;br&gt;<code>&lt;/B&gt;</code></td>
</tr>
<tr>
<td>A/B[1]/following::node()</td>
<td><code>&lt;B&gt;</code>&lt;br&gt;<code>&lt;C&gt;sample&lt;/C&gt;</code>&lt;br&gt;<code>sample</code>&lt;br&gt;<code>&lt;C&gt;sample2&lt;/C&gt;</code>&lt;br&gt;<code>sample2</code>&lt;br&gt;<code>&lt;D&gt;sample3&lt;/D&gt;</code>&lt;br&gt;<code>sample3</code>&lt;br&gt;<code>&lt;/B&gt;</code></td>
</tr>
</tbody>
</table>

**preceding::**

The example for the preceding:: axis refer to this instance document:

```xml
<A><B><C test="sampletest">sample</C><C>sample2</C><B><C>sample</C><C>sample2</C><D>sample3</D><B><B></B></B></A>
```

<table>
<thead>
<tr>
<th>Query</th>
<th>Returned nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;B&gt;</code>&lt;br&gt;<code>&lt;C test=&quot;sampletest&quot;&gt;sample&lt;/C&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
The example for the following-sibling:: axis refers to this instance document:

```xml
<A>
  <B>
    <C test="sampletest">sample</C>
    <C>sample2</C>
  </B>
  <B>
    <C>sample</C>
    <C>sample2</C>
    <D>sample3</D>
  </B>
</A>
```

<table>
<thead>
<tr>
<th>Query</th>
<th>Returned nodes</th>
</tr>
</thead>
</table>
| A/B[1]/following-sibling::* | `<B>
  <C>sample</C>
  <C>sample2</C>
  <D>sample3</D>
</B>` |

The example for the preceding-sibling:: axis refers to this instance document:

```xml
<A>
  <B>
    <C test="sampletest">sample</C>
    <C>sample2</C>
  </B>
  <B>
    <C>sample</C>
  </B>
</A>
```
Query | Returned nodes
--- | ---
A/B[2]/preceding-sibling::* | `<C test="sampletest">sample</C>`
 | `<C>sample2</C>`

See Also

[Location Steps] | [Node Tests] | [Predicates]
Node Tests

The node test is the only required portion of an XPath location step. As such, understanding it is critical to successfully employing XPath expressions.

There are three general types of node test:

- A name test, which uses an expanded name and the relationship of that name to the specified axis to identify the nodes to be selected.
- A node type test, which selects nodes based strictly upon their node type.
- A targeted processing instruction test, which selects only those processing instruction nodes that match the specified type.

See Also

Location Steps | Axes | Predicates
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Name Tests

A name test is the most common form of a node test since it allows you to explicitly specify the name of the nodes that you want to select. However, just specifying a node's name does not automatically result in the selection of the appropriate node since a document tree may have different node types with the same name. To select the appropriate node, a name test selects only those nodes whose name matches the specified name and whose node type matches the principal node type of the specified axis.

The following table lists the principal node type for the various axes.

<table>
<thead>
<tr>
<th>Axis</th>
<th>Principal node type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any axis other than an attribute:: axis or the namespace:: axis (that is, an axis that can contain elements)</td>
<td>element</td>
</tr>
<tr>
<td>attribute</td>
<td>attribute</td>
</tr>
<tr>
<td>namespace</td>
<td>namespace</td>
</tr>
</tbody>
</table>

The name specified in a name test may be one of three types: an asterisk (*), a QName, or the expression, NCName::* . The following table shows examples of how each of these types of names works in conjunction with the specified axis to locate a particular node-set.

<table>
<thead>
<tr>
<th>Name</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>* (asterisk)</td>
<td>True for any node of the principal node type.</td>
<td>ancestor::* selects all ancestors of the context node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attribute::* selects all attributes of the context node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>namespace::* selects all namespaces of the context node.</td>
</tr>
</tbody>
</table>
QName

True for any node of the principal node type whose expanded name equals the expanded name specified by the QName.

NCName:*  

True for any node of the principal type whose expanded name has the namespace URI to which the NCName expands, regardless of the local name.

Remarks

When the node test is a QName, XPath must first expand the QName based upon the context namespace declarations in the XML file. This expansion process is identical to the expansion process for element names in start and end tags, except that it does not use the default namespace declared with $xmlns$. If the QName does not have a prefix, XPath looks for those nodes whose local names match the given QName and whose namespace URI is null. A QName causes an error if the QName's prefix does not correspond to any of the context namespace declarations in the file.

When the node test is NCName:*, XPath expands the NCName in the same way as it would expand the prefix of a QName. Similarly, an NCName:* causes an error if the NCName part does not correspond to any of prefixes contained in the context namespace declarations in the file.

See Also

Node Tests | Node Type Tests | Targeted Processing Instruction Tests
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Node Type Tests

To select nodes types other than, or in addition to, elements nodes, use a node type test. The effect of using a node-type test is to override the principal node type for the given axis. For example, descendant::text() locates all text nodes descended from the context node, even though the principal node type for the descendant axis is element.

There are four node type tests available, as shown in the following table.

<table>
<thead>
<tr>
<th>Node type</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment()</td>
<td>True for a comment node.</td>
<td>following::comment() selects all the comment nodes that appear after the context node.</td>
</tr>
<tr>
<td>node()</td>
<td>True for a node of any type.</td>
<td>preceding::node() selects all the nodes that appear before the context node.</td>
</tr>
<tr>
<td>processing-instruction()</td>
<td>True for a processing instruction node.</td>
<td>self::processing instruction() selects all the processing instruction nodes within the context node.</td>
</tr>
<tr>
<td>text()</td>
<td>True for a text node.</td>
<td>child::text() selects the text nodes that are children of the context node.</td>
</tr>
</tbody>
</table>

**Note** The processing-instruction() test may have an argument that is literal; in this case, it is True for any processing instruction that has a name equal to the value of the literal. For more information, see [Targeted Processing Instruction Tests](#).
names are comment, text, processing-instruction, or node. When an XPath application encounters the double parentheses, it knows that this is a node type test, not a name test for nodes with those names. Similarly, node type tests are not XPath functions, references to which also contain parentheses, empty or otherwise.

See Also

Node Tests | Name Tests | Targeted Processing Instruction Tests
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Targeted Processing Instruction Tests

A node test can be a processing instruction that specifies a target. The syntax for such a node test is as follows:

`processing-instruction("target")`

By specifying a target, the processing instruction test locates all the processing instruction nodes that match that target. For example, the following node test locates all the processing instruction nodes in a document that specify an XSLT file.

`/child::processing-instruction("xml-stylesheet")`

See Also

Node Tests | Name Tests | Node Type Tests
Predicates

A predicate is an XPath expression that filters a node-set with respect to an axis and produces a new node-set. This filtering process involves sequentially evaluating the predicate against each node in the node set. Each time that the predicate is evaluated against a node:

- The context node is the node currently being evaluated.
- The context size is the number of nodes in the node-set being evaluated.
- The context position is the position of the context node in the node-set.

This last context, that of the context node in the node-set, is relative to the direction that the axis specified in the location step navigates the document tree. Typically, an axis navigates the tree in either a forward or reverse direction:

- A forward axis is an axis that contains the context node or nodes that are after the context node. The child::, descendant::, descendant-or-self::, following::, and following-sibling:: axes are forward axes. These forward axes number the nodes in the node-set in document order, starting with the first position as 1.
- A reverse axis is an axis that contains the context node or nodes that are before the context node. The ancestor::, ancestor-or-self::, preceding::, and preceding-sibling:: axes are reverse axes. These reverse axes number the nodes in the node-set in reverse document order, starting with the first position as 1.

As to the remaining axes, the self:: and parent:: axes return a single node. Therefore, the designation of forward or reverse does not make sense for these two axes. There is no order defined for the attribute:: and namespaces:: axes; so they too are neither forward nor reverse axes.
Selecting the Nodes

A predicate expression evaluates to either a numeric or Boolean value.

If the predicate results in a number, XPath compares that number to the context position of the context node. If the number and the position are the same (that is, the context node is in the appropriate position in the tree), the predicate evaluates to true and the context node is included in the new node-set. Otherwise, the context node is excluded from the new node-set.

If the predicate does not result in a number, XPath uses the boolean function to convert the result to a Boolean value. For example, the predicate [genre='Computer'] results in a node-set. If the context node has a child <genre> element with the content Computer, this predicate evaluates to true and the context node is included in the new node-set. Otherwise, the context node is excluded from the new node-set.

Note  A numeric predicate [x] is equivalent to the Boolean predicate [position()=x].

See Also

Location Steps | Axes | Node Tests
Location Path Examples

The following examples illustrate the use of location paths.

Path

child::node()

attribute::name

attribute::*

descendant::para

ancestor::div

ancestor-or-self::div
child::*[self::chapter or self::appendix]

child::*[self::chapter or self::appendix][position()=last()]

See Also

 XPath Examples
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
## XPath Examples

This topic reviews the syntax examples that appear throughout the XPath Reference. All are based on the Sample XML File for XPath Syntax ([inventory.xml]). For an example of using an XPath expression in a test file, see Example of Unions (|), at the bottom of this topic.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>./author</td>
<td>All <code>&lt;author&gt;</code> elements within the current context. Note that this is equivalent to the expression in the next row.</td>
</tr>
<tr>
<td>author</td>
<td>All <code>&lt;author&gt;</code> elements within the current context.</td>
</tr>
<tr>
<td>first.name</td>
<td>All <code>&lt;first.name&gt;</code> elements within the current context.</td>
</tr>
<tr>
<td>/bookstore</td>
<td>The document element (&lt;bookstore&gt;) of this document.</td>
</tr>
<tr>
<td>//author</td>
<td>All <code>&lt;author&gt;</code> elements in the document.</td>
</tr>
<tr>
<td>book[/bookstore/@specialty = @style]</td>
<td>All <code>&lt;book&gt;</code> elements whose style attribute value is equal to the specialty attribute value of the <code>&lt;bookstore&gt;</code> element at the root of the document.</td>
</tr>
<tr>
<td>author/first-name</td>
<td>All <code>&lt;first-name&gt;</code> elements that are children of an <code>&lt;author&gt;</code> element.</td>
</tr>
<tr>
<td>bookstore//title</td>
<td>All <code>&lt;title&gt;</code> elements one or more levels deep in the <code>&lt;bookstore&gt;</code> element (arbitrary descendants). Note that this is different from the expression in the next row.</td>
</tr>
<tr>
<td>bookstore/*/title</td>
<td>All <code>&lt;title&gt;</code> elements that are grandchildren of <code>&lt;bookstore&gt;</code></td>
</tr>
</tbody>
</table>
All `<emph>` elements anywhere inside `<excerpt>` children of `<book>` elements, anywhere inside the `<bookstore>` element.

All `<title>` elements one or more levels deep in the current context. Note that this situation is essentially the only one in which the period notation is required.

All elements that are the children of `<author>` elements.

All `<last-name>` elements that are grandchildren of `<book>` elements.

All grandchildren elements of the current context.

All elements with the specialty attribute.

The style attribute of the current context.

The exchange attribute on `<price>` elements within the current context. Returns an empty node set, because attributes do not contain element children. This expression is allowed by the XML Path Language (XPath) grammar, but is not strictly valid.

All `<book>` elements with style attributes, of the current context.

The style attribute for all `<book>` elements of the current context.

All attributes of the current element context.

All `<first-name>` elements in the current context node. Note that this
./first-name

first-name is equivalent to the expression in the next row.

author[1]
The first <author> element in the current context node.

author[first-name][3]
The third <author> element that has a <first-name> child.

my:book
The <book> element from the my namespace.

my:*
All elements from the my namespace.

@my:*
All attributes from the my namespace (this does not include unqualified attributes on elements from the my namespace).

Note that indexes are relative to the parent. Consider the following data:

```xml
<x>
  <y/>
  <y/>
</x>
<x>
  <y/>
  <y/>
</x>
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>x/y[1]</td>
<td>The first &lt;y&gt; child of each &lt;x&gt;. This is equivalent to the expression in the next row.</td>
</tr>
<tr>
<td>x/y[position() = 1]</td>
<td>The first &lt;y&gt; child of each &lt;x&gt;.</td>
</tr>
<tr>
<td>(x/y)[1]</td>
<td>The first &lt;y&gt; from the entire set of &lt;y&gt; children of &lt;x&gt; elements.</td>
</tr>
</tbody>
</table>
The remaining examples refer to the Sample XML file for XPath.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>book[last()]</code></td>
<td>The last <code>&lt;book&gt;</code> element of the current context node.</td>
</tr>
<tr>
<td><code>book/author[last()]</code></td>
<td>The last <code>&lt;author&gt;</code> child of each <code>&lt;book&gt;</code> element of the current context node.</td>
</tr>
<tr>
<td><code>(book/author)[last()]</code></td>
<td>The last <code>&lt;author&gt;</code> element from the entire set of <code>&lt;author&gt;</code> children of <code>&lt;book&gt;</code> elements of the current context node.</td>
</tr>
<tr>
<td><code>book[excerpt]</code></td>
<td>All <code>&lt;book&gt;</code> elements that contain at least one <code>&lt;excerpt&gt;</code> element child.</td>
</tr>
<tr>
<td><code>book[excerpt]/title</code></td>
<td>All <code>&lt;title&gt;</code> elements that are children of <code>&lt;book&gt;</code> elements that also contain at least one <code>&lt;excerpt&gt;</code> element child.</td>
</tr>
<tr>
<td><code>book[excerpt]/author[degree]</code></td>
<td>All <code>&lt;author&gt;</code> elements that contain at least one <code>&lt;degree&gt;</code> element child, and that are children of <code>&lt;book&gt;</code> elements that also contain at least one <code>&lt;excerpt&gt;</code> element.</td>
</tr>
</tbody>
</table>
All `<book>` elements that contain `<author>` children that in turn contain at least one `<degree>` child.

All `<author>` elements that contain at least one `<degree>` element child and at least one `<award>` element child.

All `<author>` elements that contain at least one `<degree>` element child and at least one `<award>` element child.

All `<author>` elements that contain at least one `<degree>` element child and at least one `<award>` element child.

All `<author>` elements that contain at least one `<degree>` element child and at least one `<award>` element child.

All `<author>` elements that contain at least one `<degree>` element child and that contain no `<publication>` element children.

All `<author>` elements that contain at least one `<degree>` element child and that contain no `<publication>` element children.

All `<author>` elements that contain at least one `<degree>` element child and that contain no `<publication>` element children.
author[last-name = "Bob"]

author[last-name[1] = "Bob"]

author[last-name[position()=1]= "Bob"]

degree[@from != "Harvard"]

author[. = "Matthew Bob"]

author[last-name = "Bob" and ../price &gt; 50]

book[position() &lt;= 3]
author[not(last-name = "Bob")]

All <author> elements that do not contain <last-name> child elements with the value Bob.

author[first-name = "Bob"]

All <author> elements that have at least one <first-name> child with the value Bob.

author[* = "Bob"]

All author elements containing any child element whose value is Bob.

author[last-name = "Bob" and first-name = "Joe"]

All <author> elements that have a <last-name> child element with the value Bob and a <first-name> child element with the value Joe.

price[@intl = "Canada"]

All <price> elements in the context node which have an intl attribute equal to "Canada".

degree[position() &lt; 3]

The first two <degree> elements that are children of the context node.

p/text()[2]

The second text node in each <p> element in the context node.

ancestor::book[1]

The nearest <book> ancestor of the context node.
ancestor::book[author][1]

ancestor::author[parent::book][1]

The nearest <book> ancestor of the context node and this <book> element has an <author> element as its child.
The nearest <author> ancestor in the current context and this <author> element is a child of a <book> element.
Example of Unions ( | )

To demonstrate the union operation, we use the following XPath expression:

{x | y/x}

selects all the <x> elements whose values are green or blue in the following XML file:

**XML File (data1.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="union.xsl"?>
<root>
  <x>green</x>
  <y>
    <x>blue</x>
    <x>blue</x>
  </y>
  <z>
    <x>red</x>
    <x>red</x>
  </z>
  <x>green</x>
</root>
```

**XSLT File (union.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="root">
  <xsl:for-each select="x | y/x">
    <xsl:value-of select="."/>
    <xsl:if test="not(position()=last())">,</xsl:if>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

**Formatted Output**

green,blue,blue,green
**Processor Output**

```xml
<?xml version="1.0" encoding="UTF-16"?>
green,blue,blue,green
```

**See Also**

[Sample XML File for XPath Syntax (inventory.xml)](inventory.xml) | [Location Path Examples](locationpathexamples.html)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Sample XML File for XPath Syntax (inventory.xml)

This XML file is used by some topics in the XPath Reference documentation. This file represents a fragment of a bookstore inventory database.

XML File (inventory.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="myfile.xsl" ?>
<bookstore specialty="novel">
  <book style="autobiography">
    <author>
      <first-name>Joe</first-name>
      <last-name>Bob</last-name>
      <award>Trenton Literary Review Honorable Mention</award>
    </author>
    <price>12</price>
  </book>
  <book style="textbook">
    <author>
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
      <publication>Selected Short Stories of</publication>
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
    </author>
    <editor>
      <first-name>Britney</first-name>
      <last-name>Bob</last-name>
    </editor>
    <price>55</price>
  </book>
  <magazine style="glossy" frequency="monthly">
    <price>2.50</price>
    <subscription price="24" per="year"/>
  </magazine>
  <book style="novel" id="myfave">
    <author>
      <first-name>Toni</first-name>
      <last-name>Bob</last-name>
      <degree from="Trenton U">B.A.</degree>
    </author>
  </book>
</bookstore>
```
<degree from="Harvard">Ph.D.</degree>
<award>Pulitzer</award>
<publication>Still in Trenton</publication>
<publication>Trenton Forever</publication>

<price intl="Canada" exchange="0.7">6.50</price>

<excerpt>
  <p>It was a dark and stormy night.</p>
  <p>But then all nights in Trenton seem dark and stormy to someone who has gone through what <emph>I</emph> have.</p>
  <definition-list>
    <term>Trenton</term>
    <definition>misery</definition>
  </definition-list>
</excerpt>

<my:book xmlns:my="uri:mynamespace" style="leather" price="29.50">
  <my:title>Who's Who in Trenton</my:title>
  <my:author>Robert Bob</my:author>
</my:book>

See Also

XPath Examples | XPath Syntax
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
XPath Functions

You can use XPath functions to refine XPath queries and enhance the programming power and flexibility of XPath.

The functions are divided into the following groups.

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-Set</td>
<td>Takes a node-set argument, returns a node-set, or returns/provides information about a particular node within a node-set.</td>
</tr>
<tr>
<td>String</td>
<td>Performs evaluations, formatting, and manipulation on string arguments.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Evaluates the argument expressions to obtain a Boolean result.</td>
</tr>
<tr>
<td>Number</td>
<td>Evaluates the argument expressions to obtain a numeric result.</td>
</tr>
<tr>
<td>Microsoft XPath Extension Functions</td>
<td>Microsoft extension functions to XPath that provide the ability to select nodes by XSD type. Also includes string comparison, number comparison, and date/time conversion functions.</td>
</tr>
</tbody>
</table>

Each function in the function library is specified using a function prototype that provides the return type, function name, and argument type. If an argument type is followed by a question mark, the argument is optional; otherwise, the argument is required. Function names are case-sensitive.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Node-Set Functions

Node-set functions take a node-set argument. They return a node-set, or information about a particular node within a node-set.

Node-Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count</code></td>
<td>Returns the number of nodes in the node-set argument.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>Selects elements by their unique ID.</td>
</tr>
<tr>
<td><code>last</code></td>
<td>Returns a number equal to context size of the expression evaluation context.</td>
</tr>
<tr>
<td><code>local-name</code></td>
<td>Returns the local part of the expanded name of the node in the node-set argument that is first in document order.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Returns a string containing a QName representing the expanded name of the node in the node-set argument that is first in document order.</td>
</tr>
<tr>
<td><code>namespace-uri</code></td>
<td>Returns the namespace Uniform Resource Identifier (URI) of the expanded name of the node in the node-set argument that is first in document order.</td>
</tr>
<tr>
<td><code>position</code></td>
<td>Returns the index number of the node within the parent.</td>
</tr>
</tbody>
</table>

See Also

Processing Node-Sets by Using Node-Set Functions | Qualified Names
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
count Function

Returns the number of nodes in the node-set argument.

\[
\text{number \ count(node-set)}
\]

Examples

XML File (test.xml)

```xml
<?xml version="1.0"?>
<test>
  <x a="1">
    <x a="2">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
  <x a="1">
    <x a="2">
      <y>y21</y>
      <y>y22</y>
    </x>
  </x>
  <x a="1">
    <y>y11</y>
    <y>y12</y>
  </x>
  <y>y03</y>
  <y>y04</y>
</test>
```

XSLT File (test.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text" />
```
<xsl:template match="/">
  //x,  <xsl:value-of select="count(//x)"/>
  //x[1],  <xsl:value-of select="count(//x[1])"/>
  //x/y,  <xsl:value-of select="count(//x/y)"/>
  //x/y[1],  <xsl:value-of select="count(//x/y[1])"/>
  //x[1]/y[1],  <xsl:value-of select="count(//x[1]/y[1])"/>
</xsl:template>

**Output**

The XSLT stylesheet, when applied to the XML file above, produces the following results:

<table>
<thead>
<tr>
<th>node-set</th>
<th>count(node-set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>//x</td>
<td>7</td>
</tr>
<tr>
<td>//x[1]</td>
<td>4</td>
</tr>
<tr>
<td>//x/y</td>
<td>8</td>
</tr>
<tr>
<td>//x/y[1]</td>
<td>4</td>
</tr>
<tr>
<td>//x[1]/y[1]</td>
<td>2</td>
</tr>
</tbody>
</table>

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#) | [Processing Node-Sets by Using Node-Set Functions](#)
**id Function**

Selects elements by their unique ID.

\[ \textit{node-set id(object)} \]

**Remarks**

When the argument is of type node-set, the result is the union of the result of applying `id()` to the string value of each of the nodes in the node-set argument.

When the argument is of any other type, it is converted to a string and then split into a white space–separated list of tokens (white space is any sequence of characters matching the production); the result is a node-set containing the elements in the same document as the context node that have a unique ID equal to any of the tokens in the list.

**Example**

**XML FILE (test.xml)**

```xml
<?xml version="1.0"?>
<!DOCTYPE test [  
  <!ELEMENT test (x+)>
  <!ELEMENT x (x+, y+)>
  <!ATTLIST x  
    a ID #REQUIRED>
  <!ELEMENT y ANY>
]>  
<test>  
  <x a="a11">  
    <x a="a21">  
      <x a="a31">  
        <y>y31</y>  
        <y>y32</y>  
      </x>  
    </x>  
  </x>  
  <x a="a12">  
    <x a="a22">  
      <y>y21</y>  
    </x>  
  </x>  
</test>
```
XSLT File (test.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" omit-xml-declaration="yes"/>
    <!-- suppress text nodes not covered in subsequent template -->
    <xsl:template match="text()"/>
    <xsl:template match="*">
        <xsl:element name="{name()}">
            <xsl:apply-templates select="*|@*"/>
            <xsl:if test="text()">
                <xsl:value-of select="."/>
            </xsl:if>
        </xsl:element>
    </xsl:template>
    <xsl:template match="@*">
        <xsl:attribute name="{name()}">
            <xsl:value-of select="."/>
        </xsl:attribute>
    </xsl:template>
    <xsl:template match="/test">
        <xsl:apply-templates select="id('a21')"/>
        <xsl:apply-templates select="id('a11')//y[1]"/>
    </xsl:template>
</xsl:stylesheet>

Output
The XSLT stylesheet, when applied to the XML file above results in the following node-set:

```xml
<x a="a21">
  <x a="a31">
    <y>y31</y>
    <y>y32</y>
  </x>
</x>
and
<y>y31</y>
```

See Also

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
last Function

Returns a number equal to context size of the expression evaluation context.

\[ \text{number} \text{ last()} \]

Remarks

The following finds the last \texttt{author} child of each \texttt{book} element:

\texttt{book/author[last()]}  

Example

In this example we illustrate how to use the \texttt{last()} function to select the last \texttt{y} element contained in each \texttt{x} element of the following XML document.

**XML FILE (test.xml)**

```xml
<?xml version="1.0"?>
<!DOCTYPE test [
  <!ELEMENT test (x+)>
  <!ELEMENT x (x+, y+)>
  <!ATTLIST x
    a ID #REQUIRED>
  <!ELEMENT y ANY>
]>
<test>
  <x a="a11">
    <x a="a21">
      <x a="a31">
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
  <x a="a12">
    <x a="a22">
      <y>y21</y>
      <y>y22</y>
    </x>
  </x>
</test>
```
XSLT File (test.xsl)

Pay attention to the instruction in bold in this XSLT stylesheet.

```xml
<?xml version='1.0'?><xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <!-- Suppress text nodes not covered in subsequent template rule -->
   <xsl:template match="text()"/>

   <xsl:template match="*"><xsl:element name="{name()}">
     <xsl:apply-templates select="*|@*"/>
     <xsl:if test="text()">
       <xsl:value-of select="."/>
     </xsl:if></xsl:element>
   </xsl:template>

   <xsl:template match="@*"><xsl:attribute name="{name()}">
     <xsl:value-of select="."/>
   </xsl:attribute></xsl:template>

   <xsl:template match="/test">
     <xsl:apply-templates select="//x/y[last()]"/>
   </xsl:template>
</xsl:stylesheet>
```

Output

The XSLT stylesheet, when applied to the XML file above results in the
following node-set:

<y>y32</y>
<y>y22</y>
<y>y12</y>
<y>y04</y>

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
local-name Function

Returns the local part of the expanded name of the node in the node-set argument that is first in document order.

\[
string \, local-name(\text{node-set?})
\]

Remarks

An expanded name typically consists of a namespace URI (or associated prefix), a colon (:) and a local part. For example, \texttt{b:author} is an expanded name, where \texttt{b} is the namespace URI prefix and \texttt{author} is the local part. So applying the \texttt{local-name()} function to this node returns \texttt{author}. When a node has no expanded name, such as \texttt{author}, applying the \texttt{local-name()} function on this node returns the node name as is, i.e., \texttt{author}.

If the node-set argument is omitted, it defaults to a node-set with the context node as its only member.

Sample

XML File (data.xml)

```xml
<?xml-stylesheet type="text/xsl" href="sample.xsl"?>
<b:catalog xmlns:b="x-schema:book-schema.xml">
  <b:book id="bk101">
    <b:author>Gambardella, Matthew</b:author>
    <b:genre>Computer</b:genre>
    <b:price>44.95</b:price>
    <b:publish_date>2000-10-01</b:publish_date>
    <b:description>An in-depth look at creating applications with</b:description>
  </b:book>
  <b:book id="bk102">
    <b:author>Ralls, Kim</b:author>
    <b:title>Midnight Rain</b:title>
    <b:genre>Fantasy</b:genre>
    <b:price>5.95</b:price>
    <b:publish_date>2000-12-16</b:publish_date>
    <b:description>A former architect battles corporate zombies, a
```
XSLT File (sample.xsl)

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h3>local-name() Function</h3>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="*">
    <xsl:value-of select="local-name()"/>
    \t=\t<xsl:value-of select="text()"/>
    <xsl:apply-templates select="*"/>
  </xsl:template>
</xsl:stylesheet>
```

Auxiliary XSLT File (book-schema.xml)

```xml
<Schema name="books" xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="author"/>
  <ElementType name="title"/>
  <ElementType name="genre"/>
  <ElementType name="price"/>
  <ElementType name="publish_date"/>
  <ElementType name="description"/>
  <AttributeType name="id" dt:type="id"/>

  <ElementType name="catalog">
    <element type="book"/>
  </ElementType>

  <ElementType name="book" model="closed" content="eltOnly">
    <attribute type="id"/>
    <element type="author"/>
    <element type="title"/>
    <element type="genre"/>
    <element type="price"/>
    <element type="publish_date"/>
    <element type="description"/>
  </ElementType>
</Schema>
```
local-name() Function

catalog =
book =
author = Gambardella, Matthew
title = XML Developer's Guide
genre = Computer
price = 44.95
publish_date = 2000-10-01
description = An in-depth look at creating applications with XML.

book =
author = Ralls, Kim
title = Midnight Rain
genre = Fantasy
price = 5.95
publish_date = 2000-12-16
description = A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.

Processor Output

<html>
<body>
<h3>local-name() Function</h3>
catalog = <br>book = <br>author = Gamb
See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
**name Function**

Returns a string containing a QName representing the expanded name of the node in the *node-set* argument that is first in document order.

```
string name(node-set?)
```

**Remarks**

The QName must represent the expanded name with respect to the namespace in effect on the node. This need not be the case if there are namespace declarations in effect on the node that associate multiple prefixes with the same namespace. If the node-set argument is empty or the first node has no expanded name, an empty string is returned. If the argument it omitted, it defaults to a node-set with the context node as its only member.

**Example**

**XML File (bcat.xml)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="sample.xsl"?>
<b:catalog xmlns:b="x-schema:book-schema.xml">
  <b:book id="bk101">
    <b:author>Gambardella, Matthew</b:author>
    <b:genre>Computer</b:genre>
    <b:price>44.95</b:price>
    <b:publish_date>2000-10-01</b:publish_date>
    <b:description>An in-depth look at creating applications with XML.</b:description>
  </b:book>
  <b:book id="bk102">
    <b:author>Ralls, Kim</b:author>
    <b:title>Midnight Rain</b:title>
    <b:genre>Fantasy</b:genre>
    <b:price>5.95</b:price>
    <b:publish_date>2000-12-16</b:publish_date>
    <b:description>A former architect battles corporate zombies, a</b:description>
  </b:book>
</b:catalog>
```
XSLT File (sample.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
<html>
  <body>
    <h3>name() Function</h3>
    <xsl:apply-templates />
  </body>
</html>
</xsl:template>

<xsl:template match="*">
  <xsl:value-of select="name()"/>
  =
  <xsl:value-of select="text()"/>
  <xsl:apply-templates select="*"/>
</xsl:template>

</xsl:stylesheet>

Auxiliary XSLT File (book-schema.xml)

<Schema name="books" xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="author"/>
  <ElementType name="title"/>
  <ElementType name="genre"/>
  <ElementType name="price"/>
  <ElementType name="publish_date"/>
  <ElementType name="description"/>
  <AttributeType name="id" dt:type="id"/>

  <ElementType name="catalog">
    <element type="book"/>
  </ElementType>

  <ElementType name="book" model="closed" content="eltOnly">
    <attribute type="id"/>
    <element type="author"/>
    <element type="title"/>
    <element type="genre"/>
    <element type="price"/>
    <element type="publish_date"/>
    <element type="description"/>
  </ElementType>
</Schema>
Formatted Output

**name() Function**

b:catalog =

b:book =

b:author = Gambardella, Matthew

b:title = XML Developer's Guide

b:genre = Computer

b:price = 44.95

b:publish_date = 2000-10-01

b:description = An in-depth look at creating applications with XML.

b:book =

b:author = Ralls, Kim

b:title = Midnight Rain

b:genre = Fantasy

b:price = 5.95

b:publish_date = 2000-12-16

b:description = A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.

**Processor Output**

<html>
<body>

</body>
</html>
See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
namespace-uri Function

Returns the namespace Uniform Resource Identifier (URI) of the expanded name of the node in the node-set argument that is first in document order.

\[ string \ namespace\-uri(node\-set?) \]

**Remarks**

If the node-set argument is empty, the first node has no expanded name or the namespace URI of the expanded name is Null and an empty string is returned. If the argument is omitted, it defaults to a node-set with the context node as its only member.

**Note** The string returned will be empty except for element nodes and attribute nodes.

**Sample**

**XML File (data.xml)**

```xml
<?xml-stylesheet type="text/xsl" href="sample.xsl"?>
<b:catalog xmlns:b="x-schema:book-schema.xml">
  <b:book id="bk101">
    <b:author>Gambardella, Matthew</b:author>
    <b:genre>Computer</b:genre>
    <b:price>44.95</b:price>
    <b:publish_date>2000-10-01</b:publish_date>
    <b:description>An in-depth look at creating applications with</b:description>
  </b:book>
  <b:book id="bk102">
    <b:author>Ralls, Kim</b:author>
    <b:title>Midnight Rain</b:title>
    <b:genre>Fantasy</b:genre>
    <b:price>5.95</b:price>
    <b:publish_date>2000-12-16</b:publish_date>
    <b:description>A former architect battles corporate zombies, a</b:description>
  </b:book>
</b:catalog>
```
XSLT File (sample.xsl)

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h3>namespace-uri() Function</h3>
        <table width="100%" border="1">
          <tr>
            <td width="25%"><b>namespace-uri()</b></td>
            <td width="25%"><b>name()</b></td>
            <td width="25%"><b>local-name</b></td>
            <td width="25%"><b>text()</b></td>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="*">
    <tr>
      <td><xsl:value-of select="namespace-uri()"/></td>
      <td><xsl:value-of select="name()"/></td>
      <td><xsl:value-of select="local-name"/></td>
      <td><xsl:value-of select="text()"/></td>
    </tr>
    <xsl:apply-templates select="*"/>
  </xsl:template>
</xsl:stylesheet>
```

Auxiliary XSLT File (book-schema.xml)

```xml
<Schema name="books" xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="author"/>
  <ElementType name="title"/>
  <ElementType name="genre"/>
  <ElementType name="price"/>
</Schema>
```
XMLElement name="publish_date"/>
<XMLElement name="description"/>
<AttributeType name="id" dt:type="id"/>

<XMLElement name="catalog">
  <element type="book"/>
</XMLElement>

<XMLElement name="book" model="closed" content="eltOnly">
  <attribute type="id"/>
  <element type="author"/>
  <element type="title"/>
  <element type="genre"/>
  <element type="price"/>
  <element type="publish_date"/>
  <element type="description"/>
</XMLElement>
</Schema>

Formatted Output

namespace-uri() Function

<table>
<thead>
<tr>
<th>namespace-uri()</th>
<th>name()</th>
<th>local-name</th>
<th>text()</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-schemabook-schemaxml</td>
<td>book</td>
<td>book</td>
<td></td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>catalog</td>
<td>catalog</td>
<td></td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>b:author</td>
<td>author</td>
<td>Gambardella, Matthew</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>b:title</td>
<td>title</td>
<td>XML Developer’s Guide</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>b:genre</td>
<td>genre</td>
<td>Computer</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>b:price</td>
<td>price</td>
<td>44.95</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>b:publish_date</td>
<td>publish_date</td>
<td>2000-10-01</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>b:description</td>
<td>description</td>
<td>An in-depth look at creating applications with XML.</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:book</td>
<td>book</td>
<td></td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:author</td>
<td>author</td>
<td>Ralls, Kim</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:title</td>
<td>title</td>
<td>Midnight Rain</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:genre</td>
<td>genre</td>
<td>Fantasy</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:price</td>
<td>price</td>
<td>5.95</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:publish_date</td>
<td>publish_date</td>
<td>2000-12-16</td>
</tr>
<tr>
<td>x-schemabook-schemaxml</td>
<td>x:description</td>
<td>description</td>
<td>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</td>
</tr>
</tbody>
</table>
Processor Output

```html
<html>
<body>
<h3>namespace-uri() Function</h3>
<table width="100%" border="1">
<tr>
<td width="25%"><b>namespace-uri()</b></td>
<td width="25%"><b>name()</b></td>
<td width="25%"><b>local-name</b></td>
<td width="25%"><b>text()</b></td>
</tr>
<tr>
<td>x-schema:book-schema.xml</td>
<td>b:catalog</td>
<td>catalog</td>
<td></td>
</tr>

... 

<tr>
<td>x-schema:book-schema.xml</td>
<td>b:author</td>
<td>author</td>
<td>Gambardella, Matthew</td>
</tr>

...</table>
</body>
</html>

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
position Function

Returns the position, or index number, of the node, relative to all the selected nodes in the node list.

\[ \text{number} \text{ position}() \]

Remarks

The position of the node is 1-based, so the first node returns a position of 1.

Example

The following code example illustrates the effects of the position() function.

XML File (position.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="position.xsl"?>
<test>
  <x a="a11">
    <x a="a21">
      <x a="a31">
        <y b="b31">y31</y>
        <y b="b32">y32</y>
      </x>
    </x>
  </x>
  <x a="a12">
    <x a="a22">
      <y b="b21">y21</y>
      <y b="b22">y22</y>
    </x>
  </x>
  <x a="a13">
    <y b="b11">y11</y>
    <y b="b12">y12</y>
  </x>
  <x a="a14">
    <y b="b01">y01</y>
    <y b="b02">y02</y>
  </x>
</test>
```
XSLT File (position.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml"
        omit-xml-declaration="yes"/>
    <xsl:template match="/">
        <xsl:apply-templates select="//x"/>
    </xsl:template>
    <xsl:template match="*">
        <xsl:element name="{name()}">
            <xsl:apply-templates select="@*"/>
            <xsl:value-of select="position()"/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="@*">
        <xsl:attribute name="{name()}">
            <xsl:value-of select="."/>
        </xsl:attribute>
    </xsl:template>
</xsl:stylesheet>

Output

The above XSLT style sheet, when applied to the source XML file, maps all the <x> elements to new <x> elements whose content holds their position in the document order.

<x a="a11">1</x>
<x a="a21">2</x>
<x a="a31">3</x>
<x a="a12">4</x>
<x a="a22">5</x>
<x a="a13">6</x>
<x a="a14">7</x>

To illustrate the sensitivity of the position() function to the context from which
it is operated, let's replace the following template rule (from the XSLT file above):

```xml
<xsl:template match="/">
    <xsl:apply-templates select="//x"/>
</xsl:template>
```

with this one:

```xml
<xsl:template match="/">
    <xsl:apply-templates select="//x[1]"/>
</xsl:template>
```

The result is as follows:

```xml
<x a="a11">1</x>
<x a="a21">2</x>
<x a="a31">3</x>
<x a="a22">4</x>
```

On the other hand, if we replace the template rule with the following one:

```xml
<xsl:template match="/">
    <xsl:apply-templates select="//x[2]"/>
</xsl:template>
```

we get the following result:

```xml
<x a="a12">1</x>
```

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
String Functions

String functions are used to evaluate, format, and manipulate string arguments, or to convert an object to a string.

<table>
<thead>
<tr>
<th>String Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>concat</code></td>
<td>Returns the concatenation of the arguments.</td>
</tr>
<tr>
<td><code>contains</code></td>
<td>Returns true if the first argument string contains the second argument string; otherwise returns false.</td>
</tr>
<tr>
<td><code>normalize-space</code></td>
<td>Returns the argument string with the white space stripped.</td>
</tr>
<tr>
<td><code>starts-with</code></td>
<td>Returns true if the first argument string starts with the second argument string; otherwise returns false.</td>
</tr>
<tr>
<td><code>string</code></td>
<td>Converts an object to a string.</td>
</tr>
<tr>
<td><code>string-length</code></td>
<td>Returns the number of characters in the string.</td>
</tr>
<tr>
<td><code>substring</code></td>
<td>Returns the substring of the first argument starting at the position specified in the second argument and the length specified in the third argument.</td>
</tr>
<tr>
<td><code>substring-after</code></td>
<td>Returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string.</td>
</tr>
<tr>
<td><code>substring-before</code></td>
<td>Returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string.</td>
</tr>
<tr>
<td><code>translate</code></td>
<td>Returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
**concat Function**

Returns the concatenation of the arguments.

`string concat(string, string, string*)`

**Remarks**

If an argument is not of type `string`, it is first converted to a string and then evaluated.

**Example**

The following function call returns "abcdefg".

```plaintext
concat("abc", "d", "ef", "g")
```

The following function call returns "abc3efg".

```plaintext
concat("abc", 3, "ef", "g")
```

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
contains Function

Checks whether the first argument string contains the second argument string.

boolean contains(str1, str2)

Parameters

str1
A string that might contain the second argument.

str2
A string that might be contained in the first argument.

Return Values

Returns true if the first argument string contains the second argument string.
Returns false otherwise.

Remarks

If an argument is not of type string, it is first converted to a string and then evaluated. The contains() function is case-sensitive.

Example

The following example illustrates how to use the contains() function to query a collection of books whose titles contain the word "Pattern".

XML FILE (contains.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="contains.xsl"?>
<bookstore>
   <book>
      <title>The Weather Pattern</title>
      <author>Weather Man</author>
   </book>
</bookstore>
```
<price>100.00</price>
</book>
<book>
  <title>Weaving Patterns</title>
  <author>Weaver</author>
  <price>150.00</price>
</book>
<book>
  <title>Speech Pattern</title>
  <author>Speaker</author>
  <price>15.00</price>
</book>
<book>
  <title>Writing Style</title>
  <author>Writer</author>
  <price>1500.00</price>
</book>
</bookstore>

XSLT FILE (contains.xsl)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"
    omit-xml-declaration="yes"/>

  <xsl:template match="/">
    <html>
      <head><title>example</title></head>
      <body>
        <xsl:apply-templates select="//book"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="book">
    <xsl:if test="contains(title, 'Pattern')">
      <DIV>
        <B><xsl:value-of select="title"/></B> by
        <I><xsl:value-of select="author"/></I> costs
        <xsl:value-of select="price"/>.</DIV>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
Output

When applied to the XML file (`contains.xml`), the XSLT style sheet above produces the following output:

**The Weather Pattern** by *Weather Man* costs 100.00.

**Weaving Patterns** by *Weaver* costs 150.00.

**Speech Pattern** by *Speaker* costs 15.00.

See Also

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
normalize-space Function

Returns the argument string with the leading, trailing, and repeating white spaces stripped.

string normalize-space(string)

Remarks

White space is normalized by stripping leading and trailing white space and replacing sequences of white space characters with a single space. If the argument is omitted, the string-value of the context node is normalized and returned.

The following function call returns "abc def":

normalize-space(" abc def ")

If an argument is not of the type string, it is first converted to a string and then evaluated. See the example below.

Example

The following example normalizes a block of text string with unnormalized white spaces (tabs, leading and trailing spaces, and multiple spaces between words. The text string is the value of an <text> element.

XML File (normSpace.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="normalizeSpace.xsl"?>
<text>
  This is a test, with a lot of irregular spacing and waiting to be normalized.
</text>
```
XSLT File (normSpace.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes"/>

  <xsl:template match="/text">
    Unnormalized: "<xsl:value-of select='.'/>"
    Normalized: "<xsl:value-of select='normalize-space()'/>'
  </xsl:template>
</xsl:stylesheet>
```

This XSLT produces the following output:

Unnormalized:
"This is a test, with a lot of irregular spacing and waiting to be normalized."

Normalized:
"This is a test, with a lot of irregular spacing and waiting to be normalized."

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
starts-with Function

Returns true if the first argument string starts with the second argument string; otherwise returns false.

boolean starts-with(string, string)

Remarks

If an argument is not of type string, it is first converted to a string and then evaluated. The function is case-sensitive.

Example

The following example illustrates how to use the starts-with() function to query a collection of books whose title begins with a capital letter "W".

XML FILE (starts-with.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="contains.xsl"?>
<bookstore>
  <book>
    <title>The Weather Pattern</title>
    <author>Weather Man</author>
    <price>100.00</price>
  </book>
  <book>
    <title>Weaving Patterns</title>
    <author>Weaver</author>
    <price>150.00</price>
  </book>
  <book>
    <title>Speech Pattern</title>
    <author>Speaker</author>
    <price>15.00</price>
  </book>
  <book>
    <title>Writing Style</title>
    <author>Writer</author>
  </book>
</bookstore>
```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"
    omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <html>
      <head><title>example</title></head>
      <body>
        <xsl:apply-templates select="//book"/>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

Output

This XSLT style sheet, when applied to the XML file (starts-with.xml) produces the following output:

Weaving Patterns by Weaver costs 150.00.

Writing Style by Writer costs 1500.00.

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types
Reference
string Function

Converts an object to a string.

\texttt{string \ string(object?)}

Remarks

An object is converted to a string as follows.

A node-set is converted to a string by returning the string value of the node in the node-set that is first in document order. If the node-set is empty, an empty string is returned.

A number is converted to a string as follows.

- NaN is converted to the string NaN.
- positive zero is converted to the string "0".
- negative zero is converted to the string "0".
- positive infinity is converted to the string "Infinity".
- negative infinity is converted to the string "-Infinity".
- If the number is an integer, the number is represented in decimal form as a number with no decimal point and no leading zeros, preceded by a minus sign (-) if the number is negative.
- Otherwise, the number is represented in decimal form as a number with a decimal point and at least one digit before the decimal point and at least one digit after the decimal point, preceded by a minus sign (-) if the number is negative; there must be no leading zeros before the decimal point, apart possibly from the one required digit immediately before the decimal point; beyond the one required digit after the decimal point there must be as many, but only as many, more digits as are needed to uniquely distinguish the number from all other IEEE 754 numeric values.

The Boolean false value is converted to the string "false". The Boolean true value is converted to the string "true".

An object of a type other than the four basic types is converted to a string in a
way that is dependent on that type.

If the argument is omitted, it defaults to a node-set with the context node as its only member.

**Note** The string() function is not intended for converting numbers into strings for presentation to users. The format-number() function and <xsl:number> element in XSL Transformations (XSLT) provide this functionality.

**Example**

The following example illustrates how to use the string() function in an XPath expression. In two cases (see the instructions in bold in the XSLT file below), the function is used to ensure that its argument is treated as a string expression.

**XML File (string.xml)**

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="string.xsl"?>
<arithmetics>
    <operation>
        <operator>+</operator>
        <operand>1</operand>
        <operand>2.00</operand>
    </operation>
    <operation>
        <operator>+</operator>
        <operand>One</operand>
        <operand>2.00</operand>
    </operation>
    <operation>
        <operator>-</operator>
        <operand>1</operand>
        <operand>2.00</operand>
    </operation>
    <operation>
        <operator>*</operator>
        <operand>1</operand>
        <operand>2.00</operand>
    </operation>
    <operation>
        <operator>div</operator>
        <operand>-1</operand>
    </operation>
</arithmetics>
```
Output

1 + 2.00 = 3
One + 2.00 = NaN
1 - 2.00 = -1
1 * 2.00 = 2
-1 div 0.0 = -Infinity
5 mod 2 = 1
5 mod 2.5 = 0
5 mod 2.25 = 0.5
0 & 1 = Invalid arithmetic operation

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference | NaN Values | format-number Function
string-length Function

Returns the number of characters in the string.

\textit{number} \textit{string-length(}\textit{string} \textit{)}

\textbf{Remarks}

If the argument is omitted, it returns the number of characters in the string-value of the context node.

The following function call returns 4:

\texttt{string-length("abcd")}

The following function call returns 5:

\texttt{string-length("ab cd")}

If an argument is not of type \textit{string}, it is first converted to a string and then evaluated.

\textbf{See Also}

\texttt{Data Types in Schemas} | \texttt{XDR Schema Data Types Reference} | \texttt{XML Data Types Reference}
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
substring Function

Returns the substring of the first argument starting at the position specified in the second argument and the length specified in the third argument.

\[ \text{string } \text{substring}(\text{string}, \text{number}, \text{number})? \]

Remarks

Each character in the string is considered to have a numeric position: the position of the first character is 1, the position of the second character is 2, and so on.

If the third argument is not specified, it returns the substring starting at the position specified in the second argument and continuing to the end of the string.

The following function call returns "234":

\[ \text{substring}("12345", 2, 3) \]

The following function call returns "2345":

\[ \text{substring}("12345", 2) \]

The returned substring contains those characters for which the position of the character is greater than or equal to the rounded value of the second argument, and if the third argument is specified, less than the sum of the rounded value of the second argument and the rounded value of the third argument. The comparisons and addition used for the preceding follow the standard IEEE 754 rules; rounding is done as if by a call to the \text{round()} function.

The following examples illustrate unusual cases.

\[ \text{substring}("12345", 1.5, 2.6) \text{ returns } "234" \]

\[ \text{substring}("12345", 0, 3) \text{ returns } "12" \]

\[ \text{substring}("12345", 0 \text{ div } 0, 3) \text{ returns } "" \]
substring("12345", 1, 0 div 0) returns ""

substring("12345", -42, 1 div 0) returns "12345"

substring("12345", -1 div 0, 1 div 0) returns ""

**Example**

This example demonstrates the preceding substring() expressions.

**XML File**

None; the XSLT file calls itself.

**XSLT File (substring.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    
    <xsl:template match="/"
    
    substring("12345", 2, 3) =
    
    <xsl:value-of select='substring("12345", 2, 3)'/><br/>
    
    substring("12345", 2) =
    
    <xsl:value-of select='substring("12345", 2)'/><br/>
    
    substring("12345", 1.5, 2.6) =
    
    <xsl:value-of select='substring("12345", 1.5, 2.6)'/><br/>
    
    substring("12345", 0, 3) =
    
    <xsl:value-of select='substring("12345", 0, 3)'/><br/>
    
    substring("12345", 0 div 0, 3) =
    
    <xsl:value-of select='substring("12345", 0 div 0, 3)'/><br/>
    
    substring("12345", 1, 0 div 0) =
    
    <xsl:value-of select='substring("12345", 1, 0 div 0)'/><br/>
    
    substring("12345", -42, 1 div 0) =
    
    <xsl:value-of select='substring("12345", -42, 1 div 0)'/><br/>
    
    substring("12345", -1 div 0, 1 div 0) =
    
    <xsl:value-of select='substring("12345", -1 div 0, 1 div 0)'/>

</xsl:template>
</xsl:stylesheet>
```

**Try It**

1. Follow the instructions in [Initiate XSLT in a Script](#)
2. Use the XSLT style sheet above (substring.xsl) as the input for both the
XML and XSLT files.

3. Run the script. You should get the following output.

Formatted Output

substring("12345",2,3) = 234
substring("12345",2) = 2345
substring("12345", 1.5, 2.6) = 234
substring("12345", 0, 3) = 12
substring("12345", 0 div 0, 3) =
substring("12345", 1, 0 div 0) =
substring("12345", -42, 1 div 0) = 12345
substring("12345", -1 div 0, 1 div 0) =

Processor Output

<?xml version="1.0" ?>
    substring("12345",2,3) =
        234<br />
    substring("12345",2) =
        2345<br />
    substring("12345", 1.5, 2.6) =
        234<br />
    substring("12345", 0, 3) =
        12<br />
    substring("12345", 0 div 0, 3) =
        <br />
    substring("12345", 1, 0 div 0) =
        <br />
    substring("12345", -42, 1 div 0) =
        12345<br />
    substring("12345", -1 div 0, 1 div 0) =

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
substring-after Function

Returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string.

\[ string \ substring-after(string, \ string) \]

**Remarks**

If an argument is not of type `string`, it is first converted to a string and then evaluated.

The following function call returns "04/01":

\[ substring-after("1999/04/01","/") \]

The following function call returns "99/04/01":

\[ substring-after("1999/04/01","19") \]

The following function call returns an empty string ":"

\[ substring-after("1999/04/01","2") \]

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
substring-before Function

Returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string.

$string$ substring-before($string$, $string$)

Remarks

If an argument is not of type $string$, it is first converted to a string and then evaluated.

The following function call returns "1999":

substring-before("1999/04/01","/")

The following function call returns an empty string "":

substring-before("1999/04/01","2")

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
**translate Function**

Returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string.

\[ \text{string translate(string, string, string)} \]

**Remarks**

This function provides a mapping of characters on the first argument. The second and third arguments describes the scheme of the mapping.

The following function call returns "BAr":

\[ \text{translate("bar", "abc", "ABC")} \]

If there is a character in the second argument string with no character at a corresponding position in the third argument string (because the second argument string is longer than the third argument string), occurrences of that character in the first argument string are removed.

The following function call returns "AAA":

\[ \text{translate("--aaa--", "abc-", "ABC")} \]

If a character occurs more than once in the second argument string, the first occurrence determines the replacement character. If the third argument string is longer than the second argument string, excess characters are ignored.

If an argument is not of type \text{string}, it is first converted to a string and then evaluated.

**Example**

This example demonstrates the preceding translate() expressions.
XML File

None; the XSLT file calls itself.

XSLT File (translate.xsl)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="translate.xsl"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/"
        translate("bar","abc","ABC") =
        <xsl:value-of select='translate("bar","abc","ABC")'/>
    </xsl:template>
    <br/>
    translate("--aaa--","abc-","ABC") =
    <xsl:value-of select='translate("--aaa--","abc-","ABC")'/>
</xsl:stylesheet>
```

Formatted Output

```
translate("bar","abc","ABC") = BAr
translate("--aaa--","abc-","ABC") = AAA
```

Processor Output

```
<?xml version="1.0" encoding="UTF-16"?>
translate("bar","abc","ABC") = BAr<br>translate("--aaa--","abc-","ABC") = AAA
```

See Also

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
Boolean Functions

The XML Path Language (XPath) syntax supports Boolean functions that return strings or numbers, and can be used with comparison operators in filter patterns.

**Boolean Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Converts the argument to a Boolean.</td>
</tr>
<tr>
<td>false</td>
<td>Returns false.</td>
</tr>
<tr>
<td>lang</td>
<td>Returns true if the <code>xml:lang</code> attribute of the context node is the same as the argument string.</td>
</tr>
<tr>
<td>not</td>
<td>Returns true if the argument is false, otherwise, false.</td>
</tr>
<tr>
<td>true</td>
<td>Returns true.</td>
</tr>
</tbody>
</table>
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
boolean Function

Converts the argument to a Boolean.

\textit{boolean} boolean\textit{(arg)}

Remarks

This function converts arguments to Booleans according to the following rules.

- If the argument is a negative or positive number, it is converted to the Boolean value \textit{true}. If the argument is zero or an \textit{NaN value}, it is converted to \textit{false}.
- If the argument is a non-empty node-set, it is converted to \textit{true}. An empty node-set is converted to \textit{false}.
- If the argument is a non-empty string, it is converted to \textit{true}. An empty string is converted to \textit{false}.
- If the argument is an object of a type other than the four basic types, it is converted to a Boolean in a way that is dependent on that type. For more information about the basic types of objects, see \textbf{Using XPath Expressions to Select Nodes}.

Example

XML File

None; the XSLT file calls itself.

If you use the sample XML file, books.xml, and change the \textit{href} attribute to reference \textbf{bool.xsl}, \textit{boolean}//book resolves as \textit{true}.

XSLT File (bool.xsl)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="bool.xsl"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl=http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>
```
boolean() Function

- boolean(0) = false
- boolean(1) = true
- boolean(-100) = true
- boolean(100) = true
- boolean(NaN) = false
- boolean('hello') = true
- boolean('') = false
- boolean(//book) = true
- boolean(//notfound) = false
boolean(100) = true
boolean(NaN) = false
boolean('hello') = true
boolean('') = false
boolean(/book) = false
boolean(/notfound) = false

Processor Output

<html>
<body>
<h3>boolean() Function</h3>
<ul>
<li><b>boolean(0)</b> = false</li>
<li><b>boolean(1)</b> = true</li>
<li><b>boolean(-100)</b> = true</li>
<li><b>boolean(100)</b> = true</li>
<li><b>boolean(NaN)</b> = false</li>
<li><b>boolean('hello')</b> = true</li>
<li><b>boolean('') </b> = false</li>
<li><b>boolean(/book)</b> = true</li>
<li><b>boolean(/notfound)</b> = false</li>
</ul>
</body>
</html>

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference | NaN Values | Sample XML File (books.xml)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
false Function

Returns false.

boolean false()

Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xsl:value-of select=&quot;false() and false()&quot;/&gt;</td>
<td>false</td>
</tr>
<tr>
<td>&lt;xsl:value-of select=&quot;true() and false()&quot;/&gt;</td>
<td>false</td>
</tr>
<tr>
<td>&lt;xsl:value-of select=&quot;false() or false()&quot;/&gt;</td>
<td>true</td>
</tr>
<tr>
<td>&lt;xsl:value-of select=&quot;true()&quot;/&gt;</td>
<td>true</td>
</tr>
</tbody>
</table>

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
**lang Function**

Returns true if the `xml:lang` attribute of the context node is the same as a sublanguage of the language specified by the argument string.

**Syntax**

```java
boolean lang(string)
```

**Remarks**

The language of the context node is determined by the value of the `xml:lang` attribute on the context node, or if the context node has no `xml:lang` attribute, by the value of the `xml:lang` attribute on the nearest ancestor of the context node that has an `xml:lang` attribute. If there is no such attribute, `lang()` returns false. If there is such an attribute, `lang()` returns true if the attribute value is equal to the argument (ignoring case) or if there is a suffix starting with "-", such that the attribute value is equal to the argument, ignoring the suffix of the attribute value and ignoring case.

Given the following XML elements:

```xml
<para xml:lang="en"/>
<div xml:lang="en"><para/></div>
<para xml:lang="EN"/>
<para xml:lang="en-us"/>
```

The following function call returns true for all:

```
lang("en")
```

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
not Function

Returns true if the argument is false; otherwise false.

*boolean not(boolean)*

**Example**

In this example, the *notexample.xsl* stylesheet is applied to the *books.xml* document. For each book node, the value of the `<price>` element is evaluated. If it's not greater than 10, the value of the `<title>` element and the value of the `<price>` element are output to the browser.

**XML File (books.xml)**

```xml
<?xml version='1.0'?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the wor</description>
  </book>
</catalog>
```

**XSLT File (notexample.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
```
Formatted Output

Midnight Rain - $5.95

Processor Output

<?xml version="1.0" encoding="UTF-16"?>Midnight Rain - $5.95<br />

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
true Function

Returns true.

boolean true()

Example

See the example for the false() function.

See Also

Data Types in Schemas  |  XDR Schema Data Types Reference  |  XML Data Types Reference
Number Functions

The XML Path Language (XPath) syntax supports Number functions that return strings or numbers and can be used with comparison operators in filter patterns.

Number Functions

ceiling  Returns the smallest integer that is not less than the argument.
floor    Returns the largest integer that is not greater than the argument.
number   Converts the argument to a number.
round    Returns an integer closest in value to the argument.
sum      Returns the sum of all nodes in the node-set. Each node is first converted to a number value before summing.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
ceiling Function

Returns the smallest integer that is not less than the argument.

\[ \text{number ceiling(number)} \]

Example

This example demonstrates ceiling() expressions.

XML File

None; the XSLT file calls itself.

XSLT File (ceiling.xsl)

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="ceiling.xsl"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <p>ceiling(2.5) = \<xsl:value-of select='ceiling(2.5)'/></p>
    <p>ceiling(-2.3) = \<xsl:value-of select='ceiling(-2.3)'/></p>
    <p>ceiling(4) = \<xsl:value-of select='ceiling(4)'/></p>
  </xsl:template>
</xsl:stylesheet>
```

Formatted Output

ceiling(2.5) = 3
ceiling(-2.3) = -2
ceiling(4) = 4

Processor Output
ceiling(2.5) = 3
ceiling(-2.3) = -2
ceiling(4) = 4

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
floor Function

Returns the largest integer that is not greater than the argument.

\[ \text{number} \ \text{floor}(\text{number}) \]

Example

This example demonstrates \text{floor()} expressions.

XML File

None; the XSLT file calls itself.

XSLT File (floor.xsl)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="floor.xsl"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <p/>floor(3.5) =
        <xsl:value-of select='floor(3.5)'/>
        <p/>floor(-1.3) =
        <xsl:value-of select='floor(-1.3)'/>
        <p/>floor(4) =
        <xsl:value-of select='floor(4)'/>
    </xsl:template>
</xsl:stylesheet>
```

Formatted Output

floor(3.5) = 3
floor(-1.3) = -2
floor(4) = 4

Processor Output
floor(3.5) = 3
floor(-1.3) = -2
floor(4) = 4

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
**number Function**

Converts the argument to a number.

\[ \text{number number(object?)} \]

**Remarks**

Converts its argument to a number as follows.

- A string that consists of optional white space followed by an optional minus sign followed by a number followed by white space is converted to the IEEE 754 number that is nearest (according to the IEEE 754 round-to-nearest rule) to the mathematical value represented by the string; any other string is converted to NaN.
- Boolean true is converted to 1; Boolean false is converted to 0.
- A node-set is first converted to a string and then converted in the same way as a string argument.
- An object of a type other than the four basic types (node-set, Boolean, number, or string) is converted to a number in a way that is dependent on that type.

If the argument is omitted, it defaults to a node-set with the context node as its only member.

**Note** The `number()` function should not be used for conversion of numeric data occurring in an element in an XML document unless the element is of a type that represents numeric data in a language-neutral format (that would typically be transformed into a language-specific format for presentation to a user). In addition, the `number()` function cannot be used unless the language-neutral format used by the element is consistent with the XML Path Language (XPath) syntax for a number.

**Example**

**XML File**
None; the XSLT file calls itself.

**XSLT File (number.xsl)**

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="number.xsl"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <body>
                <h3>number() Function</h3>
                <ul>
                    <li><b>number('2048')</b> = <xsl:value-of select="number('2048')"/></li>
                    <li><b>number('-2048')</b> = <xsl:value-of select="number('-2048')"/></li>
                    <li><b>number('text')</b> = <xsl:value-of select="number('text')"/></li>
                    <li><b>number('109.54' div '1')</b> = <xsl:value-of select="number('109.54' div '1')"/>
                </ul>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

**Formatted Output**

**number() Function**

- `number('2048')` = 2048
- `number('-2048')` = -2048
• `number('text') = NaN`
• `number('109.54') = 109.54`

**Processor Output**

```html
<html><body><h3>number() Function</h3><ul><li><b>number('2048')</b> = 2048</li><li><b>number('-2048')</b> = -2048</li><li><b>number('text')</b> = NaN</li><li><b>number('109.54')</b> = 109.54</li></ul></body></html>
```

**See Also**

[Data Types in Schemas](#) | [XDR Schema Data Types Reference](#) | [XML Data Types Reference](#)
round Function

Returns an integer closest in value to the argument.

\[ \text{number round(number)} \]

Remarks

If there are two such numbers, the one that is closest to positive infinity is returned.

If the argument is NaN, NaN is returned.

If the argument is positive infinity, positive infinity is returned.

If the argument is negative infinity, negative infinity is returned.

If the argument is positive zero, positive zero is returned.

If the argument is negative zero, negative zero is returned.

If the argument is less than zero but greater than or equal to -0.5, negative zero is returned.

For the last two cases, the result of calling the `round()` function is not the same as the result of adding 0.5 and then calling the `floor()` function because positive zero will be returned in such cases.

Example

This example demonstrates `round()` expressions.

XML File

None; the XSLT file calls itself.

XSLT File (round.xsl)
<round(2.6) = 3
round (2.4) = 2
round(2.5) = 3
round(-1.6) = -2
round(-1.5) = -1

See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
sum Function

Returns the sum of all nodes in the node-set. Each node is first converted to a number value before summing.

\[
\text{number} \ \text{sum}(\text{node-set})
\]

Example

This example demonstrates the XPath expression \( \text{sum}(/\text{a}) \). It returns 6, which is the sum of the numbers in \(<\text{a}>\) elements.

XML File (as.xml)

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="sum.xsl"?>
<root>
  <a>1</a>
  <a>3</a>
  <a>2</a>
</root>
```

XSLT File (sum.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:value-of select='sum(//a)'/>
  </xsl:template>
</xsl:stylesheet>
```

Formatted Output

6

Processor Output

```
<?xml version="1.0" encoding="UTF-16"?>6
```
See Also

Data Types in Schemas | XDR Schema Data Types Reference | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Microsoft XPath Extension Functions

MSXML provides a number of extension functions to offer additional features beyond those specified in the XPath Version 1.0 specification. Some of these extension functions enable manipulations of nodes based on their XSD data types. Others provide some popular utilities, such as lexicographical comparison of strings, formatting times and dates, converting date/time to Coordinated Universal Time units, etc.

The names of extended functions must be of qualified name consisting of a namespace URI (or its proxy), a colon, and a local part. Microsoft XPath extension functions typically sport a ms prefix that has been associated with the namespace URI ("urn:schemas-microsoft-com:xslt") for the Microsoft extension functions. For more information, see Defining Namespace Prefixes for Extension Functions.

**XPath Extension Functions for XSD Support**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ms:type-is</code></td>
<td>Compares the current node's data type against the specified node type.</td>
</tr>
<tr>
<td></td>
<td>Returns the nonqualified name of the XSD type of the current node or the</td>
</tr>
<tr>
<td></td>
<td>first node (in document order) in the provided node-set.</td>
</tr>
<tr>
<td><code>ms:type-local-name ([node-set])</code></td>
<td>Returns the namespace URI associated with the XSD type of a current node or</td>
</tr>
<tr>
<td></td>
<td>the first node (in document order) in the provided node-set.</td>
</tr>
<tr>
<td><code>ms:type-namespace-uri ([node-set])</code></td>
<td>Returns true if XSD information is available for a current node.</td>
</tr>
</tbody>
</table>

**XPath Extension Functions of Miscellaneous Utilities**

<table>
<thead>
<tr>
<th>ms:string-</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>compare</td>
<td>Performs lexicographical string comparison.</td>
</tr>
<tr>
<td>ms:utc</td>
<td>Converts the prefixed date/time related values into Coordinated Universal Time and into a fixed (normalized) representation that can be sorted and compared lexicographically.</td>
</tr>
<tr>
<td>ms:namespace-uri</td>
<td>Resolves the prefix part of a qualified name into a namespace URI.</td>
</tr>
<tr>
<td>ms:local-name</td>
<td>Returns the local name part of a qualified name by stripping out the namespace prefix.</td>
</tr>
<tr>
<td>ms:number</td>
<td>Takes a string argument in XSD format and converts it into an XPath number.</td>
</tr>
<tr>
<td>ms:format-date</td>
<td>Converts standard XSD date formats to characters suitable for output.</td>
</tr>
<tr>
<td>ms:format-time</td>
<td>Converts standard XSD time formats to characters suitable for output.</td>
</tr>
</tbody>
</table>
Defining Namespace Prefixes for Extension Functions

To use the Microsoft XPath Extension functions, you must declare the appropriate namespaces in an XSLT style sheet, as follows:

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ms="urn:schemas-microsoft-com:xslt"
    version="1.0">

It is important that the namespace URI, urn:schemas-microsoft-com:xslt, must appear exactly as shown above. It is reasonable to choose ms as the namespace prefix, although you can use a name of your choosing for the prefix. For example, the following code snippet is equally valid:

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ext="urn:schemas-microsoft-com:xslt"
    version="1.0">

The namespace prefix can also be defined within the XSLT element, as follows.

```xml
<xsl:for-each select="//*[ms:type-local-name()='ID']" xmlns:ms='urn:schemas-microsoft-com:xslt'>

When using the DOM, the namespace prefix is declared using the `setProperty` method of IXMLDOMDocument2.

```javascript
docObj.setProperty("SelectionNamespaces", "xmlns:ms='urn:schemas-microsoft-com:xslt'");
```

See Also

[setProperty method](#) | [Using Namespaces in Documents](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Using XPath Extension Functions for XSD Support

The topics in this section provide an example of how to use XPath extension functions for XSD support. The example consists of the following pieces.

books.xml
The file that is loaded and validated against the books.xsd schema.

books.xsd
The XSD schema that is used to validate the books.xml file.

Microsoft Visual Basic example code
This Visual Basic example code:
- Creates an XMLSchemaCache object.
- Adds the schema to the XMLSchemaCache object.
- Creates a DOMDocument object and sets the schemas property to reference the books.xsd schema. The example code also sets the SelectionLanguage and SelectionNamespaces properties on the DOMDocument object.
- Uses the XSD type-local-name function to select a node based on it's data type.
- Uses the XSD type-is function to return a node list that contains nodes only of the data type string.

To get started

1. Open Visual Basic 6.0, and in the New Project dialog box, double-click Standard EXE.
2. On the Project menu, click References.
3. In the Available References list, select Microsoft XML,v5.0, and then click OK.
4. Add a Command button to Form1.
5. Save the project to a folder on your hard drive.

This example uses the following topics.

- The XML Document (books.xml)
• The XSD Schema (books.xsd)
• Visual Basic XSD Extension Function Code

See Also

XML Schema Examples | XML Schema Element Map | XML Schema Elements | XML Data Types Reference | Primitive XML Data Types | Derived XML Data Types | Data Type Facets | IXMLDOMSchemaCollection/XMLSchemaCache

Other Resources

Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
The XML Document (books.xml)

The books.xml file used in this example, shown below, is a modified version of the books.xml file used throughout the MSXML SDK. Notice that urn:books is declared as the default namespace for this document.

If you want to run the example in this topic, copy the following code to a text editor, such as Notepad, and then save the file as books.xml.

```xml
<?xml version="1.0"?>
<x:catalog xmlns:x="urn:books">
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
  </book>
</x:catalog>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
The XSD Schema (books.xsd)

The following document shows the books.xsd schema used in this example.

If you want to run the example in this topic, copy the following code to a text editor, such as Notepad, and then save the file as books.xsd.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book"
          minOccurs="0"
          maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="author" type="xsd:string"/>
              <xsd:element name="title" type="xsd:string"/>
              <xsd:element name="genre" type="xsd:string"/>
              <xsd:element name="price" type="xsd:float"/>
              <xsd:element name="publish_date" type="xsd:date"/>
              <xsd:element name="description" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
Visual Basic XSD Extension Function Code

The following code snippet shows how you can use XSD extension functions to return nodes based on their XSD data type. This example code uses the type-is function to return the first string data type in the document and the type-local-name function to return a list of nodes of the string data type.

The Visual Basic code creates an XMLSchemaCache50 object, adds the namespace URI declaration (urn:books) and the schema (books.xsd) to the object, and then references the schema using the schemas property of the DOMDocument object. Notice in the code that the SelectionLanguage property is set to XPath and the SelectionNamespaces property is set to "xmlns:ms='urn:schemas-microsoft-com:xslt" enabling the ms: namespace prefix to be used. The Validation is performed when the books.xml file is loaded into the DOMDocument object. Validation errors are returned using the parseError property of the DOMDocument object.

To run the example

1. Copy the code shown below to the Command1_Click procedure. It should look like the code in the code snippet below when you are done.
2. On the Visual Basic toolbar, click Start, and then click the Command1 button on form1.

When you run the example, the Set objNodeList = xmldom.selectNodes "/*. [ms:type-is('http://www.w3.org/2001/XMLSchema','string')]") statement returns a list of nodes of data type string. The For/Next loop iterates through the nodes and displays the text of each node in a message box.

Private Sub Command1_Click()
Dim xmlschema As Msxml2.XMLSchemaCache50
Set xmlschema = New Msxml2.XMLSchemaCache50
Dim xmldom As Msxml2.DOMDocument50
Set xmldom = New Msxml2.DOMDocument50
Dim objElem As IXMLDOMNode
Dim objNodeList As IXMLDOMNodeList
}
xmlschema.Add "urn:books", App.Path & "\books.xsd"
Set xmldom.schemas = xmlschema

xmldom.setProperty "SelectionLanguage", "XPath"
xmldom.setProperty "SelectionNamespaces", "xmlns:ms='urn:schemas-mic

xmldom.async = False
xmldom.Load App.Path & "\books.xml"

'returns a list of nodes with the string data type
Set objNodeList = xmldom.selectNodes _
    ("//*[ms:type-is('http://www.w3.org/2001/XMLSchema','string')]"
For i = 0 To (objNodeList.length - 1)
    MsgBox objNodeList.Item(i).Text
Next

If xmldom.parseError.errorCode <> 0 Then
    MsgBox xmldom.parseError.errorCode & " " & xmldom.parseError.rea
Else
    MsgBox "No Error"
End If

End Sub
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
ms:type-is Function

Tests whether the current node's data type belongs to the given namespace. The data type and the namespace are specified in the arguments.

boolean ms:type-is(string URI, string local-name)

Parameters

string URI
The namespace URI for the data type against which the current data type is evaluated.

string local-name
The local name of the data type against which the current data type is evaluated.

Remarks

The function returns true if the current node is of the specified data type belonging to the given namespace. Otherwise it returns false. If local-name refers to a data type undeclared in the given namespace (URI), the function returns false. Nameless data types defined in the given namespace cause the function to return false as well.

This function is aware of XSD inheritance, so that if type "b" is derived from "a", then for the node of type "b", type-is("http://www.example.microsoft.com/catalog", "a") returns true.

Example

The following example uses an XSLT template rule to select all the elements in books.xml whose data type is date, as defined in books.xsd. The example also shows how to do the query using an XML DOM instance.

XML File (books.xml)
Use books.xml.

XSD File (books.xsd)

Use books.xsd.

XSLT File (books.xslt)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:ms="urn:schemas-microsoft-com:xslt"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="text()"/>
    <xsl:output method="html"
        omit-xml-declaration="yes"/>
    <xsl:template match="/">
        <H3>nodes of date data types:</H3>
        <xsl:apply-templates select="//*[ms:type-is('http://www.w3.org/2001/XMLSchema','date')]"/>
    </xsl:template>
    <xsl:template match="*">
        <DIV>
            <xsl:value-of select="name()"/>
            =
            <xsl:value-of select="."/>
        </DIV>
        <xsl:apply-templates/>
    </xsl:template>
</xsl:stylesheet>
```

HTML File (books.html)

The HTML file contains a JScript that handles loading XML, XSLT, and XSD files.

```html
<html>
    <head>
        <script>
            function init() {
                try {
                    var objxsd = new ActiveXObject("Msxml2.XMLSchemaCache.5.0")
                    var objxml = new ActiveXObject("Msxml2.DOMDocument.5.0");
                    var objxsl = new ActiveXObject("Msxml2.DOMDocument.5.0");
                }
```
// namespace uri ("urn:books") must be declared as one of the namespace declarations in the "books.xml" that is an instance of "books.xsd"
objxsd.add("urn:books", "books.xsd");

objxml.schemas = objxsd;
objxml.setProperty("SelectionLanguage", "XPath");
objxml.setProperty("SelectionNamespaces",
    "xmlns:ms='urn:schemas-microsoft-com:xslt'");
objxml.async = false;
obxml.validateOnParse = true;
obxml.load("books.xml");

objxsl.async = false;
obxsl.load("books.xsl");
// Select nodes of 'date' type using DOM:
var nodes = objxml.selectNodes("//*[ms:type-is('http://www.w3.org/2001/XMLSchema', 'date')]");
result = "<h2>used in a DOM</h2> ";
result += "<h3>nodes of date data types</h3>";
for (i=0; i<nodes.length; i++) {
    result += "<DIV>"+nodes.item(i).nodeName +"=>"+nodes.item(i).text+"</DIV>";
}

// Select nodes of 'date' type using XSLT:
result += "<h2>Used in an XSLT</h2>";
result += objxml.transformNode(objxsl);
document.body.innerHTML = result;

} catch (e) {
    alert(e.description);
}
</script>
</head>

<body onload="init()">
</body>
</html>

Output

Publish_date = 2000-10-01

See Also
**ms:type-local-name([node-set])**

**Function**

Returns the nonqualified name of the XSD type of the current node or the first node (in document order) in the provided node-set.

\[
\text{string } \text{ms:type-local-name}([\text{node-set}])
\]

**Remarks**

For simple types, the \text{type-local-name} function returns a name such as "ID" or "ENTITY". For complex XSD types that have the \text{name} attribute specified, the \text{type-local-name} returns a nonqualified name such as "Class". Nameless types cause the function to return an empty string.

The following sample expression selects all nodes with the XSD built-in primitive data type "string".

"//*[ms:type-local-name()='string']"

**Example**

The following example uses an XSLT template rule to select all the elements in \text{books.xml}, and to output the elements' data types as defined in \text{books.xsd}.

**XML File (books.xml)**

Use \text{books.xml}.

**XSD File (books.xsd)**

Use \text{books.xsd}.

**XSLT File (books.xslt)**

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
The HTML file contains a JScript that handles loading XML, XSLT, and XSD files.

<html>
<head>
<script>
    function init() {
        try {
            var objxsd = new ActiveXObject("Msxml2.XMLSchemaCache.5.0");
            var objxml = new ActiveXObject("Msxml2.DOMDocument.5.0");
            var objxsl = new ActiveXObject("Msxml2.DOMDocument.5.0");

            // namespace uri ("urn:books") must be declared as one of the
            // namespace declarations in the "books.xml" that is an instance
            // of "books.xsd"
            objxsd.add("urn:books", "books.xsd");

            objxml.schemas = objxsd;
            objxml.setProperty("SelectionLanguage", "XPath");
            objxml.setProperty("SelectionNamespaces",
                "xmlns:ms='urn:schemas-microsoft-com:xslt'");
            objxml.async=false;
            objxml.validateOnParse=true;
            objxml.load("books.xml");
        }
        catch (err) {
            console.log(err);
        }
    }
</script>
</head>
<body>
</body>
</html>
objxsl.async=false;
objxsl.load("books.xsl");

result += "<h2>Used in an XSLT</h2>"
result += objxml.transformNode(objxsl);
document.body.innerHTML = result;
}
catch (e) {
    alert(e.description);
}
</script>
</head>

<body onload="init()"
</body>
</html>

Output

x:catalog is of
book is of
author is of string
title is of string
genre is of string
price is of float
publish_date is of date
description is of string

Notice that x:catalog and book elements have nameless types.

See Also

Using XPath Extension Functions for XSD Support | Data Types in Schemas | XML Schema Reference (XSD) | XML Data Types Reference
**ms:type-namespace-uri([node-set])**

**Function**

Returns the namespace URI associated with the XSD data type of a current node or the first node (in document order) in the provided node-set.

`string ms:type-namespace-uri([node-set])`

**Remarks**

For simple XSD types, the `type-namespace-uri` function returns an empty string. For complex XSD types that have the `name` attribute specified, the `type-namespace-uri` function returns a complete URI such as "http://www.example.microsoft.com/my-xsd-types."

The following sample expression returns nodes whose data type has a namespace URI of "PurchaseOrderType".

`"//*[ms:type-namespace-uri()='uri:PurchaseOrderType')]"`

**Example**

The following example uses an XSLT template rule to select from `books.xml` all the elements and to output the elements data types and the namesapce URI as defined in `books.xsd`.

**XML File (books.xml)**

Use `books.xml`.

**XSD File (books.xsd)**

Use `books.xsd`.

**XSLT File (books.xslt)**

`<?xml version='1.0'?>`
The HTML file contains a JScript that handles loading XML, XSLT, and XSD files.

```html
<html>
<head>
  <script>
    function init() {
      try {
        var objxsd = new ActiveXObject("Msxml2/XMLSchemaCache.5.0");
        var objxml = new ActiveXObject("Msxml2/DOMDocument.5.0");
        var objxsl = new ActiveXObject("Msxml2/DOMDocument.5.0");

        // namespace uri ("urn:books") must be declared as one of the namespace declarations in the "books.xml" that is an instance of "books.xsd"
        objxsd.add("urn:books", "books.xsd");
        objxml.schemas = objxsd;
        objxml.setProperty("SelectionLanguage", "XPath");
        objxml.setProperty("SelectionNamespaces",
        "xmlns:ms='urn:schemas-microsoft-com:xslt'");
        objxml.async=false;
        objxml.validateOnParse=true;
        objxml.load("books.xml");
```
objxsl.async=false;
objxsl.load("books.xsl");

result += objxml.transformNode(objxsl);
document.body.innerHTML = result;

} catch (e) {
    alert(e.description);
}
</script>

</head>
<body onload="init()">
</body>
</html>

Output

x:catalog is of "" in ""
book is of "" in ""
author is of "string" in "http://www.w3.org/2001/XMLSchema"
title is of "string" in "http://www.w3.org/2001/XMLSchema"
genre is of "string" in "http://www.w3.org/2001/XMLSchema"
price is of "float" in "http://www.w3.org/2001/XMLSchema"
publish_date is of "date" in "http://www.w3.org/2001/XMLSchema"
description is of "string" in "http://www.w3.org/2001/XMLSchema"

description is of "string" in "http://www.w3.org/2001/XMLSchema"

Notice that x:catalog and book elements have anonymous data types. Thus, both the ms:type-local-name() and ms:type-namespace-uri() function return an empty string.

See Also

Using XPath Extension Functions for XSD Support | Data Types in Schemas | XML Schema Reference (XSD) | XML Data Types Reference
ms:string-compare Function

Performs string comparison. This function is necessary because two strings cannot be compared lexicographically using XPath.

```
number ms:string-compare(string x, string y, [,string language _
 [,string options]]))
```

**Parameters**

- **string x**
  String value for comparison. Nonstring values are converted to a string value before comparison.

- **string y**
  String value for comparison. Nonstring values are converted to a string value before comparison.

- **string language [optional]**
  Defines the language to determine the sort order. This is similar to the `<xsl:sort> lang attribute (for example, "en-US" or "fr-CA"). If this parameter is omitted or an empty string, the language is defined by the system environment (language of the current thread).

- **string options [optional]**
  By default, comparison is case-sensitive, with lowercase first.

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Makes the comparison case-sensitive, uppercase first.</td>
</tr>
<tr>
<td>i</td>
<td>Makes the comparison case-insensitive to all traditional differences between characters, such as case, kana, and width. (Character width only applies to some languages.)</td>
</tr>
</tbody>
</table>

**Return values**

Returns one of the following three numerical values.

- If \( x < y \)  
  The example function returns \(-1\)
Remarks

Comparisons between strings are made in lexicographical (dictionary) order. Specifying an unsupported language or option causes a runtime error to occur. For MSXML components, E_FAIL is returned.

Example

The following code example shows how to compare two strings by using the ms:string-compare function. The example compares three strings of numbers ("1", "2", and "3") with "2". The results of the comparisons are shown in the table that follows the code.

XML File

```xml
<?xml version="1.0"?>
<string-sample>
  <string>1</string>
  <string>2</string>
  <string>3</string>
</string-sample>
```

XSLT File (translate.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ms="urn:schemas-microsoft-com:xslt">
  <xsl:output method="html" omit-xml-declaration="yes"/>
  <xsl:template match="/string-sample">
    <xsl:for-each select="string">
      <xsl:value-of select="ms:string-compare(., '2', 'en-US', 'i')",/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

The results of the comparisons are as follows:

<table>
<thead>
<tr>
<th>x, y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = y</td>
<td>0</td>
</tr>
<tr>
<td>x &gt; y</td>
<td>1</td>
</tr>
</tbody>
</table>
See Also

Data Types in Schemas | XML Schema Reference (XSD) | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
ms:schema-info-available Function

Returns true if XSD information is available for the current node.

boolean ms:xsd-info-available()

Remarks

The following expression returns true for all nodes with XSD-type information.

"/**[ms:schema-info-available()]

Example

The following example uses an XSLT template rule to select all the elements in books.xml, and to output the elements' data types and the namespace URI as defined in books.xsd.

XML File (books.xml)

Use books.xml.

XSD File (books.xsd)

Use books.xsd.

HTML File (books.html)

The HTML file is the same as that listed in the ms:type-namespace-uri([node-set]) Function topic.

XSLT File (books.xslt)

<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
     xmlns:ms="urn:schemas-microsoft-com:xslt"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="text()"/>
  <xsl:output method="html"/>
Output

x:catalog is of "" in ""
book is of "" in ""
author is of "string" in "http://www.w3.org/2001/XMLSchema"
title is of "string" in "http://www.w3.org/2001/XMLSchema"
genre is of "string" in "http://www.w3.org/2001/XMLSchema"
price is of "float" in "http://www.w3.org/2001/XMLSchema"
publish_date is of "date" in "http://www.w3.org/2001/XMLSchema"
description is of "string" in "http://www.w3.org/2001/XMLSchema"

description is of "string" in "http://www.w3.org/2001/XMLSchema"

The output here is the same as that shown in the ms:type-namespace-uri topic, because the schema information is available for every element.

See Also

Using XPath Extension Functions for XSD Support | Data Types in Schemas | XML Schema Reference (XSD) | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
**ms:utc Function**

Converts date/time related values into Coordinated Universal Time and into a fixed (normalized) representation that can be sorted and compared lexicographically.

`string ms:utc(string)`

**Remarks**

This function works only with time instances such as timeInstant, time, date, yearMonth, year, monthDay, day, and month. The `utc()` function does not work with periods of time such as timeDuration or its derived types. For nonstring arguments, this function acts as if a string function were first applied. This function does not depend on the presence of XSD information. If a string cannot be interpreted as a valid XSD date/time-related format, it returns an empty string. The rules for date/time values are:

- A string is interpreted as a time-related XSD format and converted to the following format:

  \[ YYYY-MM-DDTHH:NN:SS.III \]

  *where*

<table>
<thead>
<tr>
<th>Characters</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY</td>
<td>A four digit year (for example, 2001).</td>
</tr>
<tr>
<td>MM</td>
<td>The number of a month (for example, 01 for January).</td>
</tr>
<tr>
<td>DDD</td>
<td>A day of a month. Three digits are used to accommodate the ability to point out the exact day of the year (for example, in &quot;2001-04-123&quot;, 123 points out the 123rd day of the year).</td>
</tr>
<tr>
<td>HH</td>
<td>A two-digit hour (for example, &quot;00-23&quot;).</td>
</tr>
<tr>
<td>SS</td>
<td>Seconds (for example, &quot;00-59&quot;).</td>
</tr>
<tr>
<td>III</td>
<td>Milliseconds (for example, &quot;000-999&quot;). If no milliseconds are specified, dashes representing milliseconds are right-truncated.</td>
</tr>
</tbody>
</table>
• For negative time and years beyond 9999, the utc() function returns an empty string because negative timeInstances, periods, invalid Gregorian dates and other cases described in the following string constraints section will not be able to compare correctly.

• Fields indicating recurrence are replaced with the same number of "-" characters.

• Unspecified fields are replaced with "-" characters.

**String Comparison Constraints:**

• Lexicographical comparison does not work for sets of mixed AC and BC dates, and gives a reverse order for BC dates. Lexicographical comparison does not work for years greater than 9999.

**See Also**

[Data Types in Schemas](#) | [XML Schema Reference (XSD)](#) | [XML Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
ms:namespace-uri Function

Resolves the prefix part of a qualified name into a namespace URI.

\textit{string} ms:namespace-uri(string)

\textbf{Remarks}

The \textit{ms:namespace-uri()} function assumes that the parameter is a qualified name (for example, \texttt{q:mine} or \texttt{coffee:beans}) and attempts to resolve its prefix in the context of the current node. If the argument is not a qualified name or a noncolon name, this function returns an empty string. For nonstring arguments, this function behaves as if a \textit{string()} function were applied.

\textbf{Example}

The following example uses an XSLT template rule to select all the elements in \texttt{books.xml}, and to output the elements' data types and the namesapce URI as defined in \texttt{books.xsd}.

\textbf{XML File (books.xml)}

Use the XML file in \texttt{Using XPath Extension Functions for XSD Support}.

\textbf{XSD File (books.xsd)}

Use the XSD file in \texttt{Using XPath Extension Functions for XSD Support}.

\textbf{HTML File (books.html)}

The HTML file is the same as that listed in the \texttt{ms:type-namespace-uri([node-set]) Function} topic.

\textbf{XSLT File (books.xsl)}

\begin{verbatim}
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ms="urn:schemas-microsoft-com:xslt"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.w3.org/2001/XMLSchema-instance">
  <xsl:template match="/">
    <xsl:copy-of select="/"/>
  </xsl:template>
</xsl:stylesheet>
\end{verbatim}
Output

(ms:) urn:books:catalog
(std:) urn:books:catalog
(name():) x:catalog

The first line of the output is the result returned by the ms:namespace-uri(string) plus ms:local-name(string). The second line is the results from the standard XPath function, namespace-uri(nodeset), local-name(nodeset). The third line is the result from the standard XPath function, name(nodeset).

See Also

Data Types in Schemas | XML Schema Reference (XSD) | XML Data Types Reference
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
ms:local-name Function

Resolves the prefix part of a qualified name into a namespace URI.

\[ \text{string } \text{ms:local-name}(\text{string}) \]

Remarks

The ms:local-name() function assumes that the parameter is a qualified name (for example, q:mine or coffee:beans) and returns the local name within the context of the current node. If the argument is not a qualified name or a noncolon name, this function returns an empty string. For nonstring arguments, this function behaves as if a string() function were applied.

Example

See the example for the ms:namespace-uri function.

See Also

[Data Types in Schemas](#) | [XML Schema Reference (XSD)](#) | [XML Data Types Reference](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XPath Refer
**ms:number Function**

Takes a string argument in XSD format and converts it into an XPath number.

```xml
number ms:number(string)
```

**Remarks**

The `ms:number()` function performs conversion the same way as the native XPath `number()` function, with the exceptions that the `ms:number()` function converts "-INF" and "INF" into minus and plus infinity, and mathematical notation such as (1.2e-3) into numbers, not NaNs. For a complete set of numeric formats that can be processed, see the XSD Specification. If the `ms:number()` function is unable to convert a string, it returns a NaN.

**See Also**

[Data Types in Schemas](#) | [XML Schema Reference (XSD)](#) | [XML Data Types Reference](#)
**ms:format-date Function**

Converts standard XSD date formats to characters suitable for output.

```
string ms:format-date(string datetime, string format [,locale])
```

**Parameters**

- `string datetime` contains a date-related value in XSD format. For nonstring arguments, this function behaves as if a `string()` function were applied. If the argument is not a date, the output is an empty string. If it is a valid XSD date and some fields are empty, this function attempts to fill unspecified fields.

- `string format` Contains a format string, according to Win32 API `GetDateFormat` functions.

- `locale [optional]` Defines the locale to format the data for (for example, "EN_us").

**Remarks**

The following lists the format characters for `ms:format-date()`.

<table>
<thead>
<tr>
<th>Character(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Months as 1-12</td>
</tr>
<tr>
<td>MM</td>
<td>Months as 01-12</td>
</tr>
<tr>
<td>MMM</td>
<td>Months as Jan-Dec</td>
</tr>
<tr>
<td>MMMM</td>
<td>Months as January-December</td>
</tr>
<tr>
<td>d</td>
<td>Days as 1-31</td>
</tr>
<tr>
<td>dd</td>
<td>Days as 01-31</td>
</tr>
<tr>
<td>ddd</td>
<td>Days as Sun-Sat</td>
</tr>
<tr>
<td>dddd</td>
<td>Days as Sunday-Saturday</td>
</tr>
<tr>
<td>y</td>
<td>Years as 0-99</td>
</tr>
<tr>
<td>yy</td>
<td>Years as 00-99</td>
</tr>
</tbody>
</table>
yyyy Years as 1900-9999

Note that this format does not allow time zones; does not allow the inclusion of arbitrary text into the formatted string; does not allow centuries; does not allow suffixes like May 2^{nd} or April 1^{st}.

Example

**XML Document (Sample.xml)**

```xml
<?xml version='1.0'?>
<XMLSamples xml:space="preserve"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <FileDetails>
  <Author>Robert Brown</Author>
  <Date dt:dt="datetime">2000-02-16T15:56:00</Date>
 </FileDetails>
</XMLSamples>
```

**XSLT Style Sheet (Sample.xsl)**

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:ms="urn:schemas-microsoft-com:xslt"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <xsl:template match="/">
 <HTML>
  <HEAD>
  </HEAD>
  <BODY>
   <xsl:for-each select="XMLSamples/FileDetails/Date">
    <DIV>
     Date Unedited:
     <xsl:value-of select="."/>
    </DIV>
    <DIV>
     Date Edited:
     <xsl:value-of select="ms:format-date(. , 'MMM dd, yyyy')"/>
    </DIV>
   </xsl:for-each>
  </BODY>
 </HTML>
</xsl:template>
```
Output

The output HTML page shows the following entries:

Date Unedited: 2000-02-16T15:56:00  
Date Edited: Feb 16, 2000

See Also

Data Types in Schemas | XML Schema Reference (XSD) | XML Data Types Reference
ms:format-time Function

Converts standard XSD time formats to characters suitable for output.

```plaintext
string ms:format-time(string datetime, string format [,locale])
```

**Parameters**

- `string datetime`
  - Contains a time-related value in XSD format. For nonstring arguments, this function behaves as if a `string()` function were applied. If the argument is not a time, the output is an empty string. If it is a valid XSD time and some fields are empty, this function attempts to fill unspecified fields.

- `string format`
  - Contains a format string, according to Win32 API GetTimeFormat functions.

- `locale [optional]`
  - Defines the locale to format the data for (for example, "EN_us").

**Remarks**

The following lists the format characters for `ms:format-time()`.

<table>
<thead>
<tr>
<th>Character(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Hours as 0-12</td>
</tr>
<tr>
<td>hh</td>
<td>Hours as 00-12</td>
</tr>
<tr>
<td>H</td>
<td>Hours as 0-23</td>
</tr>
<tr>
<td>HH</td>
<td>Hours as 00-23</td>
</tr>
<tr>
<td>m</td>
<td>Minutes as 0-59</td>
</tr>
<tr>
<td>mm</td>
<td>Minutes as 00-59</td>
</tr>
<tr>
<td>s</td>
<td>Seconds as 0-59</td>
</tr>
<tr>
<td>ss</td>
<td>Seconds as 00-59</td>
</tr>
<tr>
<td>tt</td>
<td>Insert AM or PM, display hours as 12-hour clock</td>
</tr>
</tbody>
</table>
Insert A or P, display hours as 12-hour clock

Example

XML Document: (Sample.xml)

```xml
<?xml version='1.0'?>
<XMLSamples xml:space="preserve" xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <FileDetails>
    <Author>Robert Brown</Author>
    <Date dt:dt="datetime">2000-02-16T15:56:00</Date>
  </FileDetails>
</XMLSamples>
```

XSLT Style sheet (Sample.xsl)

```xml
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ms="urn:schemas-microsoft-com:xslt"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <xsl:template match="/">
    <HTML>
      <HEAD>
      </HEAD>
      <BODY>
        <xsl:for-each select="XMLSamples/FileDetails/Date">
          <DIV>
            Time Unedited:
            <xsl:value-of select="."/>
          </DIV>
          <DIV>
            Time Edited:
            <xsl:value-of select="ms:format-time(., 'hh:mm:ss tt')"/>
          </DIV>
        </xsl:for-each>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Try It

1. Follow the instructions given in [Initiate XSLT in a Script](#)
2. Supply **Sample.xml** and **Sample.xsl** above as the input for the required .xml and .xsl files.
3. Run the script. You should get the following output.

**Output**

The output HTML page shows the following entries:

*Time Unedited*: 2000-02-16T15:56:00  
*Time Edited*: 03:56:00PM

**See Also**

[Data Types in Schemas](#) | [XML Schema Reference (XSD)](#) | [XML Data Types Reference](#)
SAX2 Developer's Guide

This guide introduces the Microsoft COM implementation of SAX2, the latest version of the Simple API for XML. SAX2 is a publicly developed standard for event-based parsing of XML documents. This guide, along with the reference material that accompanies it, describes the Microsoft COM implementation of the SAX2 standard. The Microsoft SAX2 implementation provides both Microsoft® Visual Basic® and Microsoft Visual C++® interfaces.

The following topics provide more information about creating applications with SAX2.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter Kit</td>
<td>This section explains the benefits of SAX2 and helps you choose between SAX2 and the XML Document Object Model (DOM). This section also offers ideas for building SAX2 applications.</td>
</tr>
<tr>
<td>Concepts</td>
<td>This section introduces some of the fundamental concepts pertinent to events-based parsing.</td>
</tr>
<tr>
<td>How Do I?</td>
<td>This section helps you build SAX2 applications. It provides tutorials for using events-based parsers with applications that range from simple searches to more complex chain filtering and atomic lookups.</td>
</tr>
</tbody>
</table>

See Also

SAX2 Reference | Using XSLT with SAX
Starter Kit

This starter kit is designed to provide tools and information for the latest version of the Simple API for XML (SAX2).

- What is SAX?
- Frequently Asked Questions About SAX2
- JumpStart for Creating a SAX2 Application
- Creating C++ Applications Using SAXAppWizard
What is SAX?

The Simple API for XML (SAX) is a publicly developed standard for the events-based parsing of XML documents. SAX defines an abstract programmatic interface that models the XML information set (infoset) through a linear sequence of familiar method calls.

Although SAX and DOM may seem different, each is simply a different projection of the XML infoset onto programmatic types. Because SAX does not demand resources for an in-memory representation of the document, it is a good alternative to the Document Object Model (DOM). SAX facilitates the search of large documents to extract small pieces of information and allows you to abort processing after the information is located. SAX can be used to build DOM trees (or portions of DOM trees); conversely, developers can traverse DOM trees and emit SAX streams.

However, the performance benefits of SAX come at a price. For example, in the case of advanced queries, SAX can be burdensome because of the complexities involved in managing context during processing. In these cases, most developers use the DOM or some combination of SAX and DOM.

SAX support in MSXML 5.0 includes methods for completing transformations and validation, which result in better messaging support for end users. This improves server-side transformations of documents as small as 10-20 kilobytes (KB) and the processing of documents of 1 megabyte (MB) or more.

See Also

Why Should I Use SAX? | Frequently Asked Questions About SAX2 | Using XSLT with SAX

Other Resources

Megginson Technologies
Frequently Asked Questions about SAX2

The following are some frequently asked questions about the Simple API for XML (SAX).

What are the system requirements for SAX2?

Can I pass a BSTR instead of a URL to the SAXXMLReader?

Does SAX support validation?

When would I want to use standalone SAX validation (i.e. the MXValidator CoClass) instead of the integrated validation feature provided on the SAX reader?

Why does the MXValidator CoClass contain methods and properties that are not implemented?

Why is white space reported as characters()? Why isn't ignoreWhitespace called?

How do I get XML header information?

How can I use SAXXMLWriter from scripts?

Can I continue parsing if my Visual Basic program breaks because the XML file has errors?

Can I use the same instance of SAXXMLReader to parse XML files sequentially?

How can I write from SAXXMLWriter in a memory buffer in nonUnicode encoding?

How can I reset SAXXMLWriter to create a new string?
How do I avoid appending a new XML document to the previous one?

How can I tell if attribute values have an entity reference?

Can I find the order of element attributes?

How do I handle errors with SAX?

What are the system requirements for SAX2?

The system requirements for SAX2 are the same as those for Installing and Registering the MSXML 5.0 SDK.

Return to top

Does SAX support validation?

Yes. With MSXML 5.0, SAX supports validation to XSD schemas but does not support validation using Document Type Definition (DTD) files.

To validate documents when using SAX, you set the validation flag on the SAXXMLReader through the putFeature method.

When setting this feature, the feature name is "schema-validation" and its value is set to True like this:

```python
oReader.putFeature("schema-validation", True)
```

This feature is read-only during parsing and read/write otherwise.

For more information, see Validate Documents Using SAX.

Return to top

When would I want to use standalone SAX validation (i.e. the MXValidator CoClass) instead of the integrated validation feature provided on the SAX reader?

SAXXMLReader and MXValidator behave in very similar ways when used for
validating SAX document streams. The only difference is that the SAX reader might provide some additional controls that the \texttt{MXValidator} does not. Also, unlike the SAX reader, which allows you to toggle the error-reporting mode, \texttt{MXValidator} always operates in "exhaustive-errors" mode and reports all errors.

\texttt{MXValidator} is most useful when the data to be parsed is not in XML form, such as comma-delimited text, a Microsoft Word document, or a C-style data structure. The SAX reader cannot read any of these formats. If you need to parse non-XML data, you can implement your own data transformer using SAX interfaces, and connect to \texttt{MXValidator} to verify the data stream. For an example, see the \texttt{putProperty Method (MXValidator)} sample.

**Why does the MXValidator CoClass contain methods and properties that are not implemented?**

\texttt{MXValidator} inherits from the \texttt{IMXFilter} interface, which is intended to provide a generic front end for SAX event filtering. For \texttt{MXValidator}, only the ability to get and put features and properties is needed. Currently, only the \texttt{contentHandler} and \texttt{errorHandler} properties are implemented, but other SAX filters might be developed in the future.

**Can I pass a BSTR, instead of a URL, to the SAXXMLReader?**

You can pass a \texttt{VARIANT} containing a BSTR to \texttt{ISAXXMLReader::parse(VARIANT)}. In this case, the encoding is UTF-16.

**Why is white space reported as characters()? Why isn't ignorableWhitespace called?**

White space can occur in several places, for example, in an element without character data, which contains only child nodes and white space. To ignore white space, the parser must be able to distinguish those cases. The SAX parser is a nonvalidating parser and cannot distinguish those cases, so \texttt{ignoreableWhitespace()} never gets called. Nonvalidating parsers treat all white space between elements as characters.
How do I get XML header information?

The XML header contains version and encoding information, for example, `<?xml version="1.0" encoding="UTF-8"?>`. To get XML header information, call ISAXXMLReader::getProperty([in] const wchar_t * pwchName, [out, retval] VARIANT * pvarValue); and pass one of following three property values:

"xmldecl-encoding"
"xmldecl-version"
"xmldecl-standalone"

**Note** The "xmldecl-encoding", "xmldecl-version", and "xmldecl-standalone" properties provide information about the presence and content of the XML header. The information is available only when SAXXMLReader reads and parses the XML document. After processing, the control returns to the application, and this information is no longer available.

XML header information was designed for low-level reader and parser use, not for applications.

To get the processing instruction, implement a ContentHandler that supports ISAXContentHandler and handles the processingInstruction event.

How can I use SAXXMLWriter from scripts?

SAXXMLReader implements IVBSAXXMLReader, which is accessible from scripts. You can call handler events directly from SAXXMLWriter and generate XML without the reader.

Can I continue parsing if my Visual Basic program breaks because the XML file has errors?
All parser errors are fatal (E_FAIL). However, the classification of parsing errors in SAX is independent from the classification of errors in Microsoft® Visual Basic®. In SAX, a fatal parsing error means only that parsing cannot continue; in Visual Basic, a fatal error means that the application cannot continue. Use the ON ERROR statement in Visual Basic.

**Return to top**

**Can I use the same instance of SAXXMLReader to parse XML files sequentially?**

You can use the same instance of SAXXMLReader to parse two XML files sequentially, but not in different threads. The MSXML implementation does not support multithreaded use. AddRef/Release are not multithread-safe and there is no locking on any of the API entry points.

However, you can use two instances of SAXXMLReader in two threads, and parse two different XML files, as long as nothing gets shared.

**Return to top**

**How can I write from SAXXMLWriter in a memory buffer in nonUnicode encoding?**

If "memory buffer" is a Visual Basic string, you cannot write to it because it is always in Unicode format. However, you can provide an IStream/ISequentialStream object, which writes to a memory buffer. XML will be generated the same way as for output into a file.

**Return to top**

**How can I reset SAXXMLWriter to create a new string?**

**To reset SAXXMLWriter to create a new string:**

1. Generate an XML document.
2. Use a string object from the writer.
How do I avoid appending a new XML document to the previous one?

To reset XML writer to create a new string, reset the output property. Internally, the flush method of IMXXMLWriter will be called.

How can I tell if attribute values have an entity reference?

There is no indication of whether attribute values have an entity reference.

Can I find the order of element attributes?

The order of attributes is not important in XML, and is therefore not exposed. Enumeration with attributes may follow the original order of the attributes.

How do I handle errors with SAX?

ISAXErrorHandler/IVBSAXErrorHandler provides the basic interface for handling parsing errors. Currently, all errors are fatal.

In C++, a fatal error will result in returning a value other than S_OK HRESULT from the parse or parseURL method.

In Visual Basic, the On Error statement handles exceptions.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
JumpStart for Creating a SAX2 Application

This section provides a hands-on introduction to the latest version of the Simplified API for XML (SAX2) and includes a step-by-step process for creating a basic SAX2 application. This section provides two tutorials:

- [JumpStart for Creating a SAX2 Application with Visual Basic](#)
- [JumpStart for Creating a SAX2 Application with C++](#)
- [Resource Listings for JumpStart for Creating a SAX2 Application](#)
JumpStart for Creating a SAX2 Application with Visual Basic

To help you get started building Simple API for XML (SAX2) applications with Microsoft® Visual Basic®, this section shows you how to quickly build an application that uses SAX2 to read an XML document and print the document's contents to a text box.

To run this tutorial, you will need the following:

- MSXML version 3.0 or later.
- Visual Basic 6.0.

You can download the JumpStart application from the MSDN Code Center. This downloadable version contains more files and differs slightly from the sample application covered in this topic.

This section includes the following topics:

- Overview of the JumpStart Application
- Implementing the ContentHandler
- Implementing the ErrorHandler
- Creating the Main Form
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Overview of the JumpStart Application (Visual Basic)

The Simple API for XML (SAX2) is a push-model parser. When SAX2 parses a document, the SAXXMLReader (reader) reads the document and passes a series of events to the implemented event handlers. The SAX2 reader generates several categories of events, including:

- those that occur in the content of an XML document.
- those that occur in the document type definition (DTD).
- those that occur as errors.

To handle such events, you implement a corresponding handler class that contains methods to process the appropriate events. You implement handlers only for those events you wish to process. If you do not implement a handler for a specific type of event, the reader simply ignores that event.

The JumpStart application covered in this section contains the three components that you most often implement for SAX2 applications in Microsoft® Visual Basic®.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentHandler</td>
<td>Implements the IVBSAXContentHandler interface. The ContentHandler is a class that provides methods for processing the main content of an XML document. When SAX2 parses a document, the reader passes a series of events to the ContentHandler. For example, for each element in a document, the reader passes the startElement, characters, and endElement events. To handle these events, you add code to the methods in the ContentHandler to process the information passed by the reader.</td>
</tr>
<tr>
<td>ErrorHandler</td>
<td>Implements the IVBSAXErrorHandler interface. The ErrorHandler is a class that provides methods for processing error events passed by the reader. At this time, the reader invokes the fatalError method only.</td>
</tr>
</tbody>
</table>
main form

Provides the user interface for the application. This consists of two text boxes, one for entering a file name and another for displaying results. This interface also has Parse and Exit buttons. In addition to providing the user interface, the main form also contains code that creates instances of the SAXXMLReader, ContentHandler, and ErrorHandler; sets the handler classes to the reader; sets the source for parsing the document; and starts the parse operation.
Implementing the ContentHandler (Visual Basic)

For the JumpStart application, you create the ContentHandler by adding a class that implements the IVBSAXContentHandler interface. Although creating a class may sound complex, Microsoft® Visual Basic® automates most of the process.
Opening a New Project

Before you can create a class based on the IVBSAXContentHandler interface, you must create a new project and then reference that project to Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office.

To create a new project

- Open Visual Basic 6.0 and, in the **New Project** dialog box, double-click **Standard EXE**.
Referencing MSXML

You must now instruct your application to use the MSXML2 type library provided with MSXML 5.0.

To create a reference to MSXML 5.0

1. On the Project menu, click References.
2. In the Available References list, select Microsoft XML, v5.0, and then click OK.
Creating and Implementing the Handler Class

You are now ready to create and implement the new `ContentHandler` class.

To create a new class

1. On the **Project** menu, click **Add Class Module**.
2. In the **Add Class Module** dialog box, double-click **Class Module**.
3. On the **View** menu, select **Properties Window**.
4. In the Properties Window, enter "ContentHandlerImpl" for the **Name** property.

To implement the `ContentHandlerImpl` Class

1. In the Class Window, type "Implements", and then press the space bar.

   A drop down list appears with the all the available classes. (This list includes interfaces that are actually abstract classes.)

2. In the drop-down list, double-click `IVBSAXContentHandler`. This is the interface you implement to create the `ContentHandlerImpl` class.
3. In the **Object** drop-down list on the left side of the Class Window, click `IVBSAXContentHandler`.

   This automatically adds code to the Class Window.

4. In the **Procedure** drop-down list on the right-hand side of the Class Window, select the `IVBSAXContentHandler` method or property that you want to add to this class. (You have to select each method even though you do not need to add code to each one.)

   For example, select `startElement` to add this method to the `ContentHandlerImpl` class. (When parsing an XML document, the reader will invoke this method each time it encounters a new element in the document.) After you select `startElement` from the list, the following code appears in the Class Window:

```vbnet
Private Sub IVBSAXContentHandler_startElement_
    (strNamespaceURI As String, _
```
strLocalName As String, _
strQName As String, _
ByVal oAttributes As MSXML2.IVBSAXAttr

End Sub

5. To enable the method or event selected in Step 4 to handle the corresponding event, you need to add the appropriate code to that method or property.

For the `startElement` method added in Step 4, enter the code required to make the method look like this:

```vba
Private Sub IVBSAXContentHandler_startElement _
(strNamespaceURI As String, _
strLocalName As String, _
strQName As String, _
ByVal attributes As MSXML2.IVBSAXAttr)
Dim i As Integer
Form1.Text2.text = Form1.Text2.text & "<" & strLocalName
For i = 0 To (attributes.length - 1)
    Form1.Text2.text = Form1.Text2.text & " " _
    & attributes.getLocalName(i) & "=" & attributes.getValue(i) & ""
Next
Form1.Text2.text = Form1.Text2.text &">
If strLocalName = "qu" Then
    Err.Raise vbObjectError + 1, "ContentHandler.startElement" _
    "Found element <qu>"
End If
End Sub
```

6. To handle additional events passed by the reader, repeat steps 4 and 5 for each method or property that you want to add to the `ContentHandlerImpl` class.
Complete Code for the ContentHandlerImpl Class

The preceding instructions explain how to create a new project with a reference to Microsoft XML, v5.0, and how to create a ContentHandler class called ContentHandlerImpl by implementing the IVBSAXContentHandler interface. The following shows the complete code for the ContentHandlerImpl class.

Option Explicit

Implements IVBSAXContentHandler

Private Sub IVBSAXContentHandler_startElement_(strNamespaceURI As String, _
strLocalName As String, _
strQName As String, _
ByVal attributes As MSXML2.IVBSAXAttributes)

Dim i As Integer

Form1.Text2.text = Form1.Text2.text & "<" & strLocalName

For i = 0 To (attributes.length - 1)
    Form1.Text2.text = Form1.Text2.text & " " & _
    attributes.getLocalName(i) & "=" & _
    attributes.getValue(i) & ""
Next

Form1.Text2.text = Form1.Text2.text &">

If strLocalName = "qu" Then
    Err.Raise vbObjectError + 1, "ContentHandler.startElement", _
    "Found element <qu>"
End If

End Sub

Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As String, _
strLocalName As String, strQName As String)

Form1.Text2.text = Form1.Text2.text & "</" & strLocalName & "">

End Sub

Private Sub IVBSAXContentHandler_characters(text As String)

text = Replace(text, vbCrLf, vbCrLf)

End Sub
As you can see from the sample code, not all methods or properties have custom code added to them. When you implement a handler for the Simple API for XML (SAX2), you can add only the methods or properties that you need.

In the next section, you create the Error Handler for the JumpStart application.
Implementing the ErrorHandler

For the JumpStart application, you create the `ErrorHandler` by implementing the `IVBSAXErrorHandler` interface.

**To create the ErrorHandlerImpl class**

1. On the *Project* menu, click *Add Class Module*.
2. In the *Add Class Module* dialog box, double-click *Class Module*.
3. On the *View* menu, select *Properties Window*.
4. In the Properties Window, enter "ErrorHandlerImpl" for the *Name* property.

**To implement the ErrorHandlerImpl class**

1. In the Class Window, type "Implements", press the spacebar, and then double-click `IVBSAXErrorHandler` from the drop-down list that appears.
2. In the *Object* drop-down list on the left side of the Class Window, select `IVBSAXErrorHandler`.
3. In the *Procedure* drop-down list on the right side of the Class Window, select a method or property to add to the class.
4. Add the desired code to the selected method or property.
5. Repeat steps 3 and 4 until you have added the desired methods and properties.

**Note** If you have trouble implementing methods and procedures for the `ErrorHandlerImpl` class, refer back to *Implementing the ContentHandler* to see how methods and properties were implemented for the `ContentHandlerImpl` class.
Complete Code for the ErrorHandlerImpl Class

The following is the complete code for the ErrorHandlerImpl class of the JumpStart application.

Option Explicit

Implements IVBSAXErrorHandler

Private Sub IVBSAXErrorHandler_fatalError(ByVal lctr As IVBSAXLocator, _
msg As String, ByVal errCode As Long)
    Form1.Text2.text = Form1.Text2.text & "*** error *** " & msg
End Sub

Private Sub IVBSAXErrorHandler_error(ByVal lctr As IVBSAXLocator, _
msg As String, ByVal errCode As Long)
End Sub

Private Sub IVBSAXErrorHandler_ignorableWarning(ByVal oLocator As MSXML2.IVBSAXLocator, _
strErrorMessage As String, ByVal nErrorCode As Long)
End Sub

In the preceding code, only the fatalError method contains code for handling events. For this version of Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office, only the fatalError method is called for IVBSAXErrorHandler.
Creating the Main Form

The main form provides the user interface for the JumpStart application and consists of four elements.

<table>
<thead>
<tr>
<th>Interface Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text1</td>
<td>A text box for entering the file name of the file to parse.</td>
</tr>
<tr>
<td>Command1</td>
<td>A button to start the parse process.</td>
</tr>
<tr>
<td>Command2</td>
<td>A button to exit the application.</td>
</tr>
<tr>
<td>Text2</td>
<td>A text box that displays the results of the parse operation.</td>
</tr>
</tbody>
</table>

In addition to providing the interface elements, the main form also provides code that creates the required instances of the classes and starts the parse process.

To

Create an instance of the reader (SAXXMLReader)

Dim reader As New SAXXMLReader

Create an instance of ContentHandlerImpl and set it to receive events from the reader

Dim contentHandler As New ContentHandlerImpl
Set reader.contentHandler = contentHandler

Create an instance of ErrorHandlerImpl and set it to receive events from the reader

Dim errorHandler As New ErrorHandlerImpl
Set reader.errorHandler = errorHandler

Start the parse process

reader.parseURL (Text1.text)
Complete Code for the Main Form

The following is the complete code for the main form of the JumpStart application.

Option Explicit

Private Sub Command1_Click()
    Dim reader As New SAXXMLReader 'Reads the XML document
    Dim contentHandler As New ContentHandlerImpl 'Receives parsing e
    Dim errorHandler As New ErrorHandlerImpl 'Receive error e
    Text2.text = ""
    Set reader.contentHandler = contentHandler 'They work togethe
    Set reader.errorHandler = errorHandler 'They also work to
    On Error GoTo 10
    reader.parseURL (Text1.text) 'Parse the document
    Exit Sub 'That's all, folks!
    10: Text2.text = Text2.text & "*** Error *** " & Err.Number _
        & " : " & Err.Description
End Sub

Private Sub Command2_Click() End End Sub

The Exit Sub and line marked 10: are required only if you want to handle processing outside the ErrorHandler. Otherwise, you can use On Error Resume Next.

To run the JumpStart application

1. Download the JumpStart application, available from the MSDN Code Center, to your computer.
2. On your computer, double-click VBSAXExample.exe.
3. In the File name box, type "Test.xml" and then click Parse. The JumpStart application parses the file and displays the results in the Text2 text box.

You can use this introduction to the Simple API for XML (SAX2) as a starting point for writing your own applications. For more information about the Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office
implementation of the SAX2 interfaces, see the [SAX2 Reference](#).
JumpStart for Creating a SAX2 Application with C++

As a quick introduction to how Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office implements the Simple API for XML (SAX2), and to help you get started building applications using the SAX2 interfaces, this section shows you how to build a basic C++ application that reads an XML document and prints the document's tags to the console.

To run this tutorial, you will need the following:

- MSXML version 3.0 or later.
- The JumpStart application, available from the [MSDN Code Center](https://msdn.microsoft.com).

This section includes the following topics:

- Overview of the JumpStart Application
- Implementing the ContentHandler
- Creating the Header File
- Creating the MyContent Class
- Creating the Main Program
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Overview of the JumpStart Application (C++)

The Simple API for XML (SAX2) is a push-model parser. When SAX2 parses a document, the SAXXMLReader (reader) reads the document and passes a series of events to the event handlers that you implement. The SAX2 reader generates several categories of events, including:

- Those that occur in the content of an XML document.
- Those that occur in the document type definition (DTD).
- Those that occur as errors.

To handle such events, you implement a corresponding handler class that contains methods to process the appropriate events. You implement handlers only for those events you wish to process. If you do not implement a handler for a specific type of event, the reader simply ignores that event.

You implement two main components in this JumpStart application.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentHandler</td>
<td>Implements the ISAXContentHandler interface. The ContentHandler is a class that provides methods for processing the main content of an XML document. When SAX2 parses a document, the reader passes a series of events to the ContentHandler. For example, for each element in a document, the reader passes startElement, characters, and endElement events. You handle these events by adding code to the methods in the ContentHandler to process the information passed by the reader.</td>
</tr>
</tbody>
</table>
| main program     | - Creates an instance of ISAXXMLReader (reader).  
- Creates an instance of the ContentHandler.  
- Sets the ContentHandler to the reader.  
- Sets the source for the XML document. |
To create the JumpStart application, you must first implement a handler class extending the ISAXContentHandler interface. With C++, you can create a class and not a full-featured COM object. In the methods of this class, you tell the application what to do when it receives notification of an event. After you implement the ContentHandler class, you create the main program that creates an instance of the SAXXMLReader, sets the ContentHandler, and starts the parse operation.
Implementing the ContentHandler (C++)

The first step when creating a Simple API for XML (SAX2) application is to implement handler classes. This example implements only the ContentHandler that is derived from the ISAXContentHandler interface. Implementing the ContentHandler is described in the following topics.

- Creating the Header File
- Creating the MyContent Class
Creating the Header File

When it comes to using the Simple API for XML (SAX2), the most useful handler class is the `ContentHandler`. You derive this class from the `ISAXContentHandler` interface.

To use the SAX2 interfaces in Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office, you must declare them with the following code. The following shows importing msxml5.dll.

```cpp
#import <msxml5.dll> raw_interfaces_only
using namespace MSXML2;
```

**Note** For the JumpStart application, you declare the interfaces in the `StdAfx.h` file.

To implement the `ContentHandler`, create the header file, named "MyContent.h", as shown in the following sample code.

```cpp
#include "SAXContentHandlerImpl.h"

class MyContent : public SAXContentHandlerImpl
{
public:
    MyContent();
    virtual ~MyContent();

    virtual HRESULT STDMETHODCALLTYPE startElement(
        /* [in] */ wchar_t __RPC_FAR *pwchNamespaceUri,
        /* [in] */ int cchNamespaceUri,
        /* [in] */ wchar_t __RPC_FAR *pwchLocalName,
        /* [in] */ int cchLocalName,
        /* [in] */ wchar_t __RPC_FAR *pwchQName,
        /* [in] */ int cchQName,
        /* [in] */ ISAXAttributes __RPC_FAR *pAttributes);

    virtual HRESULT STDMETHODCALLTYPE endElement(
        /* [in] */ wchar_t __RPC_FAR *pwchNamespaceUri,
        /* [in] */ int cchNamespaceUri,
        /* [in] */ wchar_t __RPC_FAR *pwchLocalName,
        /* [in] */ int cchLocalName,
        /* [in] */ wchar_t __RPC_FAR *pwchQName,
        /* [in] */ int cchQName);
};
```
virtual HRESULT STDMETHODCALLTYPE startDocument();

private:
    void prt(
        /* [in] */ const wchar_t * pwchFmt,
        /* [in] */ const wchar_t __RPC_FAR *pwchVal,
        /* [in] */ int cchVal);
    int idnt;
};

#endif // !defined(AFX_MYCONTENT_H__E1B3AF99_0FA6_44CD_82E3_55719F9E
Creating the MyContent Class

After you create the MyContent.h file, the next step is to implement the ContentHandler class, named MyContent.

```c
#include "stdafx.h"
#include "MyContent.h"

/////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////

MyContent::MyContent()
{
    idnt = 0;
}

MyContent::~MyContent()
{
}

HRESULT STDMETHODCALLTYPE MyContent::startElement(
    /* [in] */ wchar_t __RPC_FAR *pwchNamespaceUri,
    /* [in] */ int cchNamespaceUri,
    /* [in] */ wchar_t __RPC_FAR *pwchLocalName,
    /* [in] */ int cchLocalName,
    /* [in] */ wchar_t __RPC_FAR *pwchQName,
    /* [in] */ int cchQName,
    /* [in] */ ISAXAttributes __RPC_FAR *pAttributes)
{
    HRESULT hr = S_OK;
    int l;
    printf("%*s", 3 * idnt++, "");
    prt(L"<%s", pwchLocalName, cchLocalName);
    pAttributes->getLength(&l);
    for ( int i=0; i<l; i++ ) {
        wchar_t * ln, * vl; int ln1, vl1;
        pAttributes->getLocalName(i, &ln, &ln1);
        pAttributes->getValue(i, &vl, &vl1);
        prt(L" %s = ", ln, ln1);
        prt(L" %s = ", vl, vl1);
    }
    printf(">");
}
```
// A little example, how to abort parse
if ( wcscmp(pwchLocalName,L"qu",2) == 0 ) {
    printf("\n<qu> tag encountered, parsing aborted.");
    hr = E_FAIL;
}
return hr;

HRESULT STDMETHODCALLTYPE MyContent::endElement(
    /* [in] */ wchar_t __RPC_FAR *pwchNamespaceUri,
    /* [in] */ int cchNamespaceUri,
    /* [in] */ wchar_t __RPC_FAR *pwchLocalName,
    /* [in] */ int cchLocalName,
    /* [in] */ wchar_t __RPC_FAR *pwchQName,
    /* [in] */ int cchQName)
{
    printf("\n%*s",3 * --idnt, "");
    prt(L"</%s>",pwchLocalName,cchLocalName);
    return S_OK;
}

HRESULT STDMETHODCALLTYPE MyContent::startDocument()
{
    printf("<?xml version="1.0" ?>");
    return S_OK;
}

void MyContent::prt(
    /* [in] */ const wchar_t * pwchFmt,
    /* [in] */ const wchar_t __RPC_FAR *pwchVal,
    /* [in] */ int cchVal)
{
    static wchar_t val[1000];
cchVal = cchVal>999 ? 999 : cchVal;
wcsncpy( val, pwchVal, cchVal ); val[cchVal] = 0;
wprintf(pwchFmt,val);
}
Creating the Main Program

In this topic, you create a main program that:

- Provides a command prompt interface.
- Creates a parser by instantiating a class that implements the SAXXMLReader interface.
- Creates a ContentHandler by instantiating the MyContent class.
- Registers the ContentHandler with the parser.
Complete Code for the Main Program

The following is the code for the main program.

```
#include "stdafx.h"
#include "MyContent.h"
#include "SAXErrorHandlerImpl.h"

int main(int argc, char* argv[]) {
    if (argc<2) {
        printf("\nTry something like\n\ntestSax_
file://drive:/path/file.xml\nGood luck!\n\n");
        return 0; // Need URL to read
    }
    CoInitialize(NULL);
    ISAXXMLReader* pRdr = NULL;

    HRESULT hr = CoCreateInstance(
        __uuidof(SAXXMLReader),
        NULL,
        CLSCTX_ALL,
        __uuidof(ISAXXMLReader),
        (void**)&pRdr);

    if(!FAILED(hr)) {
        MyContent * pMc = new MyContent();
        hr = pRdr->putContentHandler(pMc);

        // No sense to do so in this example, just an illustration how set other handlers
        //===============================================================================
        SAXErrorHandlerImpl * pEc = new SAXErrorHandlerImpl();
        hr = pRdr->putErrorHandler(pEc);
        // SAXDTDHandlerImpl * pDc = new SAXDTDHandlerImpl();
        // hr = pRdr->putDTDHandler(pDc);

        static wchar_t URL[1000];
        mbstowcs( URL, argv[1], 999 );
        wprintf(L"\nParsing document: %s\n", URL);

        hr = pRdr->parseURL(URL);
        printf("\nParse result code: %08x\n", hr);
    }
}
```
pRdr->Release();
}
else
{
    printf("\nError %08X\n\n", hr);
}

CoUninitialize();
return 0;
}

To run the JumpStart application

1. Download the JumpStart application, available from the MSDN Code Center, to your computer.
2. From the command prompt, type the following:

    pathname\debug\cppsaxsample.exe test.xml.

where pathname is the name of the folder to which you downloaded the JumpStart application files. The test.xml file is a test file provided with the download. The parsed test.xml file should appear in the command prompt window.

You can use this brief introduction to the Simple API for XML (SAX2) as a starting point for writing your own applications. For more information about the Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office implementation of the SAX2 interfaces, see the SAX2 Reference.
Resource Listings for the SAX2 JumpStart Application

The following XML documents are used for testing the SAX2 Application.

**test.xml**

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<root foo="bar.com">
<PARTS>
  <PART foo="bar.com" foo2="bar.com">
    <PARTNO>12345</PARTNO>
    <DESCRIPTION>VIP - Very Important Part</DESCRIPTION>
  </PART>
  <PART>
    <PARTNO>5678</PARTNO>
    <DESCRIPTION>LIP - Less Important Part</DESCRIPTION>
  </PART>
</PARTS>
</root>
```

**test1.xml**

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<root foo="bar.com">
<PARTS>
  <PART foo="bar.com" foo2="bar.com">
    <PARTNO>12345</PARTNO>
    <DESCRIPTION>VIP - Very Important Part</DESCRIPTION>
  </PART>
  <PART>
    <PARTNO>5678</PARTNO>
    <DESCRIPTION>LIP - Less Important Part</DESCRIPTION>
  </PART>
</PARTS>
</root>
```

**test2.xml**

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<root foo="bar.com">
<PARTS>
```
<PART foo="bar.com" foo2="bar.com">
  <PARTNO>12345</PARTNO>
  <DESCRIPTION>VIP - Very Important Part</DESCRIPTION>
</PART>

<PART>
  <PARTNO>5678</PARTNO>
  <DESCRIPTION>LIP - Less Important Part</DESCRIPTION>
  <qu>...</qu>
</PART>
</PARTS>
</root>
Creating C++ Applications Using SaxAppWizard

Programming C++ applications that use SAX can require some extra work. The SAX Win32 AppWizard, available for download from Microsoft® Product Support Services, facilitates programming SAX applications in C++. Included with the download is a sample program created using this wizard.

**Note**  As with other sample code provided as an aid to developers, there is no support plan for the SAX Win32 AppWizard.

**Other Resources**

[SAMPLE: SAX Win32 AppWizard Creates a Visual C++ SAX Application](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Concepts

The Simple API for XML (SAX) is an events-based parser that provides an alternative to the Document Object Model (DOM). This section introduces some of the fundamental concepts pertinent to SAX and events-based parsing.

- How Events-Based Parsing Works
- Receiving Events with SAX
- Choosing Between SAX and DOM

See Also

Using XSLT with SAX
How Events-Based Parsing Works

To appreciate the benefits of the Microsoft COM implementation of the Simple API for XML (SAX), it helps to understand how you can use SAX2 to process XML documents. The SAX2 parser reads an XML document and generates events based on specific symbols that it encounters in the document. The SAX2 parser does not create a tree structure in memory for the document, although you can programmatically instruct it to do so. Rather, the SAX2 parser processes a document's contents sequentially and generates events as it reads through the document.

When you write an application based on events-based programming, you create functions to respond to user-generated events, such as the `onClick` event. Writing an application for an events-based parser is similar. However, in the case of SAX, the parser, not a user, generates the events.

**Example**

Consider the following simplified document.

```xml
<?xml version="1.0"?>
<parts>
 <part>TurboWidget</part>
</parts>
```

As the SAX2 parser processes this document, it generates a sequence of events such as the following.

```plaintext
StartDocument( )
StartElement( "parts" )
StartElement( "part" )
Characters( "TurboWidget" )
EndElement( "part" )
EndElement( "parts" )
EndDocument( )
```

Think of SAX2 as a push parser. SAX2 generates the events and you implement the event handlers that contain methods to process the events. For more information about event handlers and how to implement them, see JumpStart for...
Creating a SAX2 Application.
Receiving Events with SAX

The following are the building blocks of a typical SAX application.

**SAXXMLReader**

SAXXMLReader implements the ISAXXMLReader interface. SAXXMLReader consumes the XML and throws events to the handlers that you set for the reader. First you create an instance of SAXXMLReader, and then set a ContentHandler and an ErrorHandler for receiving events from the reader.

**IVBSAXContentHandler**

The ContentHandlerImpl.cls class implements the IVBSAXContentHandler interface. IVBSAXContentHandler catches events thrown by the reader. The main events are: startDocument, endDocument, startElement, endElement, and characters.

**IVBSAXErrorHandler**

The ContentHandlerImpl.cls class implements the IVBSAXErrorHandler interface. IVBSAXErrorHandler catches any errors raised during the read process. At this time, MSXML supports only the fatalError method.

**Application**

The application consists of code that typically creates an instance of the SAXXMLReader, sets the content handler and error handler for the reader, and then calls the parse or parseURL method of the SAXXMLReader to process an XML file.

See Also

JumpStart for Creating a SAX2 Application with Visual Basic | JumpStart for Creating a SAX2 Application with C++ | Create a Simple Filter | IVBSAXXMLReader Interface | IMXWriter Interface | IVBSAXContentHandler Interface | IVBSAXDeclHandler Interface | IVBSAXDTDHandler Interface | IVBSAXErrorHandler Interface | IVBSAXLexicalHandler Interface | output
Property
Choosing Between SAX and DOM

The following topics discuss the benefits and tradeoffs of using SAX or DOM to process XML documents.

- [When Should I Use SAX?](#)
- [When Should I Use DOM?](#)
When Should I Use SAX?

As an events-based parser that processes documents in a serial manner, the Simple API for XML (SAX) presents an excellent alternative to the Document Object Model (DOM).
When your documents are large

Perhaps the biggest advantage of SAX is that it requires significantly less memory to process an XML document than the DOM. With SAX, memory consumption does not increase with the size of the file. For example, a 100 kilobyte (KB) document can occupy up to 1 megabyte (MB) of memory using the DOM; the same document requires significantly less memory when using SAX. If you must process large documents, SAX is the better alternative, particularly if you do not need to change the contents of the document.
When you need to abort parsing

Because SAX allows you to abort processing at any time, you can use it to create applications that fetch particular data. For example, you can create an application that searches for a part in inventory. When the application finds the part, it returns the part number and availability, and then stops processing.
When you want to retrieve small amounts of information

For many XML-based solutions, it is not necessary to read the entire document to achieve the desired results. For example, if you want to scan data for relevant news about a particular stock, it's inefficient to read the unnecessary data into memory. With SAX, your application can scan the data for news related only to the stock symbols you indicate, and then create a slimmed-down document structure to pass along to a news service. Scanning only a small percentage of the document results in a significant savings of system resources.
When you want to create a new document structure

In some cases, you might want to use SAX to create a data structure using only high-level objects, such as stock symbols and news, and then combine the data from this XML file with other news sources. Rather than build a DOM structure with low-level elements, attributes, and processing instructions, you can build the document structure more efficiently and quickly using SAX.
When you cannot afford the DOM overhead

For large documents and for large numbers of documents, SAX provides a more efficient method for parsing XML data. For example, consider a remote procedure call (RPC) that returns 10 MB of data to a middle-tier server to be passed to a client. Using SAX, the data can be processed using a small input buffer, a small work buffer, and a small output buffer. Using the DOM, the data structure is constructed in memory, requiring a 10 MB work buffer and at least a 10 MB output buffer for the formatted XML data.
When Should I Use DOM?

The following are some suggestions for when to use DOM.
When you need random access to document data

If random access to information is crucial, it is better to use the DOM to create a tree structure for the data in memory. When you use SAX, the document is not in memory. Therefore, you must handle data in the order in which it is processed. SAX can be difficult to use when the document contains many internal cross-references, such as ID and IDREF attributes. DOM stores the document in memory and allows you to access its parts without reading the entire document.
When you want to implement complex searches

It is your responsibility to maintain data structures holding context information that you must retain, such as the attributes of the current element's ancestor.
No SAX implementation in current browsers

SAX support is not built into Microsoft® Internet Explorer.
When you need to perform XSLT transformations

The DOM works better for XSL Transformations (XSLT) where the source XML document is transformed based on the XSLT template applied. For example, to create multiple views of the same data, you must transform it using one of two style sheets. In order for this transformation to take place, you must also create two instances of the DOM. One stores the XML source; the other stores the transformed content.
When you need complex XPath filtering

Use the DOM if you must perform complex XML Path Language (XPath) filtering and retain complex data structures that hold context information. The tree structure of the DOM retains context information automatically. With SAX, you must retain the context information yourself.
When you want to modify and save XML

The DOM allows you to create or modify a document in memory, as well as read a document from an XML source file. SAX is designed for reading, not writing, XML documents. The DOM is the better choice for modifying an XML document and saving the changed document to memory.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
How Do I?

The following tutorials demonstrate common SAX tasks.

- Create a Simple Filter
- Validate Documents Using SAX
- Create a DOMDocument Object from SAX Events
- Convert DOM to SAX
- Use MXXMLWriter
- Extract Data From a Large Document

See Also

Using XSLT with SAX
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Create a Simple Filter

The simple filter example demonstrates how SAX can be used to filter content to create a subset of the original document being parsed. In this example, the application parses `books.xml` and produces a document fragment that contains only `<title>` elements and characters. The application is kept simple to demonstrate the fundamental principles involved in transforming documents with SAX, including the following:

- How to detect the presence of an element in an XML document and how to write the element's content to the application's output.
- The essential SAX methods you must use to conditionally process content.
- How to set a flag to indicate across events that an element is within scope.
- How to create a document fragment.

This topic is divided into the following sections:

- Overview of the Simple Filter Application
- Application Form (Simple Filter)
- How Filter Criteria is Set
- ContentHandlerImpl Class
- Essential Handler Methods
- Run the Application

See Also

Sample XML File (books.xml)
Overview of the Simple Filter Application

The simple filter application contains the building blocks of almost all SAX applications, including:

**SAXXMLReader**
CoClass that implements the ISAXXMLReader interface. SAXXMLReader consumes the XML and throws events to the handlers that you set for the reader. This example creates an instance of SAXXMLReader, and then sets a ContentHandler and an ErrorHandler for receiving events from the reader.

**IVBSAXContentHandler**
Catches events thrown by the reader. For this application, the ContentHandlerImpl class implements the IVBSAXContentHandler interface. This application highlights the main events received by the ContentHandler, including startDocument, endDocument, startElement, end, and characters.

**IVBSAXErrorHandler**
Catches any errors raised during the read process. For this application, the ContentHandlerImpl class also implements the IVBSAXErrorHandler interface. At this time, MSXML supports the fatalError method only.

This application consists of two main parts.

**Application form**
Provides the user interface for the application. This consists of three text boxes: one for entering a file name; one for entering an element name; and the other for displaying results. The form also has Parse and Exit buttons. Pressing Parse creates an instance of the SAXXMLReader; creates an instance of the class implementing the IVBSAXContentHandler and IVBSAXErrorHandler interfaces; and then sets the contentHandler and errorHandler properties of the reader to point to the class. Pressing Parse also sets the filter criteria in the content handler, sets the input for the reader, and starts the parse operation.

**ContentHandlerImpl**
A class that implements the IVBSAXContentHandler and
IVBSAXErrorHandler interfaces. When parsing begins, the reader throws a series of events that are received by an instance of the ContentHandlerImpl class. For example, for each element in a document, the reader throws the startElement, characters, and endElement events. To conditionally output content passed by these events, you add code to the methods in the ContentHandlerImpl class. The ContentHandlerImpl class also implements the IVBSAXErrorHandler interface, which catches any error events thrown by the reader. At this time, the reader only invokes the fatalError method.

The simple filter application uses the sample XML file, books.xml.

See Also

Application Form | IMXWriter Interface | MXXMXMLWriter CoClass | ISAXContentHandler Interface | ISAXErrorHandler Interface | Sample XML File (books.xml)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Application Form (Simple Filter)

To get started, create a new project and then reference Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office.

To create a new project

- Open Microsoft® Visual Basic® 6.0, and in the New Project dialog box, double-click Standard EXE.

To create a reference to MSXML 5.0

1. On the Project menu, click References.
2. In the Available References list, select Microsoft XML,v5.0, and then click OK.
Add Controls to the Form

Now you must build the user interface for the simple filter application. Add the following controls to the form.

- A text box, Text1, with the Multiline property set to True. This text box should be large enough to display the results of the parse process.
- A text box, Text2, with "books.xml", the name of the file that the application parses and converts, specified as the Text property.
- A text box, Text3. This text box is used to specify the element name that serves as the filter criteria.
- A command button, Command1, with the caption "Parse". This button starts the parse process.
- A command button, Command2, with the caption "Exit". This button closes the application.

The form should look like the following.
Add Code to the Form

The code for the form:

- Creates instances of MXXMLWriter and SAXAttributes objects.
- Creates an instance of the SAXXMLReader.
- Creates an instance of the ContentHandlerImpl class implementing the IVBSAXContentHandler and IVBSAXErrorHandler interfaces.
- Sets the errorHandler and the contentHandler properties for the SAXXMLReader to point to the ContentHandlerImpl class.
- Sets the instance of MXXMLWriter to variables in the ContentHandlerImpl class that hold references to the IVBSAXContentHandler and IVBSAXErrorHandler interfaces. This enables the MXXMLWriter object to receive ContentHandler and ErrorHandler events.
- Calls the SetFilterCriteria method in the ContentHandlerImpl class and sets the filter criteria.
- Starts the parsing operation.
Complete Code for the Form

'Create a writer and an attribute helper.
Public oWriter As New MXXMLWriter50
Public atrs As New SAXAttributes50

Private Sub Command1_Click()
'Create the reader.
Dim rdr As New SAXXMLReader50
'Create the content handler.
Dim cnth As New ContentHandlerImpl
'Set the content handler for the reader.
Set rdr.contentHandler = cnth
'Set the error handler for the reader.
Set rdr.errorHandler = cnth
'Set the writer for the content handler.
Set cnth.oContentHandler = oWriter
'Set the error handler for the writer.
Set cnth.oErrorHandler = oWriter

'Configure output for the writer.
oWriter.indent = True
oWriter.standalone = True
oWriter.output = ""
oWriter.omitXMLDeclaration = True

'Set the filter criteria in the content handler.
cnth.SetFilterCriteria (Text3.Text)

On Error GoTo HandleError

'Parse the document.
rdr.parseURL (Text2.Text)

Text1.Text = oWriter.output

Exit Sub

HandleError:
    If Not cnth.errorHappen Then
    End If
End Sub
Private Sub Command2_Click()
    Unload Me
End Sub
See Also

How Filter Criteria is Set | IMXWriter Interface | MXXMLWriter CoClass | ISAXContentHandler Interface | ISAXErrorHandler Interface
How Filter Criteria is Set

One of the benefits of this sample application is that the code can return an element and its contents from any XML document. To achieve this, the Command1_Click() procedure contains a statement that calls the SetFilterCriteria method in the ContentHandlerImpl class and passes it the value of the Text3 text box, as shown in the following code.

```vbnet
cnth.SetFilterCriteria (Text3.Text)
```

The following shows the SetFilterCriteria method in the ContentHandlerImpl class. The FilterCriteria variable is declared in the general section of the ContentHandlerImpl class.

```vbnet
Public Sub SetFilterCriteria(elementname)
    FilterCriteria = elementname
End Sub
```

The value of the FilterCriteria variable is then tested in the startElement method in the ContentHandlerImpl class to determine if the specified element is within scope. The code to test FilterCriteria in the ContentHandler methods, along with a full code listing for the ContentHandlerImpl class, is included in the following topic.

See Also

ContentHandlerImpl Class
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
ContentHandlerImpl Class

The ContentHandlerImpl class:

- Implements the IVBSAXContentHandler and IVBSAXErrorHandler interfaces. The methods of these interfaces receive events thrown by the reader. You must add all methods for the implemented interfaces to the ContentHandlerImpl class.
- Declares public variables for IVBSAXContentHandler and IVBSAXErrorHandler interfaces. The oContentHandler and oErrorHandler variables are set as properties of the ContentHandler object in the form and enable the writer to receive ContentHandler and ErrorHandler events.
- Declares a FilterCriteria variable that holds the element name that serves as the criteria for the filter. Also declares a FilterTrue variable, a Boolean flag used to indicate whether the specified element is within scope.
- Declares an errorHappen variable, a Boolean flag that indicates whether the ErrorHandler has thrown a fatal error message. This prevents the ErrorHandler procedure in the form from overwriting the error message thrown by the error handler.
- Contains If statements and method calls that push content to the result fragment when the specified element is within scope.

To create the class

1. On the Project menu, click Add Class Module.
2. In the Add Class Module dialog box, double-click Class Module.
4. In the Properties Window, for the Name property, type "ContentHandlerImpl".

To implement an interface

1. In the code window, type "Implements" and the name of the interface, for example:

   Implements IVBSAXContentHandler
   Implements IVBSAXErrorHandler
2. In the left-hand drop-down in the code window, select the interface from the list.
3. In the right-hand drop-down in the code window, you can implement the methods for the interface by selecting them from the list.

   **Note**  You must implement all methods for the implemented interfaces.
**Complete Code for ContentHandlerImpl**

Add the following code to the class.

**Note** If you already added the `Implements` statements, you can simply copy the following code before the first `Implements` statement.

The essential methods used for the filter are discussed in the following topic, [Essential Handler Methods](#).

```vbnet
Option Explicit
Implements IVBSAXContentHandler
Implements IVBSAXErrorHandler
'Declare a variable for setting the writer to the content handler.
Public oContentHandler As IVBSAXContentHandler
'Declare a variable for setting the writer to the error handler.
Public oErrorHandler As IVBSAXErrorHandler
'Flag to indicate if the error handler has thrown a fatal error.
Public errorHappen As Boolean
'Flag to indicate if the element is in scope.
Dim FilterTrue As Boolean
'Declare a string to hold the element name.
Dim FilterCriteria As String

Private Sub IVBSAXContentHandler_characters(strChars As String)
    If FilterTrue Then
        oContentHandler.characters strChars
    End If
End Sub

Private Property Set IVBSAXContentHandler_documentLocator(ByVal RHS As MSXML2.IVBSAXLocator)
    Initialize
End Property

Private Sub IVBSAXContentHandler_endDocument()
End Sub

Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As String, strLocalName As String, strQName As String)
    If FilterTrue Then
        oContentHandler.endElement strNamespaceURI, strLocalName, _
        strQName
    End If
End Sub
```
Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As String)
End Sub

Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars As String)
End Sub

Private Sub IVBSAXContentHandler_processingInstruction(strTarget As String, _
strData As String)
End Sub

Private Sub IVBSAXContentHandler_skippedEntity(strName As String)
End Sub

Private Sub IVBSAXContentHandler_startDocument()
End Sub

Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As String, _
strLocalName As String, strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttribute)
    If strLocalName = FilterCriteria Then
        FilterTrue = True
    End If
    If FilterTrue Then
        oContentHandler.startElement strNamespaceURI, strLocalName, strQName, oAttributes
    End If
End Sub

Private Sub IVBSAXContentHandler_startPrefixMapping(strPrefix As String, strURI As String)
End Sub

Private Sub Initialize()
    errorHappen = False
    FilterTrue = False
Private Sub IVBSAXErrorHandler_error(ByVal oLocator As MSXML2.IVBSAXLocator, _
    strErrorMessage As String, _
    ByVal nErrorCode As Long)
End Sub

Private Sub IVBSAXErrorHandler_fatalError(ByVal oLocator As MSXML2.IVBSAXLocator, _
    strErrorMessage As String, _
    ByVal nErrorCode As Long)
    Form1.Text1.Text = strErrorMessage & nErrorCode
    errorHappen = True
End Sub

Private Sub IVBSAXErrorHandler_ignorableWarning(ByVal oLocator As _
    MSXML2.IVBSAXLocator, _
    strErrorMessage As String, _
    ByVal nErrorCode As Long)
End Sub

Public Sub SetFilterCriteria(elementname)
    FilterCriteria = elementname
End Sub

See Also

Essential Handler Methods | IMXWriter Interface | MXXMLWriter CoClass | ISAXContentHandler Interface | ISAXErrorHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Essential Handler Methods

Although the Simple API for XML (SAX) can catch many events as it parses through a document, this application focuses on the most frequently used methods. In this example, the `ContentHandlerImpl` class implements all methods for `IVBSAXContentHandler` and `IVBSAXErrorHandler` interfaces, but code is added only to those methods that you want the application to be aware of, including `documentLocator`, `startElement`, `endElement`, `characters`, and `endDocument` for the `ContentHandler`, and `fatalError` for the `ErrorHandler`. As the reader passes through the document, it reports events in document order: `documentLocator`, followed by `startDocument`, `startElement`, `characters`, `endElement`, and `endDocument`. 
set IVBSAXContentHandler_documentLocator method

The set IVBSAXContentHandler_documentLocator method is the first event thrown by the SAX reader when is passes through a document. This method receives a pointer to the IVBSAXLocator interface, which provides methods for returning the column number, line number, public ID, or system ID for a current document. Because this is the first method thrown, it is a good place to put initialization code.

In the following snippet, the Initialize procedure sets the TitleTrue flag, the flag that indicates if the element is within scope, to False. This procedure also sets the errorHappen flag to False. This flag is used to indicate that a fatal error has been thrown by the ErrorHandler and prevents the HandleError procedure in the form from overwriting the error message thrown by the ErrorHandler.

Private Property Set IVBSAXContentHandler_documentLocator(ByVal RHS As MSXML2.IVBSAXLoc
Initialize
End Property

Private Sub Initialize()
    errorHappen = False
    TitleTrue = False
End Sub
**startElement Method**

The `startElement` event is thrown by the reader each time it encounters the "<" character of a new element. The `startElement` method receives the namespace URI for the element, the local name, and the qualified name of the element. Attributes for the element are sent as a separate object containing the type, URI, local name, and value of each attribute. These values can be accessed from the attributes object within the `startElement` method.

In this method, the first if clause sets the `FilterTrue` flag if the element name passed by the `startElement` event matches the element name specified as the filter criteria. When `FilterTrue` is True, the second if clause invokes the `startElement` method from the `oContentHandler` object, which holds a reference pointing to `oWriter`, the `MXXMLWriter` object created in the form. This pushes the data passed by the `startElement` method to the `oWriter` object.

```vbnet
Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As String, strLocalName As String, strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttribute)

    If strLocalName = FilterCriteria Then
        FilterTrue = True
    End If

    If FilterTrue Then
        oContentHandler.startElement strNamespaceURI, strLocalName, strQName, oAttributes
    End If

End Sub
```
characters Method

The characters method receives the character content between the opening and closing tags of an element. For this example, the characters method contains code that pushes character data to the writer only when the FilterTrue flag is set to True.

Private Sub IVBSAXContentHandler_characters(strChars As String)
    If FilterTrue Then
        oContentHandler.characters strChars
    End If
End Sub
**endElement Method**

SAX often throws events in pairs, modeling the structure of the XML document. For example, a startDocument event has a corresponding endDocument event and a startElement event has a corresponding endElement event. For the endElement method, the first if clause pushes the content received by the endElement method to the writer when the FilterTrue flag is True. The second if clause sets the FilterCriteria flag to False when the value passed by the endElement method matches the filter criteria. This indicates that the element is no longer in scope.

```vbnet
Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As String,
                                         strLocalName As String,
                                         strQName As String)
    If FilterTrue Then
        oContentHandler.endElement strNamespaceURI, strLocalName, _
                                    strQName
    End If
    If strLocalName = FilterCriteria Then
        FilterTrue = False
    End If
End Sub
```
fatalError Method

Finally, code is added to the fatalError method to capture any errors thrown by the reader during parsing. If a fatal error is received, the error message is displayed in the Text1 box. The errorHappen flag is set to True, preventing the HandlerError routine in the form from overwriting the fatalError error message.

Private Sub IVBSAXErrorHandler_fatalError(ByVal oLocator As MSXML2.IVBSAXLocator, _
    ByVal strErrorMessage As String, _
    ByVal nErrorCode As Long) _
    Form1.Text1.Text = strErrorMessage & nErrorCode
    errorHappen = True
End Sub

See Also

Run the Application | IMXWriter Interface | MXXMLWriter CoClass | ISAXContentHandler Interface | ISAXErrorHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Run the Application (Simple Filter)

The following instructions are based on the books.xml file, but you can also use other XML documents in this application.

To run the application

1. Save the project and then copy books.xml to the folder where you saved the project.
2. From the Visual Basic toolbar, click Start.
3. In the Filename box, enter the path for the books.xml file.
4. In the Element name box, type the name of the element that serves as the filter criteria. For this example, type "title".
5. On the form, click Parse.

The Text2 text box displays the following results.

<title>XML Developer's Guide</title>
<title>Midnight Rain</title>
<title>Maeve Ascendant</title>
<title>Oberon's Legacy</title>
<title>The Sundered Grail</title>
<title>Lover Birds</title>
<title>Splish Splash</title>
<title>Creepy Crawlies</title>
<title>Paradox Lost</title>
<title>Microsoft .NET: The Programming Bible</title>
<title>MSXML3: A Comprehensive Guide</title>
<title>Visual Studio 7: A Comprehensive Guide</title>

See Also

Sample XML File (books.xml)
Validate Documents Using SAX

This example demonstrates how to create a simple SAX validator application in Microsoft® Visual Basic®. The application allows the user to select an XML file to be validated against an XSD schema file.

In this example, the SAX reader parses the sample XML file (books.xml), and writes validation results as output to a textbox control on the main form for the application.

The application is kept simple to demonstrate the fundamental principles involved. These principles include the following:

- Validating from within SAX applications, which differs from validating XML in DOM-based applications.
- Validating XML file input to a user-selectable XSD schema file.
- Using SAX error handler and SAX locator information to improve error reporting for your SAX application.

This topic is divided into the following sections.

- Overview of the SAX Validator Application
- Application Form (SAX Validator)
- Sample XSD Schema File (SAX Validator)
- MyValidator Class (SAX Validator)
- Run the Application (SAX Validator)
- How the SAX Validator Application Works

See Also

Sample XML File (books.xml)

Note Currently, SAX-based validation only supports external XSD schemas. SAX-based validation using Document Type Definition (DTD) files is not supported.
Overview of the SAX Validator Application

The SAX validator application uses the following SAX interfaces.

**SAXXMLReader**
A CoClass that implements the ISAXXMLReader interface. The SAX reader parses the XML and throws events to the handlers that you set for the reader. This application creates an instance of SAXXMLReader, enables SAX validation by setting the schema-validation feature, and sets the schema and schema-declaration-handler properties on the reader to fully enable support for SAX-based validation.

**XMLSchemaCache**
For this application, you declare the XMLSchemaCache object and add the schema file (in this example, books.xsd) to it as a member. The XMLSchemaCache object is then assigned to the SAX reader to be used to validate the sample XML file (books.xml).

**IVBSAXContentHandler**
Provides content handling features. For this application, the MyValidator class implements this interface to receive content events that are thrown by the SAX reader. For this application, the content handler uses only a single event, the endDocument method, to acknowledge a successful validation of the file.

**IVBSAXErrorHandler**
Provides error handling features. For this application, the MyValidator class implements this interface to receive errors that are thrown by the SAX reader.

**IVBSAXLocator**
Provides location information. For this application, the MyValidator class implements this interface and uses its lineNumber and columnNumber properties to return the position within the XML file as part of error reporting if a validation error occurs.

This application consists of the following two main parts.

Application form
A Microsoft® Visual Basic® 6.0 form that provides the user interface for the application. This form consists of three text boxes with corresponding labels. Two of the text boxes are used to enter file and path information for the XML and XSD files that are used as input. The third text box displays the output of the application.

The form also has two command buttons: Validate and Exit. Pressing Validate starts application processing. Pressing Exit unloads the application form, effectively exiting the application.

**MyValidator**

A Visual Basic class module that implements the IVBSAXContentHandler, IVBSAXErrorHandler, and IVBSAXLocator interfaces. When SAX parsing begins, the SAX reader throws a series of events that are received by an instance of this class.

**See Also**

[Validate Documents Using SAX] | [Overview of the SAX Validator Application] | [Application Form (SAX Validator)] | [Sample XSD Schema File (SAX Validator)] | [MyValidator Class (SAX Validator)] | [Run the Application (SAX Validator)] | [How the SAX Validator Application Works]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Sample XSD Schema File (SAX Validator)

To validate the sample XML file (*books.xml*), you must create an XSD schema file.

To create the sample XSD schema file

1. Open Notepad.
2. Select the code for *books.xsd* (below). Then copy it and paste it in Notepad.
3. From Notepad, save the file as *books.xsd* to the same folder where you are creating the SAX Validator application.
Complete Code for books.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="author" type="xsd:string"/>
              <xsd:element name="title" type="xsd:string"/>
              <xsd:element name="genre" type="xsd:string"/>
              <xsd:element name="price" type="xsd:float"/>
              <xsd:element name="publish_date" type="xsd:date"/>
              <xsd:element name="description" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

Note  For the SAX validator application to work properly, you must copy and save the sample XML file (books.xml) to the same folder.

See Also

Validate Documents Using SAX | Overview of the SAX Validator Application | Application Form (SAX Validator) | MyValidator Class (SAX Validator) | Run the Application (SAX Validator) | How the SAX Validator Application Works
Application Form (SAX Validator)

To get started, create a new project and then reference Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office.

To create a new project

- Open Microsoft® Visual Basic® 6.0. In the New Project dialog box, double-click Standard EXE.

To create a reference to MSXML 5.0

1. On the Project menu, click References.
2. In the Available References list, select Microsoft XML,v5.0, and then click OK.
Add Controls to the Form

Now build the user interface for the SAX validator application. Add the following controls to the form.

- Three text boxes: one for entering the XML file input to be validated; one for entering the XSD file to use to validate the input file; and one for displaying search results.
- Labels for each of the three text boxes
- Two command buttons.

For your application form to work with the sample code provided in the next section, you must set the properties in the following table.

<table>
<thead>
<tr>
<th>Control</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Name</td>
<td>frmMain</td>
</tr>
<tr>
<td></td>
<td>Caption</td>
<td>&quot;SAX Validator&quot;</td>
</tr>
<tr>
<td>Text1</td>
<td>Name</td>
<td>txtInputFile</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>&quot;\books.xml&quot;</td>
</tr>
<tr>
<td>Text2</td>
<td>Name</td>
<td>txtSchemaFile</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>&quot;\books.xsd&quot;</td>
</tr>
<tr>
<td>Text3</td>
<td>MultiLine</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Command1</td>
<td>Name</td>
<td>cmdValidate</td>
</tr>
<tr>
<td></td>
<td>Caption</td>
<td>&quot;Validate&quot;</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>cmdExit</td>
</tr>
</tbody>
</table>
After you modify the property settings, resize the controls and arrange them on the form until your user interface looks like the following:
Add Code to the Form

The following shows the complete code for the form. To run the code as you read, select all the text, then copy it and paste it into the form of your own Microsoft® Visual Basic® project.

Option Explicit

Private Sub cmdExit_Click()
    Unload Me
End Sub

Private Sub cmdValidate_Click()
    'Clear the results window.
    txtResults = ""
    'Create a SAX reader.
    Dim oReader As New Msxml2.SAXXMLReader50
    'Create an XML schema cache.
    Dim oSC As New Msxml2.XMLSchemaCache50
    'Create a class module for implementing content handler,
    'the error handler, and the locator interfaces.
    Dim oValidator As New MyValidator
    'Add the schema file to the schema cache.
    oSC.Add "", txtSchemaFile
    'Configure the SAX reader.
    oReader.putFeature "schema-validation", True
    oReader.putProperty "schemas", oSC
    oReader.putProperty "schema-declaration-handler", oValidator
    Set oReader.contentHandler = oValidator
    Set oReader.errorHandler = oValidator
    'Parse and validate the file.
    On Error Resume Next
    oReader.parseURL txtInputFile
    On Error GoTo 0
End Sub

Private Sub Form_Load()
    txtInputFile.Text = App.Path + txtInputFile.Text
End Sub

See Also

Validate Documents Using SAX | Overview of the SAX Validator Application |
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
MyValidator Class (SAX Validator)

To create the class

1. On the Project menu, click Add Class Module.
2. In the Add Class Module dialog box, double-click Class Module.
4. In the Properties Window, for the Name property, type "MyValidator".

To implement an interface

1. In the code window, type "Implements" and the name of the interface, for example:

   Implements IVBSAXContentHandler
   Implements IMXSchemaDeclHandler
   Implements IVBSAXErrorHandler
   Implements IVBSAXLocator

2. In the left-hand drop-down list in the code window, select the interface.
3. In the right-hand drop-down list in the code window, you can implement the methods for the interface by selecting them from the list.

   Note  You must implement all methods for the implemented interfaces.
Complete Code for MyValidator

Add the following code to the class.

Note If you already added the Implements statements, you can simply copy the following code and paste it before the first Implements statement.

Option Explicit
'Implement SAX interfaces.
Implements IVBSAXContentHandler
Implements IMXSchemaDeclHandler
Implements IVBSAXErrorHandler
Implements IVBSAXLocator
'Declare a module-scope variable for setting the locator.
Private oLocator As IVBSAXLocator

Private Sub IMXSXemaDeclHandler_schemaElementDecl(ByVal oSchemaElem
End Sub

Private Sub IVBSAXContentHandler_characters(strChars As String)
End Sub

Private Property Set IVBSAXContentHandler_documentLocator(ByVal RHS
    Set oLocator = RHS
End Property

Private Sub IVBSAXContentHandler_endDocument()
    frmMain.txtResults = frmMain.txtResults + "File is valid."
End Sub

Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As Strin
End Sub

Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As Strin
End Sub

Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars As Str
End Sub

Private Sub IVBSAXContentHandler_processingInstruction(strTarget As Str
End Sub

Private Sub IVBSAXContentHandler_skippedEntity(strName As String)
End Sub

Private Sub IVBSAXContentHandler_startDocument()
End Sub

Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As String, strLocalName As String, strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttributes)
End Sub

Private Sub IVBSAXContentHandler_startPrefixMapping(strPrefix As String, strURI As String)
End Sub

Private Sub IVBSAXErrorHandler_error(ByVal oLocator As MSXML2.IVBSAXLocator, strErrorMessage As String, ByVal nErrorCode As Long)
WriteErrorToResults "Error", strErrorMessage, nErrorCode, oLocator.lineNumber, oLocator.columnNumber
End Sub

Private Sub IVBSAXErrorHandler_fatalError(ByVal oLocator As MSXML2.IVBSAXLocator, strErrorMessage As String, ByVal nErrorCode As Long)
WriteErrorToResults "Fatal error", strErrorMessage, nErrorCode, oLocator.lineNumber, oLocator.columnNumber
End Sub

Private Sub IVBSAXErrorHandler_ignorableWarning(ByVal oLocator As MSXML2.IVBSAXLocator, strErrorMessage As String, ByVal nErrorCode As Long)
WriteErrorToResults "Ignorable warning", strErrorMessage, nErrorCode, oLocator.lineNumber, oLocator.columnNumber
End Sub

Private Property Get IVBSAXLocator_columnNumber() As Long
End Property

Private Property Get IVBSAXLocator_lineNumber() As Long
End Property

Private Property Get IVBSAXLocator_publicId() As String
End Property

Private Property Get IVBSAXLocator_systemId() As String
End Property

Private Function WriteErrorToResults(strLabel As String, ByVal strDescription As String, ByVal ErrCode As Long, ByVal Line As Long, ByVal Column As Long)
frmMain.txtResults = 
    strLabel + ": (" + CStr(ErrCode) + ") " + 
    strDescription & "at " + "line " + _
    Str(Line) + ", column " + _
    Str(Column) + vbCrLf
End Function

See Also

Validate Documents Using SAX | Overview of the SAX Validator Application |
Run the Application (SAX Validator)

The following instructions are based on the books.xml and books.xsd files. However, you can use other XML and XSD documents as the files for this application.

To run the application

1. Save the project and then copy books.xml to the folder where you saved the project.
2. From the Visual Basic toolbar, click Start.
3. In the XML file to validate text box, verify that the path for the books.xml file is valid and correct.
4. In the Use this XSD schema file to validate it text box, verify that the path for the books.xsd file is valid and correct.
5. On the form, click Validate.

If the validation is successful, the Results text box displays the following results.

File is valid.

To verify that validation is working, try opening books.xml and changing the names of elements to names that are not allowed by the books.xsd schema.

For example, if you change the following line in books.xml from this:

    <price>44.95</price>

...to this:

    <cost>44.95</cost>

...you will get a validation error when you next run the SAX Validator application. The following is an example of a validation error.

    Error: (-2147467259) Element content is invalid according to the DTD/Schema.
    Expecting: price.
See Also

Sample XML File (books.xml) | Sample XSD Schema File (SAX Validator) | Validate Documents Using SAX | Overview of the SAX Validator Application | Application Form (SAX Validator) | MyValidator Class (SAX Validator) | How the SAX Validator Application Works
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
How the SAX Validator Application Works

This application is similar to the search and abort application in that it uses the ISAXLocator interface, which is also implemented in that application. In the SAX validator application, the location information is used to help provide exact error information when a validation error occurs during SAX parsing.

Results are reported in one of the following ways, depending on whether the XML file validates to the specified XSD schema.

- If the file is valid, the application reports a simple acknowledgment message ("File is valid."). The SAX content handler provides this message by using code that is added to the endDocument method.
- If the file is not valid, error text is written instead. This text describes the cause of the error. It also provides the location (by line and column number) of the error in the sample file. The SAX error handler provides this information by using code that is added to each of its member methods: error, fatalError, and ignorableWarning. These methods reference a private function, WriteErrorToResults, that is added to the MyValidator class module.

See Also

endDocument Method | ISAXErrorHandler Interface | Validate Documents Using SAX | Overview of the SAX Validator Application | Sample XSD Schema File (SAX Validator) | Application Form (SAX Validator) | MyValidator Class (SAX Validator) | Run the Application (SAX Validator)
Create a DOMDocument Object from SAX Events

With MSXML 5.0, you can generate a DOMDocument (or FreeThreadedDOMDocument) object from SAX events. This feature is useful when you want to create a small subset of XML from a large XML document and then use the DOM to process the results. For example, you might want to parse only small portions of a catalog, such as a <book> node, and store the results in a DOM object. You can then use the DOM to compare content, walk the DOM tree, and modify or update content. In addition, you might want to use SAX to search a large catalog file for a particular book. For performance gains, you can use SAX to locate a single item and then abort parsing after it is located. You can output the results of the search into a DOMDocument object and then apply DOM methods to the object.

This topic provides a simple example that demonstrates how to set a DOM object as the destination output for SAX events. It also provides implementation notes that describe some of the restrictions on the SAX to DOM implementation.

- Overview of the SAX to DOM Example
- Setting a DOMDocument Object as MXXMLWriter Output
- SAX to DOM Implementation Notes
Overview of the SAX to DOM Example

This simple SAX to DOM example uses Microsoft® Visual Basic® code that creates DOMDocument and MXXMLWriter objects, sets the MXXMLWriter object as the ContentHandler, and then sets the output property of the MXXMLWriter object as the DOMDocument object. The ContentHandler method calls cause nodes to be built as the root document instance of the provided document. At the end of the procedure, DOM methods are used to return the name of the author, and to display the name in a message box.

To run the code sample

1. Open Visual Basic 6.0, and in the New Project dialog box, double-click Standard EXE.
2. In the Visual Basic 6.0 Window, from the Project menu, select References.
3. Under Available References, select Microsoft XML, 5.0, and then click OK.
4. In the Toolbox, click the Button control, and then add a button to the Form1 window.
5. Add the following code to the Command1_Click procedure and then click Start on the toolbar.
Complete Code for the SAX to DOM Example

Private Sub Command1_Click()
    Dim xmlDoc As New Msxml2.DOMDocument50
    'If using SAX lexical handler, the following line is required.
    xmlDoc.validateOnParse = False

    Dim nodeList As IXMLDOMNodeList
    Dim wrt As New MXXMLWriter50
    Dim cnth As IVBSAXContentHandler
    'If using SAX lexical handler, the following line is required.
    Dim lexh As IVBSAXLexicalHandler
    Dim atrs As New SAXAttributes50
    Dim objNodeList

    Set cnth = wrt
    'If using SAX lexical handler, the following line is required.
    Set lexh = wrt
    wrt.output = xmlDoc

    'Configures the writer to indent elements.
    wrt.indent = True

    'Starts the document.
    cnth.startDocument

    'Adds the XML declaration.
    cnth.processingInstruction "xml", "version='1.0'"

    'Inserts DOCTYPE delcaration for DTD in DOM output.
    lexh.startDTD "catalog", "", "books.dtd"
    lexh.endDTD
    'You can remove or comment out previous two lines if
    'you are not linking to a DTD.

    'Adds the <catalog> element to the page.
    cnth.startElement "", "", "catalog", atrs

    'Adds the id attribute to the collection witht he "bk0101" value
    atrs.addAttribute "", "", "id", "CDATA", "bk101"
    'Creates the <book id="bk0101"> tag.
    cnth.startElement "", "", "book", atrs
    'Clears the attribute collection.
    atrs.Clear

    'Creates the <author>Gambardella, Matthew</author> string.
    cnth.startElement "", "", "author", atrs
'Creates the <title>XML Developer's Guide</title> string.
<xml>umen.startElement "", "", "title", atrs
xml.characters "XML Developer's Guide"
xml.endElement "", "", "title"

'Creates the <description>An in-depth look at...</description> string.
xml.startElement "", "", "description", atrs
xml.characters "An in-depth look at creating applications with"
xml.endElement "", "", "description"

'Adds closing tags for <book> and <catalog> elements.
xml.endElement "", "", "book"
xml.endElement "", "", "catalog"

'Ends the document.
xml.endDocument

'Displays the author's name in a message box.
Set objNodeList = xmlDoc.getElementsByTagName("author")
MsgBox objNodeList.Item(0).Text
End Sub

See Also

Setting a DOMDocument Object as MXXMLWriter Output | MXXMLWriter CoClass | output Property | DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Setting a DOMDocument Object as MXXMLWriter Output

The following code shows how to set the DOMDocument object as MXXMLWriter output and how to get the DOMDocument output using DOMDocument methods.
Setting Output

The following code shows how to set writer output as a DOMDocument object.

Private Sub Command1_Click()
    'Create the DOMDocument object.
    Dim xmlDoc As New Msxml2.DOMDocument50
    'Create an MXXMLWriter object.
    Dim wrt As New MXXMLWriter50
    'Reference the ContentHandler interface.
    Dim cnth As IVBSAXContentHandler

    'Set the writer as the content handler.
    Set cnth = wrt
    'Set the writer output to the xmlDoc object.
    wrt.output = xmlDoc
Getting Output

You can read the document at any time after the startDocument event is called. However, you cannot modify the document until after the endDocument event is called. The following code shows DOMDocument methods called on the xmlDoc object after the endElement method is called.

```vbs
    cnth.endDocument
    'Displays the author's name in a message box.
    Set objNodeList = xmlDoc.getElementsByTagName("author")
    MsgBox objNodeList.Item(0).Text
```

See Also

[SAX to DOM Implementation Notes] | [MXXMLWriter CoClass] | [output Property]
| [DOMDocument]
SAX to DOM Implementation Notes

You should be aware of the following considerations.
Reading and Modifying the DOMDocument Object

When a DOMDocument object is built from SAX events, the document is locked and cannot be modified until the endDocument method is called. The document can, however, be read anytime after startDocument method is called. The following provides an overview of how the document is locked as it is built.

When the startDocument method is called, a write-lock is imposed on the document; existing data in the document is removed; and the lock is downgraded to a read-lock.

The read-lock is held on the document until one of the following conditions is met:

- The endDocument method is called.
- Any of the methods on the errorHandler interface are called.
- The MXXMLWriter object is released and destructed.
- There is an error (for example, a call to startElement fails).

Any attempt to modify the document while the read-lock is held will raise an error if the method is invoked from the current thread or block if invoked from a different thread.

Note  For MSXML 5.0, the new parser will always be used for a DOMDocument object connected to receive the output of SAX events. In this scenario, you should be aware that the new parser does not support DTD validation. For example, if you do not first set the validateOnParse property to False on the DOMDocument object, and then try to invoke the SAX lexical handler (ISAXLexicalHandler) by calling startDTD, it will raise an error.
**Supported Methods**

When a DOMDocument object is set as writer output, only the following methods are supported.

- `get/set output`
- `get/set indent`
- `flush()`
Allowed Sequence of Handler Callbacks

The following sequence of handler callbacks can be invoked.

```xml
startDocument
(comment | processingInstruction)*
(startDTD DTD_CONTENT endDTD)?
(comment | processingInstruction)*
(startElement ELEMENT_CONTENT endElement)
(comment | processingInstructions)*
endDocument
```

```xml
DTD_CONTENT :=
elementDecl
| attributeDecl
| internalEntityDecl
| externalEntityDecl
| unparsedEntity
| notationDecl
| startEntity ELEMENT_CONTENT* endEntity
| processingInstruction
| comment

ELEMENT_CONTENT :=
Characters
| ignorableWhitespace
| startCDATA characters* endCDATA
| startElement ELEMENT_CONTENT* endElement
| skippedEntity
| startEntity ELEMENT_CONTENT* endEntity
| processingInstruction
| comment
```

When building a DOMDocument object, a startDocument event and its corresponding endDocument event must be called. If the callback sequence is not valid (for example, endElement is called without closing an open CDATA section), errorHandler aborts the parse and builds a parseError object from the provided information.
indent Property

When the indent property is set, characters events are scanned to see if the event is purely white space. In this case, it is treated just like a white space event from the old parser. What this means is that when the only event to separate other (non-characters) events is a characters with only white space text, then there is no text node built for the white space, but the white space is remembered (as hints).

For example:

```xml
startElement("","","a",attrs);
characters(" ");
endElement("","","a");
```

normally builds a DOM that looks like this:

```
Element:nodeName="a"
    |  
   TextNode " 
```

which would be output like this:
```xml
<a>  </a>
```

If the indent property is set, then Element node does not have a text child and the output looks like this:
```
<a>  </a>
```

**Note**  The indenting is due to special hints that are stored internally. They are not exposed in any manner, except as indenting when saving.

**See Also**

[MXXMXMLWriter CoClass](#) | [output Property](#) | [DOMDocument](#)
Convert DOM to SAX

With MSXML 5.0, you can use SAX and DOM interchangeably. For example, a process might pass your application a DOMDocument or an IXMLDOMNode object. However, you might want the flexibility of using the SAX writer to produce the resulting XML for your application. In this situation, your application would require SAX input, not DOM input. Therefore, you might choose to perform DOM-to-SAX conversion.

This topic provides an example that shows how you do this by parsing a DOMDocument object with SAX events.

- Overview of the DOM to SAX Example
- Passing a DOMDocument Object as a Parse() Parameter

Note Besides the DOMDocument object, SAXXMLReader can parse objects based on IXMLDOMNode, or any interface that inherits from it.
Overview of the DOM to SAX Example

The DOM to SAX example is Microsoft® Visual Basic® code that executes when you click a command button on the application form. The code in this example performs the following tasks:

- Creates DOMDocument, SAXXMLReader, and MXXMLWriter objects.
- Sets the XML writer (wrt) object for the value of the ContentHandler property of the SAX reader (rdr) object.
- Passes the DOM document (xmlDoc) object to the parse method of the SAX reader.
- Writes the results of the SAX events to the output property of the SAX writer.
- Displays the SAX writer output using a Visual Basic text box control on the application form.

To run the code sample

1. Copy the sample XML file (books.xml) and paste it into a text file. Save the file as books.xml.
2. Create a Standard EXE project in Visual Basic. Save the empty project as dom2sax.vbp to the same directory where you saved books.xml. Name the form file using the default name of Form1.frm.
3. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, and then check the box for Microsoft XML, v5.0.
4. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named Text1. In the properties for Text1, set the ScrollBars property to a value of "2 - Vertical" and set the MultiLine property to True.
5. From the View menu, click Code.
6. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor, replacing any code already there.
7. From the Run menu, click Start.
8. Verify that the output is the same (pass-through) as the contents of the original books.xml. To view the contents of the output more easily, resize
the application form as needed.
Complete Code for the DOM to SAX Example
Private Function HandleErrors()
MsgBox Err.Description
End Function
Private Sub Form_Resize()
'Resize text box to size of form.
Text1.Width = Form1.Width - 350
Text1.Height = Form1.Height - 750
End Sub
Private Sub Form_Load()
'Resize text box to size of form.
Text1 = "<?xml version=""1.0""?>" & vbCrLf
Text1.Top = 100
Text1.Left = 100
Text1.Width = Form1.Width - 350
Text1.Height = Form1.Height – 750
'Create a DOMDocument object.
Dim xmlDoc As New DOMDocument50
'Create the SAX reader.
Dim rdr As New SAXXMLReader50
'Create the XML writer.
Dim wrt As New MXXMLWriter50
On Error GoTo errorHandler
'Load the DOM document.
xmlDoc.async = False
xmlDoc.Load App.Path & "\books.xml"
'Set properties on the XML writer.
wrt.byteOrderMark = True
wrt.omitXMLDeclaration = True
wrt.indent = True
'Set the XML writer to the SAX content handler.
Set rdr.contentHandler = wrt
Set rdr.dtdHandler = wrt
Set rdr.errorHandler = wrt
rdr.putProperty "http://xml.org/sax/properties/lexical-handler", wrt
rdr.putProperty "http://xml.org/sax/properties/declaration-handler", wrt
'Parse the DOMDocument object.
rdr.parse xmlDoc


'Show the results in the text box.
Text1.Text = Text1.Text & wrt.output

Exit Sub

errorHandler:
    HandleErrors
End Sub

See Also

Passing a DOMDocument Object as a Parse() Parameter | ISAXXMLReader Interface | parse Method | DOMDocument | Sample XML File (books.xml)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Dev
Passing a DOMDocument Object as a Parse() Parameter

To parse a DOMDocument object with SAX, all you have to do is create a DOMDocument object, for example:

```vbnet
Dim xmlDoc As New DOMDocument50
```

Instead of a file or URL, specify the DOMDocument object as a parameter of the SAXXMLReader.parse() method, for example:

```vbnet
rdr.parse xmlDoc
```

See Also

ISAXXMLReader Interface | parse Method | DOMDocument
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Use MXXMLWriter

An MXXMLWriter object provides an efficient and flexible way to handle the output produced by the latest version of the Simple API for XML (SAX2) events. For example, an MXXMLWriter object:

- Automatically handles the details of building an XML document or document fragment. When connected to SAXXMLReader, MXXMLWriter accumulates data passed by events thrown by the reader and stores the data in an output string. Also, MXXMLWriter can automatically handle the details of inserting special characters, such as the "<" character at the beginning of an element.
- Enables you to format output by providing properties, such as omitXMLDeclaration, indent, and encoding, that apply to the document output.
- Enables you to manually build an XML document by invoking the methods of the IVBSAXContentHandler, IVBSAXDTDHandler, IVBSAXDeclHandler, IVBSAXErrorHandler, and IVBSAXLexicalHandler interfaces.
- Implements the IMXWriter interface, which is a complementary application programming interface (API) to the latest version of the Simple API for XML (SAX2) implementation of Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office.

MSXML provides two CoClasses that implement IMXWriter: MXXMLWriter, which generates XML or DOM output and MXHTMLWriter, which generates HTML output.

To help you understand how you can incorporate MXXMLWriter into your applications, this section provides the following topics:

- Overview of the Sample MXXMLWriter Application
- Connecting an MXXMLWriter to a SAXXMLReader
- Configuring MXXMLWriter Output
- Manually Building an XML Document with MXXMLWriter

Although this tutorial covers the code included in the sample application, it is recommended that you download the sample MXXMLWriter application from
the MSDN Code Center.

**Other Resources**

[Sample MXXMLWriter application](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Overview of the Sample MXXMLWriter Application

The sample MXXMLWriter application, shown in the following illustration, consists of a single form with two buttons that enable you to perform two basic tasks:

- The **Try File** button results in SAXXMLReader parsing the file given in the **File name** box, and passing the events of that file to MXXMLWriter. The application then directs the output from MXXMLWriter to the large text box on the right. (The **File name** box defaults to the **test.xml** file provided with the sample application.)
- The **Try Demo** button results in the application calling the required methods to manually build an XML file. The text box on the right shows the completed XML file. The text box on the left lists the events fired when each method is called.
Other Resources

Connecting an MXXMLWriter to a SAXXMLReader
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Developer’s Guide
Connecting an MXXMLWriter to a SAXXMLReader

In many of the applications that you create with the latest version of the Simple API for XML (SAX2), you use both SAXXMLReader, which reads an XML document and passes events, and MXXMLWriter, which catches the events and produces output based on the data passed by the events. Before MXXMLWriter can catch events, however, you must connect it to SAXXMLReader. MXXMLWriter, on its own, does not provide the handlers required to catch events thrown by SAXXMLReader.

To enable MXXMLWriter to handle events, you must assign the SAX2 handler interfaces to it. These handler interfaces include IVBSAXContentHandler, IVBSAXDTDHandler, IVBSAXDeclHandler, IVBSAXErrorHandler, and IVBSAXLexicalHandler. By assigning the SAX2 handler interfaces to your MXXMLWriter object, you make all the properties and methods of the interfaces available to it. In this way, your MXXMLWriter object can catch any of the events passed by SAXXMLReader. The following sections describe how to create SAXXMLReader and MXXMLReader objects and how to connect them.
Creating the SAXXMLReader and MXXMLWriter Objects

To create instances of SAXXMLReader and MXXMLWriter, use the following syntax. For the sample MXXMLWriter application, the creation of the SAXXMLReader and MXXMLWriter objects is in the Declarations section of the form.

```vbnet
Dim rdr As New SAXXMLReader
Dim wrt As New MXXMLWriter
```
Type Casting MXXMLWriter to Handler Interfaces

SAX2 provides several handler interfaces that are designed to catch events passed by SAXXMLReader. You set these handler interfaces as properties of SAXXMLReader. To make these handler interfaces available to MXXMLWriter, you type cast MXXMLWriter to them.

For example, the sample MXXMLWriter application sets the contentHandler property of the rdr (SAXXMLReader) object to the wrt (MXXMLWriter) object, as shown in the following code. This registers the wrt object as the ContentHandler of the rdr object. This also sets the dtdHandler and errorHandler properties. You must set the declaration-handler and lexical-handler properties through the putProperty method.

```
Private Sub Form_Load()
    Set rdr.contentHandler = wrt
    Set rdr.dtdHandler = wrt
    Set rdr.errorHandler = wrt
    rdr.putProperty "http://xml.org/sax/properties/declaration-handler", wrt
    rdr.putProperty "http://xml.org/sax/properties/lexical-handler",
End Sub
```
Setting the Output Destination

In the sample MXMLWriter application, clicking the **Try File** button clears both the text box and the output buffer (wrt.output). Then, the application calls the parse.URL method of SAXXMLReader and passes the name of the file in the TextFileName text box (as shown in bold in the following code sample). At this point, SAXXMLReader fires a series of events to pass the data in the XML document to the wrt object, which stores this data in the output buffer. By default, the output is a string value that, in this case, is set to the textResult.text box (as shown in bold in the following code sample). The output destination can also be set as an IStream.

```
Private Sub CommandTryFile_Click()
    ' Set parameters, clean the scene.
    TextSource.Text = ""
    wrt.output = ""
    wrt.omitXMLDeclaration = True
    On Error GoTo uhoh
    rdr.parseURL TextFileName.Text
    TextResult.Text = wrt.output
    Exit Sub

    uhoh:
    TextResult.Text = "Error: " & Err.LastDllError & " -- " & _
        Err.Description & vbCrLf & "Check if file exists in the _
        correct place"

End Sub
```

See Also

- Configuring MXMLWriter Output
- ISAXXMLReader Interface
- IMXWriter Interface
- ISAXContentHandler Interface
- ISAXDeclHandler Interface
- ISAXDTDHandler Interface
- ISAXErrorHandler Interface
- ISAXLexicalHandler Interface
- output Property
Configuring MXXMLWriter Output

MXXMLWriter provides properties that you can set to control its output, including encoding, indent, omitXMLDeclaration, and output.

For example, the sample MXXMLWriter application creates a document fragment from an XML document. It does this by the way it sets the omitXMLDeclaration property. When you set this property to True (as shown in bold in the following code sample), MXXMLWriter strips the XML declaration from an XML document before placing the document in the internal output buffer.

Private Sub CommandTryFile_Click()
    ' Set parameters, clean the scene.
    TextSource.Text = ""
    wrt.output = ""
    wrt.omitXMLDeclaration = True
    On Error GoTo uhoh
    rdr.parseURL TextFileName.Text

    TextResult.Text = wrt.output
    Exit Sub

The following table shows how setting the omitXMLDeclaration property to True removes the XML declaration from a document.

<table>
<thead>
<tr>
<th>wrt.omitXMLDeclaration = False</th>
<th>wrt.omitXMLDeclaration = True</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-16&quot; standalone=&quot;no&quot;?&gt;</td>
<td>&lt;root aaa=&quot;microsoft.com&quot;&gt;</td>
</tr>
<tr>
<td>&lt;root aaa=&quot;microsoft.com&quot;&gt;</td>
<td>&lt;PARTS&gt;...</td>
</tr>
<tr>
<td>&lt;PARTS&gt;...</td>
<td></td>
</tr>
</tbody>
</table>

There are other ways that you can configure your MXXMLWriter object to control its output. For more information about the properties you can set for MXXMLWriter, see the IMXWriter Interface.

See Also
Manually Building an XML Document with MXXMLWriter
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Manually Building an XML Document with MXXMLWriter

Using MXXMLWriter, you can build an XML document by manually calling methods of the Simple API for XML (SAX2) handlers. This section demonstrates how you can use type casting to get the interfaces that MXXMLWriter implements and how you can manually make calls to the methods of these interfaces to build an XML document. In this case, the application emulates SAXXMLReader by manually invoking the methods of the various handler interfaces, including IVBSAXContentHandler, IVBSAXDTDHandler, IVBSAXDeclHandler, IVBSAXErrorHandler, and IVBSAXLexicalHandler. In this portion of the sample MXXMLWriter application, SAXXMLReader is not used.

On its own, the IMXXMLWriter interface does not provide the handler methods required to manually construct an XML document. However, you can type cast MXXMLWriter to the required handler interfaces, which provides all the methods and properties to MXXMLWriter.

As you can see in the following sample application code, clicking the **Try Demo** button results in the application declaring the handler interface variables and then setting these handlers to the *wrt* object. This makes it possible for the *wrt* object to use any of the methods or properties of the declared handler interfaces.

Private Sub CommandTryDemo_Click()
' We need these variables for typecasting the writer.
Dim cnth As IVBSAXContentHandler
Dim dtdh As IVBSAXDTDHandler
Dim lexh As IVBSAXLexicalHandler
Dim dech As IVBSAXDeclHandler
Dim errh As IVBSAXErrorHandler

' This is just a helper.
Dim atrs As New SAXAttributes50

' Set them all to writer. Writer implements all these interfaces
Set cnth = wrt
Set dtdh = wrt
Set lexh = wrt
Set dech = wrt
Set errh = wrt

' Set parameters, clean the scene.
TextSource.Text = ""

In the sample MXMLWriter application, the remainder of the CommandTryDemo_Click() subroutine constructs the document by calling various handler methods. For example, the following statement invokes the startDocument method of the IVBSAXContentHandler interface:

    cnth.startDocument

In a similar way, the following statement invokes the startDTD method of the IVBSAXLexicalHandler interface and passes the parameters to the wrt object:

    lexh.startDTD "MyDTD", "", "http://eureka.sample/mydtd.dtd"

Coupled with each call to a handler method, there is a call to the log subroutine, as follows:

    log "Content->startDocument"

The log subroutine, as shown in the following code, creates an entry in the left-hand text box for each method called.

Private Sub log(msg As String)
    TextSource.Text = TextSource.Text & vbCrLf & msg
End Sub

At the end of the CommandTryDemo_Click() subroutine, the application sets the textResults.text box to the output of the writer (as in the following code). Unless you flush the output buffer, it continues to aggregate data passed to it by the handlers.

TextResult.Text = wrt.output
Complete Code for the CommandTryDemo_Click() Subroutine

Private Sub CommandTryDemo_Click()
    ' We need these variables for typecasting the writer.
    Dim cnth As IVBSAXContentHandler
    Dim dtdh As IVBSAXDTDHandler
    Dim lexh As IVBSAXLexicalHandler
    Dim dech As IVBSAXDeclHandler
    Dim errh As IVBSAXErrorHandler

    ' This is just a helper.
    Dim atrs As New SAXAttributes50

    ' Set handler variables to the writer so it implements the interf
    Set cnth = wrt
    Set dtdh = wrt
    Set lexh = wrt
    Set dech = wrt
    Set errh = wrt

    ' Set parameters, clean the scene.
    TextSource.Text = ""

    ' Manually call necessary events to generate the XML file.
    log "Content->startDocument"
    cnth.startDocument
    log "Lexical->startDTD"
    lexh.startDTD "MyDTD", "", "http://eureka.sample/mydtd.dtd"
    log "Decl->elementDecl"
    dech.elementDecl "book", "title | descr"
    log "Decl->attributeDecl"
    dech.attributeDecl "book", "author", "CDATA", "#IMPLIED", ""
    log "Decl->attributeDecl"
    log "Decl->attributeDecl"
    dech.attributeDecl "book", "cover", "(hard|soft)", "", "soft"
    log "Decl-elementDecl"
    dech.elementDecl "title", "(#PCDATA)"
    log "Decl-elementDecl"
    dech.elementDecl "descr", "(#PCDATA)"
    log "Lexical->endDTD"
    lexh.endDTD
    log "Content->startElement"
    atrs.addAttribute "", "", "cover", "", "hard"
cnth.startElement "", "", "book", atrs
   log "Content->startElement"
   atrs.Clear
   cnth.startElement "", "", "title", atrs
   log "Content->characters"
   cnth.characters "On the Circular Problem of Quadratic Equati
   log "Content->endElement"
   cnth.endElement "", "", "title"
   log "Content->endElement"
   cnth.endElement "", "", "book"

   TextResult.Text = wrt.output
End Sub

See Also

ISAXXMLReader Interface | IMXWriter Interface | ISAXContentHandler Interface | ISAXDeclHandler Interface | ISAXDTDHandler Interface | ISAXErrorHandler Interface | ISAXLexicalHandler Interface
Extract Data From a Large Document

This example demonstrates how to create an XML data extracting application in Microsoft® Visual Basic® using MSXML 5.0. The XML Extractor application uses SAX to parse a large XML file, and extracts data from the file to generate multiple DOM documents. The DOM documents are then processed to generate HTML output files. The primary purpose of the application is to show you how SAX and DOM can be used together to perform optimal XML processing. The design of the application emphasizes the respective strengths of both SAX and DOM.

For input, we will use a single XML file, invoices.xml. This file consists of many similar XML trees. It has a document root, <invoices>, which can contain one or more instances of the <invoice> element. Each occurrence of <invoice> contains the data to create an invoice for a patient, billing them for their medical expenses.

For output, the goal of the XML Extractor application is to extract the data for each patient invoice as a separate document that can be used to create the report document, a patient's bill. This is done by applying an XSLT style sheet file, invoice.xsl. The output is an HTML-formatted patient invoice. Each invoice is saved to its own new file, and these files can later be printed and mailed to the patients.

Each invoice is relatively small, so they can be easily processed using only the DOM and XSLT. However, the entire XML document contains numerous invoices and might be too large to load into the DOM. Besides, each bill should be processed separately to generate a separate HTML file to print and send.

The following are some of the advantages of the design for the XML Extractor application.

- It can use SAX to read a large XML document, and create DOM trees for each <invoice> element. This reuses memory and therefore wastes much less of it.
For each <invoice> element, the Extractor generates a small DOM document, which it immediately processes to generate the output. This document is discarded before the following element is read, and the process repeats. This also conserves memory for the application.

This topic is divided into the following sections.

- Overview of the XML Extractor Application
- Application Forms (XML Extractor)
- Sample Files (XML Extractor)
- MyExtractor Class (XML Extractor)
- Run the Application (XML Extractor)
- How the XML Extractor Application Works
Overview of the XML Extractor Application

The XML Extractor application uses the following SAX interfaces/CoClasses:

- ISAXXMLReader
- MXXMLWriter
- IVBSAXXMLFilter
- IVBSAXContentHandler
- IVBSAXErrorHandler

The application consists of the following three main parts.

**Main form**
A Microsoft® Visual Basic® 6.0 form that provides the initial startup form for running the application.

The form has a text box to specify the file used as input and two command buttons: Start and Exit. Pressing Start starts application processing. Pressing Exit unloads the application form, effectively exiting the application.

**Secondary form**
This Microsoft® Visual Basic® 6.0 form displays the invoices as they are generated. It only appears when the application starts. It permits you to display and walk through the invoices while generating them.

**MyExtractor class**
A Microsoft® Visual Basic® 6.0 class module that implements the interfaces listed above. When SAX parsing begins, the SAX reader throws a series of events that are received by an instance of this class.

**Sample Files**
The XML Extractor application requires several files. These include a single XML file (invoices.xml) to be used as the input for making DOM document trees. There is also a single XSLT style sheet file (invoice.xsl) used to transform each of these trees as a separate invoice, and to generate application output in HTML format.
See Also

Extract Data From a Large Document | Application Forms (XML Extractor) | Sample Files (XML Extractor) | MyExtractor Class (XML Extractor) | Run the Application (XML Extractor) | How the XML Extractor Application Works
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Sample Files (XML Extractor)

To run the application you must create the following files:

- A sample XML file, **invoices.xml**. This file contains the patient invoices for a fictitious doctor's office.
- A sample XSLT style sheet file, **invoice.xsl**. This file is used to transform the patient invoice data, which is extracted using SAX content handling. Each set of patient data is transformed into an HTML file.
- A GIF image file. This file contains a piece of business logo art for the fictitious medical center using this application.
Complete Code for invoices.xml

To create the sample XML file

1. Open Notepad.
2. Select the code for invoices.xml (below). Then copy it, and paste it in Notepad.
3. From Notepad, save the file as invoices.xml to the same folder that will contain the XML Extractor application.

invoices.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE invoices [
  <!ELEMENT invoices (invoice)*> 
  <!ELEMENT invoice (patient, insurance?, procedure*)>
  <!ATTLIST invoice number CDATA #REQUIRED date CDATA #REQUIRED> 
  <!ELEMENT patient (phone | address)*> 
  <!ATTLIST patient firstname CDATA #REQUIRED familyname CDATA #REQUIRED SSN CDATA ">
  <!ELEMENT insurance (address | phone)*> 
  <!ATTLIST insurance name CDATA "" plannumber CDATA "" planname CDATA ">
  <!ELEMENT procedure (#PCDATA)>
  <!ATTLIST procedure code CDATA #REQUIRED name CDATA "" cost CDATA "" insurance_estimate CDATA "" submitted (yes|no) "yes"> 
  <!ELEMENT phone EMPTY> 
  <!ATTLIST phone type (home|work|mobile|other) "other" number CDATA ">
  <!ELEMENT address EMPTY> 
  <!ATTLIST address type (home|work|business|other) "other" company CDATA "" line1 CDATA "" line2 CDATA "" city CDATA "" state CDATA "" zip CDATA ">
]> 
<invoices>
```
<invoice number="25" date="February 28, 2001">
  <patient firstname="Jeff" familyname="Smith" SSN="123456789">
    <phone type="home" number="123-4567890"/>
    <phone number="321-76543321" type="work"/>
    <address type="home" line1="123 Street" city="City" state="US"/>
  </patient>
  <insurance name="Humongous First Medical Insurance" plannumber="">
    <phone number="098-76543321"/>
    <address type="business" line1="321 Street" city="City" state="US"/>
  </insurance>
  <procedure code="123" name="Cleaning nose" cost="50.00" insurance_estimate="50.00"/>
  <procedure code="124" name="Tarot reading of illnesses" cost="150.00" insurance_estimate="120.00"/>
  <procedure code="125" name="Just for fun" cost="100.00" insurance_estimate="80.00"/>
</invoice>

<invoice number="27" date="February 28, 2001">
  <patient firstname="James" familyname="Smith" SSN="123456765">
    <phone type="home" number="123-4562245"/>
    <address type="home" line1="432 Street" city="City" state="US"/>
  </patient>
  <insurance name="Humongous Second Medical Insurance" plannumber="">
    <phone number="098-76543321"/>
    <address type="business" line1="344 Street" city="Some City" state="US"/>
  </insurance>
  <procedure code="123" name="Cleaning nose" cost="50.00" insurance_estimate="50.00"/>
  <procedure code="124" name="Tarot reading of illnesses" cost="150.00" insurance_estimate="120.00"/>
</invoice>

<invoice number="29" date="February 28, 2001">
  <patient firstname="Neil" familyname="Smith" SSN="123456345">
    <phone type="home" number="125-4345890"/>
    <address type="home" line1="187 Street" city="Lost City" state="US"/>
  </patient>
  <insurance name="Humongous Third Medical Insurance" plannumber="">
    <phone number="198-76345321"/>
    <address type="business" line1="342 Street" city="Completely Lost City" state="US"/>
  </insurance>
  <procedure code="123" name="Cleaning nose" cost="50.00" insurance_estimate="50.00"/>
  <procedure code="125" name="Maybe they wouldn't see this line..." cost="100.00" insurance_estimate="80.00"/>
</invoice>
Complete Code for invoice.xsl

To create the sample XSLT file

1. Open Notepad.
2. Select the code for invoice.xsl (below). Then copy it, and paste it in Notepad.
3. From Notepad, save the file as invoice.xsl to the same folder that will contain the XML Extractor application.

invoices.xsl

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <HTML>
      <BODY bgcolor="#FFFFE0">
        <TABLE border="0" width="100%">
          <TR><TD>
            <H1>Invoice #<xsl:value-of select="invoice/@number"/>
            ,
            <xsl:value-of select="invoice/@date"/></H1>
          </TD><TD align="right"><img src="sax_extractData_logo.gif"/>
        </TD></TR>
        <TABLE border="0" width="100%">
          <TR valign="top">
            <TD>
              <xsl:for-each select="/invoice/patient">
                To: <xsl:value-of select="@firstname"/>
                <xsl:text> </xsl:text>
                <xsl:value-of select="@familyname"/>
                <BR/>
                Account #<xsl:value-of select="@SSN"/>
                <BR/>
                <xsl:value-of select="address/@line1"/>
                <BR/>
                <xsl:if test="address/@line2=''">
                  <xsl:value-of select="address/@line2"/>
                </xsl:if>
                <xsl:value-of select="address/@city"/>
                ,
                <xsl:value-of select="address/@state"/>
                <xsl:value-of select="address/@zip"/>
              </xsl:for-each>
            </TD>
            <TD>
              <xsl:for-each select="/invoice/insurance">
              </xsl:for-each>
            </TD>
          </TR>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```
Creating the Logo Art File

To create the GIF logo art file

1. Open Paint.
2. Right-click on the image (below).
3. Select Copy to copy the image to the Windows Clipboard.
4. From Paint, paste the image and save it to file as sax_extractData_logo.gif. Save the image in the same folder that will contain the XML Extractor application.

See Also

Extract Data From a Large Document | Overview of the XML Extractor Application | Application Forms (XML Extractor) | MyExtractor Class (XML Extractor) | Run the Application (XML Extractor) | How the XML Extractor Application Works
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
Application Forms (XML Extractor)

To get started, create a new project and reference Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office.

To create a new project

- Open Microsoft® Visual Basic® 6.0. In the New Project dialog box, double-click Standard EXE.

To create a reference to MSXML 5.0

1. On the Project menu, click References.
2. In the Available References list, select Microsoft XML,v5.0, and then click OK.
3. Save the project.
Build the Main Form

First, you must build the initial form used by the XML Extractor application. Add the following controls to the form.

- A text box for entering the name of the XML file to be used as input.
- A label for the text box control.
- Two command buttons. One is for starting the application. The other is to exit the application.

For your application form to work with the sample code provided in the next section, you must set the properties in the following table.

<table>
<thead>
<tr>
<th>Control</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Caption</td>
<td>&quot;Medical bills (XML Extractor) Example&quot;</td>
</tr>
<tr>
<td>Label1</td>
<td>Caption</td>
<td>&quot;File name:&quot;</td>
</tr>
<tr>
<td>(Name)</td>
<td></td>
<td>txtFilename</td>
</tr>
<tr>
<td>Text1</td>
<td>Text</td>
<td>&quot;invoices.xml&quot;</td>
</tr>
<tr>
<td>(Name)</td>
<td></td>
<td>cmdStart</td>
</tr>
<tr>
<td>Command1</td>
<td>Caption</td>
<td>&quot;Start&quot;</td>
</tr>
<tr>
<td>(Name)</td>
<td></td>
<td>cmdExit</td>
</tr>
<tr>
<td>Command2</td>
<td>Caption</td>
<td>&quot;Exit&quot;</td>
</tr>
</tbody>
</table>

After you modify the property settings, resize the controls and arrange them on the form until your user interface looks like the following:
Complete Code for the Main Form

The following shows the complete code for the main form. To run the code as you read, select all the text, then copy it and paste it into the form of your own Microsoft® Visual Basic® project.

Option Explicit

Private builder As MXXMLWriter50
Private stylesheet As DOMDocument50

Private Sub cmdExit_Click()
End
End Sub

Private Sub cmdStart_Click()
    Dim reader As New SAXXMLReader50
    Dim filter As New MyExtractor
    Dim builder As New MXXMLWriter50

    builder.output = New DOMDocument50
    Set stylesheet = New DOMDocument50
    stylesheet.Load App.Path & vbCrLf & "invoices.xsl"

    Set filter.IVBSAXXMLFilter_parent = reader
    Set filter.SAXXMLReader50_contentHandler = builder
    filter.cutElement = "invoice"

    On Error GoTo Uh_Oh
    reader.parseURL App.Path & vbCrLf & txtFilename.Text
    Exit Sub

    Uh_Oh:
        MsgBox "*** Error while processing invoices *** " & Err.Description & vbCrLf & Err.Source
    Exit Sub

    Public Sub processInvoice(dom As MSXML2.DOMDocument50, sInvNumber As String)
        Dim s As String
        s = dom.transformNode(stylesheet)
        saveToFile s, App.Path & vbCrLf & "invoice" & sInvNumber & ".html"
        frmInvoice.txtAsXml = dom.xml
        frmInvoice.txtAsHtml = s
        frmInvoice.webPreviewPane.Navigate App.Path & vbCrLf & "invoice" & sInvN
frmInvoice.Show vbModal
End Sub

Private Sub saveToFile(text As String, fname As String)
    Open fname For Output As #5
    Print #5, text
    Close #5
End Sub
Build the Secondary Form

Next, add the secondary invoice preview form used by the XML Extractor application.

To add a new form to the project

1. From the Project menu, click Add Form.
2. In the Add Form dialog box, click Open.

The preview form should allow the user to step through and preview the invoices as output in a Web browser. To support this feature, use the custom WebBrowser control. This is a custom ActiveX control provided as part of the Microsoft Internet Controls library. Therefore, you must add a reference in your project to this component library.

To create a reference to Microsoft Internet Controls

1. On the Project menu, click Components.
2. In the Components dialog box, from the Controls tab, select Microsoft Internet Controls from the list. Click OK.

   The Web Browser control will now appear in the Toolbox, allowing you to add it to the new form.

3. Save the project.

You can now add the following controls to the form.

- A custom Web browser control. This enables you to use an embedded Web browser window on your form. This browser control will be used to view each output invoice as a final-formatted HTML file.
- Two text boxes for showing source text views. One will show the invoices as XML, and one will show the transformed output as HTML.
- Two command buttons: one to advance to the next invoice in the file, and one to hide the preview window and return to the main form.

For your application form to work with the sample code provided in the next
section, you must set the properties in the following table.

<table>
<thead>
<tr>
<th>Control</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Caption</td>
<td>&quot;Invoice&quot;</td>
</tr>
<tr>
<td></td>
<td>(Name)</td>
<td>frmInvoice</td>
</tr>
<tr>
<td>Text1</td>
<td>MultiLine</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>ScrollBars</td>
<td>2 - Vertical</td>
</tr>
<tr>
<td></td>
<td>(Name)</td>
<td>txtAsXml</td>
</tr>
<tr>
<td>Text2</td>
<td>MultiLine</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>ScrollBars</td>
<td>2 - Vertical</td>
</tr>
<tr>
<td>WebBrowser1</td>
<td>(Name)</td>
<td>webPreviewPane</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>cmdNext</td>
</tr>
<tr>
<td>Command1</td>
<td>Caption</td>
<td>&quot;Next invoice&quot;</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>cmdBreakAndExit</td>
</tr>
<tr>
<td>Command2</td>
<td>Caption</td>
<td>&quot;Break and exit&quot;</td>
</tr>
</tbody>
</table>

After you modify the property settings, resize the controls and arrange them on the form until your user interface looks like the following:
Complete Code for the Secondary Form

The following shows the complete code for the secondary form. To run the code as you read, select all the text, then copy it and paste it into the form of your own Microsoft® Visual Basic® project.

Private Sub cmdBreakAndExit_Click()
    End
End Sub

Private Sub cmdNext_Click()
    Hide
End Sub

See Also

Extract Data From a Large Document | Overview of the XML Extractor Application | Sample Files (XML Extractor) | MyExtractor Class (XML Extractor) | Run the Application (XML Extractor) | How the XML Extractor Application Works
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Deve
MyExtractor Class (XML Extractor)

To create the class

1. On the Project menu, click Add Class Module.
2. In the Add Class Module dialog box, double-click Class Module.
4. In the Properties Window, for the Name property, type "MyExtractor".

To implement an interface

1. In the code window, type "Implements" and the name of the interface, for example:

   Implements IVBSAXContentHandler
   Implements IVBSAXErrorHandler
   Implements SAXXMLReader50
   Implements IVBSAXXMLFilter

2. In the left-hand drop-down list in the code window, select the interface.
3. In the right-hand drop-down list in the code window, you can implement the methods for the interface by selecting them from the list.

   Note  You must implement all methods for the implemented interfaces.
Complete Code for MyExtractor

Add the following code to the class.

**Note** If you already added the Implements statements, you can simply copy the following code and paste it before the first Implements statement.

```vbs
Option Explicit

Implements IVBSAXContentHandler
Implements IVBSAXErrorHandler
Implements SAXXMLReader50
Implements IVBSAXXMLFilter

Private parent As SAXXMLReader50
Private ch As IVBSAXContentHandler
Private strInvoiceNumber As String
Private putThrough As Boolean

Public cutElement As String

Public Sub IVBSAXContentHandler_characters(strChars As String)
    If Not IsEmpty(ch) And putThrough Then
        ch.characters strChars
    End If
End Sub

Public Property Set IVBSAXContentHandler_documentLocator(ByVal RHS As MSXML2.IVBSAXLocator)
End Property

Public Sub IVBSAXContentHandler_endDocument()
End Sub

Public Sub IVBSAXContentHandler_endElement(strNamespaceURI As String, strLocalName As String, strQName As String)
    Dim bld As MXXMLWriter50
    If Not IsEmpty(ch) Then
        If putThrough Then
            ch.endElement strNamespaceURI, strLocalName, strQName
        End If
        If strQName = cutElement Then
            ch.endDocument
            Set bld = ch 'Typecast the writer.
            putThrough = False
            Form1.processInvoice bld.output, strInvoiceNumber
    End If
End Sub
```
Public Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As String)
    If Not IsEmpty(ch) And putThrough Then
        ch.endPrefixMapping strPrefix
    End If
End Sub

Public Sub IVBSAXContentHandler_ignorableWhitespace(strChars As String)
End Sub

Public Sub IVBSAXContentHandler_processingInstruction(strTarget As String, strData As String)
End Sub

Public Sub IVBSAXContentHandler_skippedEntity(strName As String)
End Sub

Public Sub IVBSAXContentHandler_startDocument()
    putThrough = False
End Sub

Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As String, strLocalName As String, strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttributes)
    If Not IsEmpty(ch) Then
        If strQName = cutElement Then
            Dim bld As Msxml2.MXXMLWriter50
            Set bld = ch ' typecast
            bld.output = New Msxml2.DOMDocument50
            putThrough = True
            ch.startDocument
            strInvoiceNumber = oAttributes.getValueFromName("", "num
        End If
        If putThrough Then
            ch.startElement strNamespaceURI, strLocalName, strQName, oAttributes
        End If
    End If
End Sub

Public Sub IVBSAXContentHandler_startPrefixMapping(strPrefix As String, strURI As String)
    If Not IsEmpty(ch) And putThrough Then
        ch.startPrefixMapping strPrefix, strURI
    End If
End Sub

Private Sub IVBSAXErrorHandler_error(ByVal oLocator As MSXML2.IVBSAXLocator, strErrorMessage As String, ByVal nErrorCode As Long)
End Sub

Private Sub IVBSAXErrorHandler_fatalError(ByVal oLocator As MSXML2.IVBSAXLocator, strErrorMessage As String, ByVal nErrorCode As Long)
End Sub
MsgBox strErrorMessage & " " & oLocator.lineNumber
End Sub

Private Sub IVBSAXErrorHandler_ignorableWarning(ByVal oLocator As MSXML2.IVBSAXLocator, strErrorMessage As String, ByVal nErrorCode As Long)
End Sub

Public Property Set IVBSAXXMLFilter_parent(ByVal RHS As MSXML2.SAXXM
Set parent = RHS
Set RHS.contentHandler = Me
Set RHS.errorHandler = Me
End Property

Public Property Get IVBSAXXMLFilter_parent() As Msxml2.SAXXMLReader5
IVBSAXXMLFilter_parent = parent
End Property

Public Property Let SAXXMLReader50_baseURL(ByVal RHS As String)
If Not IsEmpty(parent) Then
  parent.baseURL = RHS
End If
End Property

Public Property Get SAXXMLReader50_baseURL() As String
If Not IsEmpty(parent) Then
  SAXXMLReader50_baseURL = parent.baseURL
End If
End Property

Public Property Set SAXXMLReader50_contentHandler(ByVal RHS As MSXML
Set ch = RHS
End Property

Public Property Get SAXXMLReader50_contentHandler() As MSXML2.IVBSAX
SAXXMLReader50_contentHandler = ch
End Property

Public Property Set SAXXMLReader50_dtdHandler(ByVal RHS As MSXML2.IV
End Property

Public Property Get SAXXMLReader50_dtdHandler() As MSXML2.IVBSAXDTDH
End Property

Public Property Set SAXXMLReader50_entityResolver(ByVal RHS As MSXML
End Property

Public Property Get SAXXMLReader50_entityResolver() As MSXML2.IVBSAX
End Property

Public Property Set SAXXMLReader50_errorHandler(ByVal RHS As MSXML2.
Public Property Get SAXXMLReader50_errorHandler() As MSXML2.IVBSAXErrorHandler
End Property

Private Function SAXXMLReader50_getFeature(ByVal strName As String) As Boolean
    If Not IsEmpty(parent) Then
        SAXXMLReader50_getFeature = parent.getFeature(strName)
    End If
End Function

Private Function SAXXMLReader50_getProperty(ByVal strName As String) As Variant
    ' Do not pass.
End Function

Public Sub SAXXMLReader50_parse(Optional ByVal varInput As Variant)
    If Not IsEmpty(parent) Then
        parent.parse varInput
    End If
End Sub

Public Sub SAXXMLReader50_parseURL(ByVal strURL As String)
    If Not IsEmpty(parent) Then
        parent.parseURL strURL
    End If
End Sub

Public Sub SAXXMLReader50_putFeature(ByVal strName As String, ByVal fValue As Boolean)
    If Not IsEmpty(parent) Then
        parent.putFeature strName, fValue
    End If
End Sub

Public Sub SAXXMLReader50_putProperty(ByVal strName As String, ByVal varValue As Variant)
    ' Do not pass.
End Sub

Public Property Let SAXXMLReader50_secureBaseURL(ByVal RHS As String)
    If Not IsEmpty(parent) Then
        parent.secureBaseURL = RHS
    End If
End Property

Public Property Get SAXXMLReader50_secureBaseURL() As String
    If Not IsEmpty(parent) Then
        SAXXMLReader50_secureBaseURL = parent.secureBaseURL
    End If
End Property
See Also

Extract Data From a Large Document | Overview of the XML Extractor Application | Application Forms (XML Extractor) | Sample Files (XML Extractor) | Run the Application (XML Extractor) | How the XML Extractor Application Works
Run the Application (XML Extractor)

Once you have created all of the sample files and set up your Visual Basic project, follow these instructions to run the application.

To run the application

1. Save the project.

   **Note** Be sure to have created the three sample files in the same folder where you have created the XML Extractor application.

2. From the Visual Basic toolbar, click **Start**.
3. From the **Medical bills (XML Extractor) sample** application, click **Start**.
4. In the **Invoice** form, do either of the following:
   - Click **Next invoice** to create and preview the next invoice file.
   - Click **Break and exit** to return to the main application form.

If the application is successful, the **Invoice** form displays the following results when first viewed.

You might also want to verify that HTML-based invoice files are being created and named for each invoice by number. For example, if you view the folder where you created and are running the XML Extractor application in Windows Explorer, you should see files being added to the folder as you walk through the invoices using the application. These files should following the naming pattern `invoicexx.html`, where `xx` is the current invoice number for the invoice you are previewing. For example, if you preview the second invoice you should see the file `invoice27.html` added to the folder, in addition to the `invoice265.html` file created when the application first runs.

**See Also**
How the XML Extractor Application Works

The XML Extractor application uses SAX to read the XML sample file (invoices.xml), and to process it for each invoice. This approach conserves memory by creating and loading a small DOMDocument file for each invoice, and then discarding it after the invoice has generated the intended output — that is, an HTML file containing the patient's bill.

First, the SAX reader (SAXXMLReader50) connects to a content handler (IVBSAXContentHandler) implemented in the MyExtractor class module. The content handler performs the extraction work, connecting to a SAX filter (IVBSAXXMLFilter), which only lets through events that match the following XPath expression:

/invoice/**

This expression loosely translates to "match using the descendant-or-self axis to select the contents of all top-level <invoice> elements in the file."

With this selection criteria, you can simulate a filtered stream of multiple documents in the IVBSAXContentHandler implementation. You do this by firing the SAX startDocument() and endDocument() events for each instance of the <invoice> element in the sample XML file. The SAX reader has its output connected to the SAX writer (MXXMLWriter), which has been configured as a DOMDocument builder for generating each of the invoices.

For all of the three component interfaces implemented, only the SAX filter is custom-built for this application. The filter also calls an application method (processInvoice) to process the next bill, each time the <invoice> element ends and a corresponding DOMDocument tree is available in the DOM builder (MXXMLWriter).

As you review the sample code, note that the SAX filter remains unaware of what the output implementation (in this sample, MXXMLWriter) is doing. Only the SAX reader and its content handler direct the output. This means that it
remains possible to change the application (for example, by connecting its output to another custom writer or by connecting it to another filter) without requiring any changes to the current filter code used in the application.

It is fairly simple to processing each DOM document. The transformNode method applies the XSLT style sheet file (invoice.xsl) to each DOMDocument tree generated for a particular <invoice> element. This generates the HTML version of the invoice and displays it in the invoice preview window. The application also saves the HTML version of each invoice to its own file. These files are named according to the invoicexx.html naming pattern where xx is the value of the number attribute associated with each <invoice> element previewed and generated.

See Also

Extract Data From a Large Document | Overview of the XML Extractor Application | Application Forms (XML Extractor) | Sample Files (XML Extractor) | MyExtractor Class (XML Extractor) | Run the Application (XML Extractor)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
SAX2 Reference

The Simple API for XML (SAX) was originally defined for the Java programming language using Java interface definitions. Because the Java language interfaces are not language-neutral, tool vendors must decide how the SAX interfaces will map to a specific language. In the future, however, standard language bindings will emerge.

Microsoft added support for SAX in MSXML 3.0. There is support for C++ and Microsoft Visual Basic®. Each of these language bindings requires a different set of interfaces that reflect the individual language and type restrictions.

The names of the MSXML interfaces are the same as those in the Java language, but are prefixed with ISAX for C++ and IVBSAX for Visual Basic (ISAXContentHandler and IVBSAXContentHandler, for example).

This section contains the following topics.

- IMXAttributes Interface
- IMXWriter Interface
- ISAXAttributes Interface
- ISAXContentHandler Interface
- ISAXDeclHandler Interface
- ISAXDTDHandler Interface
- ISAXEntityResolver Interface
- ISAXErrorHandler Interface
- ISAXLexicalHandler Interface
- ISAXLocator Interface
- ISAXXMLFilter Interface
- ISAXXMLReader Interface
- SAX2 C++ Common Notices
- SAX2 Visual Basic Common Notices
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
IMXAttributes Interface

The IMXAttributes interface complements the implementation of the ISAXAttributes interface, which can be used to query attribute collections. Beyond the ability to query existing attribute collections using ISAXAttributes, the IMXAttributes interface allows you to create and edit attribute collections.

The implemented CoClass for ISAXAttributes and IMXAttributes is SAXAttributes50, which is the version-dependent class for Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office.

**Note**  CoClasses define a class to the Component Object Model (COM). The CoClass definitions include a list of interfaces supported by COM objects created from this class.

The implementation of the IMXAttributes interface returns only the list of attributes data stored inside of an attribute's collection object. Because the ISAXAttributes implementation cannot be sure which implementation is provided, it cannot be sure of the validity of attributes. The requirements of different applications may be different. For this reason, the implementation of the CoClass does not enforce any assumption about input information, accepting it whenever possible.

The two uses for this interface are:

- to take a persistent snapshot of an attribute.
- to construct or modify an attribute's collection object in a Simple API for XML (SAX2) driver or filter.

**Methods**

- **addAttribute** Adds an attribute to the end of the attribute collection.
- **addAttributeFromIndex** Adds an attribute, whose value is equal to the indexed attribute in the input attributes object, to the end of the list.
- **clear** Clears the attribute list for reuse.
removeAttribute

Sets an attribute in the list.

setAttribute

Copies all existing attributes of a collection to a specified object.

setAttributes

Sets the local name of a specific attribute.

setLocalName

Sets the qualified name (QName) of a specific attribute.

setQName

Sets the type of a specific attribute.

setType

Sets the namespace URI of a specific attribute.

setURI

Sets the value of a specific attribute.

setValue

Versioning

MSXML 3.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

Version 5.0 Dependent ProgID: msxml2.SAXAttributes50

Version 5.0 Dependent CLSID: 88d969ca-f192-11d4-a65f-0040963251e5
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
addAttribute Method

Adds an attribute to the end of the attribute collection.

[Visual Basic]

Implementation Syntax

Sub addAttribute(ByVal strURI As String, _
    ByVal strLocalName As String, _
    ByVal strQName As String, _
    ByVal strType As String, _
    ByVal strValue As String)

Usage Syntax

oIMXAttributes.addAttribute(strURI, _
    strLocalName, _
    strQName, _
    strType, _
    strValue)

Parameters

strURI
The namespace Uniform Resource Identifier (URI) or, if the namespace has no URI, an empty string.

strLocalName
The local name of the attribute.

strQName
The XML 1.0 qualified name or, if the qualified name is not available, an empty string.

strType
A string representing the attribute type.

strValue
A string representing the attribute default value specified by "#IMPLIED", "#REQUIRED", or "#FIXED", or Null if none applies. This value string is represented by the attribute's default value, or empty if there is none.
[C/C++]

Adds an attribute to the end of the attribute collection.

Syntax

HRESULT addAttribute(
    [in] BSTR strURI,
    [in] BSTR strLocalName,
    [in] BSTR strQName,
    [in] BSTR strType,
    [in] BSTR strValue);

Parameters

strURI
   The namespace Uniform Resource Identifier (URI) or, if the namespace has no URI, an empty string.

strLocalName
   The local name of the attribute.

strQName
   The XML 1.0 qualified name or, if the qualified name is not available, an empty string.

strType
   A string representing the attribute type.

strValue
   A string representing the attribute's default value specified by "#IMPLIED", "#REQUIRED", or "#FIXED", or Null if none applies. This value string is represented by the attribute's default value, or empty if there is none.

Return Values

S_OK
   The value returned if no errors are reported.

Remarks

To maximize efficiency, the addAttribute method does not check if the attribute is in the list. Therefore, the user application should check for the existence of an
attribute before adding it to the bottom of the attribute list.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [IMXAttributes Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**addOnAttributeFromIndex Method**

Adds an attribute, whose value is equal to the indexed attribute in the input attributes object, to the end of the list.

[Visual Basic]

**Implementation Syntax**

Sub addAttributeFromIndex(ByVal varAtts, ByVal nIndex As Long)

**Usage Syntax**

oIXAttributes.addAttributeFromIndex(varAtts, nIndex)

[C/C++]

**Syntax**

HRESULT addAttributeFromIndex(
    [in] VARIANT varAtts,
    [in] int nIndex);

**Return Values**

S_OK
  The value returned if no errors are reported.
E_FAIL
  The value returned if index is out of range.

**Parameters**

varAtts
  The variant type.
nIndex
  The attribute's index value.
To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
clear Method

Clears the attribute list for reuse.

[Visual Basic]

Implementation Syntax

Sub clear()

Usage Syntax

oIMXAttributes.clear

[C/C++]

Syntax

HRESULT clear();

Return Values

S_OK
  The value returned if no errors are reported.

E_FAIL
  There is only one specific fatal error for this IMXAttributes implementation: setAttribute with incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

Remarks

This call does not free memory. The current arrays are kept for reuse.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
removeAttribute Method

Removes an attribute from the list.

[Visual Basic]

Implementation Syntax

Sub removeAttribute(nIndex As Long)

Usage Syntax

oIMXAttributes.removeAttribute(nIndex)

[C/C++]

Syntax

HRESULT removeAttribute(
    [in] int nIndex);

Return Values

S_OK
   The value returned if no errors are reported.
E_FAIL
   There is only one specific fatal error for this IMXAttributes implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

Parameters

nIndex
   The attribute's index value (zero-based).

To view reference material for Visual Basic or C++ only, click the Language
Filter button in the upper-left corner of the page.

See Also

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
setAttribute Method

Sets an attribute in the list.

[Visual Basic]

Implementation Syntax

Sub setAttribute(ByVal nIndex As Long, _
    ByVal strURI As String, _
    ByVal strLocalName As String, _
    ByVal strQName As String, _
    ByVal strType As String, _
    ByVal strValue As String)

Usage Syntax

oIMXAttributes.setAttribute(nIndex, _
    strURI, _
    strLocalName, _
    strQName, _
    strType, _
    strValue)

Parameters

nIndex
   The attribute's index value (zero-based).
strURI
   The namespace Uniform Resource Identifier (URI) or, if the namespace has no URI, an empty string.
strLocalName
   The local name or, if namespace processing is not being performed, an empty string.
strQName
   The XML 1.0 qualified name or, if the qualified name is not available, an empty string.
strType

A string representing the attribute type.

strValue

The attribute value.

[C/C++]

Syntax

HRESULT setAttribute(
    [in] int nIndex,
    [in] BSTR strURI,
    [in] BSTR strLocalName,
    [in] BSTR strQName,
    [in] BSTR strType,
    [in] BSTR strValue);

Parameters

nIndex

The attribute's index value (zero-based).

strURI

The namespace Uniform Resource Identifier (URI) or, if the namespace has no URI, an empty string.

strLocalName

The local name or, if namespace processing is not being performed, an empty string.

strQName

The XML 1.0 qualified name or, if the qualified name is not available, an empty string.

strType

A string representing the attribute type.

strValue

The attribute value.

Return Values

S_OK

The value returned if no errors are reported.

E_FAIL
There is only one specific fatal error for this `IMXAttributes` implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: `IMXAttributes Interface`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
setAttributes Method

Copies all existing attributes of a collection to a specified object.

[Visual Basic]

Implementation Syntax

Sub setAttributes(varAtts)

Usage Syntax

oIMXAttributes.setAttribute(varAtts)

[C/C++]

Syntax

HRESULT setAttributes(IMXAttributes * varAtts);

Return Values

S_OK
   The value returned if no errors are reported.

Parameters

varAtts
   The variant attribute collection.

Remarks

It may be more efficient to reuse an existing object rather than allocating a new one.

To view reference material for Visual Basic or C++ only, click the Language
Filter button in the upper-left corner of the page.

See Also

Applies to: IMXAttributes Interface
**setLocalName Method**

Sets the local name of a specific attribute.

[Visual Basic]

**Implementation Syntax**

Sub setLocalName(ByVal nIndex As Long, ByVal strLocalName As String)

**Usage Syntax**

oIMXAttributes.setLocalName(nIndex, strLocalName)

[C/C++]

**Syntax**

HRESULT setLocalName(
    [in] int nIndex,
    [in] BSTR strLocalName);

**Return Values**

S_OK
The value returned if no errors are reported.

E_FAIL
There is only one specific fatal error for this IMXAttributes implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

**Parameters**

nIndex
The attribute's index value (zero-based).

strLocalName
The local name of the attribute or, if there is none, an empty string.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
setQName Method

Sets the qualified name (QName) of a specific attribute.

[Visual Basic]

Implementation Syntax

Sub setQName(ByVal nIndex As Long, ByVal strQName As String)

Usage Syntax

oIMXAttributes.setQName(nIndex, strQName)

[C/C++]

Syntax

HRESULT setQName(
    [in] int nIndex,
    [in] BSTR strQName);

Return Values

S_OK
The value returned if no errors are reported.

E_FAIL
There is only one specific fatal error for this IMXAttributes implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

Parameters

nIndex
The attribute's index value.

strQName
The XML 1.0 qualified name or, if the qualified name is not available, an empty string.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**setType Method**

Sets the type of a specific attribute. This is a user-created type if it has not been declared in the document type definition (DTD).

[Visual Basic]

**Implementation Syntax**

Sub setType(ByVal nIndex As Long, ByVal strType As String)

**Usage Syntax**

oIMXAttributes.setType(nIndex, strType)

[C/C++]

**Syntax**

HRESULT setType(
    [in] int nIndex,
    [in] BSTR strType);

**Return Values**

S_OK
  The value returned if no errors are reported.

E_FAIL
  There is only one specific fatal error for this IMXAttributes implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

**Parameters**

nIndex
  The attribute's index value (zero-based), used to specify the attribute.
strType

A string representing the attribute type.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
setURI Method

Sets the namespace URI of a specific attribute.

[Visual Basic]

Implementation Syntax

Sub setURI(ByVal nIndex As Long, ByVal strURI As String)

Usage Syntax

oIMXAttributes.setURI(nIndex, strURI)

[C/C++]

Syntax

HRESULT setURI(
    [in] int nIndex,
    [in] BSTR strURI);

Return Values

S_OK
    The value returned if no errors are reported.

E_FAIL
    There is only one specific fatal error for this IMXAttributes implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

Parameters

nIndex
    The attribute's index value (zero-based), used to specify the attribute.

strURI
The namespace URI or, if the namespace has no URI, an empty string.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**setValue Method**

Sets the value of a specific attribute.

[Visual Basic]

**Implementation Syntax**

Sub setValue(ByVal nIndex As Long, ByVal strValue As String)

**Usage Syntax**

oIMXAttributes.setValue(nIndex, strValue)

[C/C++]

**Syntax**

HRESULT setValue(
    [in] int nIndex, 
    [in] BSTR strValue);

**Return Values**

S_OK
   The value returned if no errors are reported.

E_FAIL
   There is only one specific fatal error for this IMXAttributes implementation: incorrect index (E_INVALIDARG). It may return a generic failure code, such as out-of-memory.

**Parameters**

* nIndex
   The attribute's index value.

* strValue
A string representing the attribute default value specified by "#IMPLIED", "#REQUIRED", or "#FIXED", or Null if none applies. This value string is represented by the attribute's default value, or Null if there is none.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: IMXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Reference
IMXSchemaDeclHandler Interface

The IMXSchemaDeclHandler interface provides schema information about the element being parsed, and about the attributes of that element.

Methods

- **schemaElementDecl**
  
  Receives the schema element object for an element of the declared type, when it first appears in the document being parsed.

Versioning

MSXML 4.0 and later

Requirements

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib

Remarks

The IMXSchemaDeclHandler interface registers with the SAXXMLReader by using the putProperty method. When you use the putProperty method, specify the "schema-declaration-handler" string identifier as the property name. For the property value, specify an event handler that you used to implement the IMXSchemaDeclHandler interface.

For example:

```c
    oReader.putProperty("schema-declaration-handler", oMyDeclHandler)
```
This property is always read/write. When set, the declared handler (oMyDeclHandler) exposes the type information (i.e. the XSD schema) associated with an XML document.

In order to return type information, you also need to specify a schema. This requires further use of the putProperty method, as in the following example:

```java
oReader.putProperty("schemas", oSchemas)
```

In this example, the value of the schemas property (oSchemas) refers to a new instance of an XMLSchemaCache object.

The schema declaration handler only fires if the SAX validation feature is enabled. For example, the following line demonstrates how to enable validation on the reader:

```java
oReader.putFeature ("schema-validation", TRUE)
```
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
schemaElementDecl Method

Receives the ISchemaElement object for each new element type that is declared and used in the document.

For example, suppose that an XSD schema document declares various element types, including the element type <book>. You configure the SAX reader to register and use this schema when it validates additional documents. As the reader parses the other documents, this method will fire the first time the <book> element appears in each document that is passed as input to the reader.

The same is true of other element types that are declared in the schema and used in a document parsed by the SAX reader.

[Visual Basic]

Implementation Syntax

Sub schemaElementDecl(ByVal oSchemaElement As MSXML2.ISchemaElement)

Parameters

oSchemaElement
    The ISchemaElement object for the element being parsed.

Return Values

If failed, the schemaElementDecl method raises a trappable error.

[C/C++]

Syntax

HRESULT schemaElementDecl(
    [in] ISchemaElement* pSchemaElement);
Parameters

*ISchemaElement [in]*

The ISchemaElement object for the element being parsed.

Return Values

S_OK

The value returned if the ISchemaElement object for the element being parsed is returned successfully.

E_INVALIDARG

The value returned if the argument provided is invalid.

Remarks

The SAX validator feature throws this event immediately prior to the `startElement` event for the current element being parsed. When this method succeeds, it returns an ISchemaElement object. This object remains valid until the end of the corresponding `startElement` event. Accessing this information at any other time produces undefined results.

You can obtain attribute type information by getting the Type object from the returned ISchemaElement object and iterating through it for attribute declarations. For an example, see the [Walk the SOM](/samples) sample application.

Exceptional validation errors might sometimes occur during parsing. For example, suppose that you enable the `exhaustive-errors` feature on the SAX reader. In this situation, an exceptional error is reported if the schema used to validate the document does not contain a type that corresponds to the current element. If this is the case, the `schemaElementDecl` event handler still fires, but it points to an empty element type. When you write your handler procedure, you need to include code to handle this error. For an idea of how to handle element content types, see the [Locate Declarations](/samples) example in the SOM documentation.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
Applies to: IMXSchemaDeclHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
IMXWriter Interface

The IMXWriter interface is a complementary application programming interface (API) to the latest version of the Simple API for XML (SAX2) implementation of Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office. MSXML provides two CoClasses that implement IMXWriter.

<table>
<thead>
<tr>
<th>CoClass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXXMLWriter</td>
<td>Generates XML or DOM output.</td>
</tr>
<tr>
<td>MXHTMLWriter</td>
<td>Generates HTML output.</td>
</tr>
</tbody>
</table>

The following can be set as output for IMXWriter:

- Any implementation of the IStream interface. (The resulting document will be written into the provided IStream.)
- A string.
- A DOMDocument object.

For more information about setting the output property to a DOMDocument object, see Create a DOMDocument Object from SAX Events.

IMXWriter also enables you to:

- control the output of a document by setting properties, such as omitXMLDeclaration, indent, and encoding.
- generate XML or HTML content by invoking methods of the ISAXContentHandler, ISAXDTDHandler, ISAXDeclHandler, and ISAXLexicalHandler interfaces.

Important The IMXWriter interface implements the ISAXContentHandler, ISAXDTDHandler, ISAXErrorHandler, ISAXDeclHandler, and ISAXLexicalHandler interfaces. However, to enable the writer to catch handler events passed by SAXXMLReader, you must set the writer as the ContentHandler, and, optionally, as any of the other handlers. For more information about how to set a writer as a handler, see Use MXXMLWriter.
Ideally, a writer connected to SAXXMLReader produces the same document as provided on the input. However, the writer may miss content deemed ignorable by XML 1.0 specifications or may output in a different encoding. For example, national encoding can be replaced with Unicode Transformation Format (UTF-8 or UTF-16). Replacing the encoding becomes useful if you use a custom SAXXMLFilter between SAXXMLReader and the writer.

IMXWriter uses the generous input approach, which means it outputs what is entered. It is the responsibility of the application to ensure that SAX events and parameters match those of a well-formed or valid XML document.

**Properties**

- **byteOrderMark**
  - Determines whether to write the Byte Order Mark (BOM). The `byteOrderMark` property has no effect for BSTR or DOM output.

- **disableOutputEscaping**
  - Matches the `disable-output-escaping` attribute of the `<xsl:text>` and `<xsl:value-of>` elements. When set to True, special symbols such as `"&"` are passed through literally.

- **encoding**
  - Sets and gets encoding for the output.

- **indent**
  - Sets whether to indent output.

- **omitXMLDeclaration**
  - Forces the IMXWriter to skip the XML declaration. Useful for creating document fragments.

- **output**
  - Sets the destination and the type of output for IMXWriter.

- **standalone**
  - Sets the value of the `standalone` attribute in the XML declaration to "yes" or "no".

- **version**
  - Specifies the version to include in XML declarations.

**Methods**

In addition to the preceding properties, the following method also applies to the IMXWriter interface.

- **flush**
  - Flushes the object's internal buffer to its destination IStream/string. The `flush` method has no effect for
DOM output.

Versioning

MSXML 3.0 and later

Requirements

[Visual Basic, C/C++]

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

Version 5.0 Dependent ProgID: msxml2.MXXMXMLWriter50

Version 4.0 Dependent CLSID: 88d969c8-f192-11d4-a65f-0040963251e5

Version 5.0 Dependent ProgID: msxml2.MXHTMLWriter50

Version 4.0 Dependent CLSID: 88d969c9-f192-11d4-a65f-0040963251e5

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
flush Method

Flushes the object's internal buffer to its destination IStream/string. This flushing occurs when one of the following conditions is met:

- The output property is set to a destination.
- The endDocument() event is called.
- The flush() method is called.

[Visual Basic]

Implementation Syntax

Sub flush()

Usage Syntax

oMXMLWriter.flush

[C/C++]

Syntax

HRESULT flush ( );

Return Values

S_OK
  The value returned if no errors are reported.

Remarks

The flush method has no effect for DOMDocument output. Avoid handling the underlying IStream object during output (for example, direct writing). Keep in mind that a correspondence between Simple API for XML (SAX2) events and output text is not generally specified, although an implementation can provide it.
Therefore, when using the `flush` method, the exact position within the generated document is not generally known. If the `flush` method interrupts writer output, all output that has been processed is put into the underlying `IStream/ISequentialStream` object.

The `flush` method is not implemented by `XSLProcessor` objects.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[output Property](#) | [MXHTMLWriter CoClass](#) | [MXXMLWriter CoClass](#)

Applies to: [IMXWriter Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
byteOrderMark Property

Determines whether to write the Byte Order Mark (BOM) to output.

When set to True, IMXWriter follows XML 1.0 specifications and outputs BOM for appropriate encoding. When set to False, IMXWriter is prohibited from putting BOM into the resulting XML document.

[Visual Basic]

**Implementation Syntax**

Property Let IMXWriter_byteOrderMark(ByVal RHS As Boolean)
Property Get IMXWriter_byteOrderMark() As Boolean

**Usage Syntax**

oMXXMLWriter.byteOrderMark = boolValue
boolValue = oMXXMLWriter.byteOrderMark

**Remarks**

Boolean. Read/write. The byteOrderMark property has no effect for BSTR or DOM output.

[C/C++]

**Syntax**

[propput]
HRESULT byteOrderMark (
    [in] VARIANT_BOOL fwriteByteOrderMark);

[propget]
HRESULT byteOrderMark (
    [out, retval] VARIANT_BOOL * fwriteByteOrderMark);

**Parameters**
**fWriteByteOrderMark**

A Boolean expression (True/False) specifying whether the feature is on or off.

**Return Values**

**S_OK**

The value returned if no errors are reported.

**Remarks**

The default of this property depends on the implementation. *IMXWriter* never outputs BOM if the output is set to a string. The *byteOrderMark* property has no effect for BSTR or DOM output.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td><em>IMXWriter</em> follows XML 1.0 specifications and outputs BOM for encoding.</td>
</tr>
<tr>
<td>False</td>
<td><em>IMXWriter</em> is prohibited from putting BOM into the resulting XML document.</td>
</tr>
</tbody>
</table>

The *XSLProcessor* object implements the *byteOrderMark* property as follows:

- When set, its value overrides the byte-order-mark setting that controls whether a byte-order-mark will precede Unicode encoded output.
- When read, its value is the current byte-order-mark setting. By default, this setting is a variant containing a value of True (VARIANT_TRUE).

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[MXHTMLWriter CoClass](#) | [MXXMLWriter CoClass](#)

Applies to: *IMXWriter Interface*
disableOutputEscaping Property

Matches the disable-output-escaping attribute of the <xsl:text> and <xsl:value-of> elements. For example, normally escaped built-in entities such as "&amp;" for "&" are passed through literally.

[Visual Basic]

Implementation Syntax

Property Let IMXWriter_disableOutputEscaping(ByVal RHS As Boolean)
Property Get IMXWriter_disableOutputEscaping() As Boolean

Usage Syntax

oMXXMLWriter.disableOutputEscaping = boolValue
boolValue = oMXXMLWriter.disableOutputEscaping

Remarks

Boolean. Read/write. The disableOutputEscaping property has no effect for DOM output.

Example

In some cases, you may want to merge an XML document fragment with an XML document. If the document fragment has not been parsed, the "&" character will be escaped as "&amp;" in the document fragment. However, if you merge the document fragment into an XML document, the "&amp;" will be escaped again when it is parsed. To avoid the situation where an already escaped character is escaped again, you can set the disableOutputEscaping property to True. The following code sample manually generates an XML document to demonstrate how this works. For the first element created, disableOutputEscaping is left as False, resulting in a double-escaped character. However, in the second element generated, disableOutputEscaping is set to True, enabling the "GG&amp;G" value to be passed through as intended.
Visual Basic File (Form1.frm)

Private Sub Command1_Click()

    Dim wrt As New MXXMLWriter50
    Dim cnth As IVBSAXContentHandler
    Dim atrs As New SAXAttributes50
    Set cnth = wrt

        'Starts the document by adding the XML declaration.
    cnth.startDocument

        'This section escapes the character.
    cnth.startElement "", "", "company", atrs
    cnth.characters "GG&amp;G"
    cnth.endElement "", "", "company"

        'This section passes the escaped character through as a literal.
    wrt.disableOutputEscaping = True
    cnth.startElement "", "", "company", atrs
    cnth.characters "GG&amp;G"
    cnth.endElement "", "", "company"

        'Ends the document.
    cnth.endDocument

    MsgBox wrt.output

End Sub

Try It!

1. Open Visual Basic and create a new Standard EXE project.
2. Create a reference in the project to Microsoft XML, v5.0.
3. Add a command button (Command1) to the application form (Form1).
4. From the View menu, click Code.
5. Use copy and paste to replace any code in the form with the code provided above.
6. From the Run menu, click Start.
7. In the test application, click Command1 (the command button).

Output

The output in the message box is as follows:

<?xml version="1.0" encoding="UTF-16" standalone="no"?>
Note that the "&" character has been escaped twice in the first <company> element ("GG&amp;amp;amp;G") and passed through as intended (shown in bold) in the second <company> element.

[C/C++]

Syntax

[propput]
HRESULT disableOutputEscaping (  
    [in] VARIANT_BOOL fValue);

[propget]
HRESULT disableOutputEscaping (  
    [out, retval] VARIANT_BOOL * fValue);

Parameters

fValue
    Boolean.

Return Values

S_OK
    The value returned if no errors are reported.

Remarks

If this property is set to True, the XML output may not be well-formed or valid. The disableOutputEscaping property has no effect for DOM output.

The XSLProcessor object does not implement the disableOutputEscaping property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Writer cannot escape input when writing output.</td>
</tr>
</tbody>
</table>
False (default) Writer should escape input when writing output.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

MXHTMLWriter CoClass | MXXMLWriter CoClass

Applies to: IMXWriter Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
encoding Property

Sets and gets encoding for the output.

[Visual Basic]

Implementation Syntax

Property Let IMXWriter_encoding(ByVal RHS As String)
Property Get IMXWriter_encoding() As String

Usage Syntax

oMXXMLWriter.encoding = strValue
strValue = oMXXMLWriter.encoding

Remarks

String. Read/write. The default string is dependent on implementation. Microsoft® Visual Basic® strings are always UTF-16 encoded.

[C/C++]

Syntax

[propput]
HRESULT encoding (  
    [in] BSTR strEncoding);

[propget]
HRESULT encoding (  
    [out, retval] BSTR * strEncoding);

Parameters

strEncoding
  The BSTR used to identify encoding.
Return Values

S_OK
    The value returned if no errors are reported.

Remarks

Read/write. The default string is dependent on implementation.

The encoding property has no effect for BSTR or DOM output.

The XSLProcessor object implements the encoding property as follows:

- When set, its value overrides the encoding that will be used when serializing the XSLT result tree.
- When read, its value is the encoding that will be used to serialize the XSLT result tree. If the style sheet does not specify an encoding, the default encoding of "UTF-16" is returned.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ☐ in the upper-left corner of the page.

See Also

Character Encoding, XML, and MSXML | MXMLWriter CoClass | MXXMLWriter CoClass

Applies to: IMXWriter Interface
indent Property

Sets whether to indent output. When set to True, this property requires output to be indented (or "pretty printed") to reflect the nesting structure of the document.

**Note** The indent property has a special effect on white space processing when writer output is specified as a DOMDocument object. For more information, see Create a DOMDocument Object from SAX Events.

[Visual Basic]

**Implementation Syntax**

Property Let IMXWriter_indent(ByVal RHS As Boolean)  
Property Get IMXWriter_indent() As Boolean

**Usage Syntax**

oMXXMXMLWriter.indent = boolValue  
boolValue = oMXXMXMLWriter.indent

**Remarks**

Read/write. A Boolean expression (True/False) specifying whether the feature is on or off.

[C/C++]

**Syntax**

[propput]  
HRESULT indent (  
    [in] VARIANT_BOOL fIndentMode);

[propget]  
HRESULT indent (  
    [out, retval] VARIANT_BOOL * fIndentMode);
Parameters

fIndentMode
A Boolean expression (True/False) specifying whether the feature is on or off.

Return Values

S_OK
The value returned if no errors are reported.

Remarks

The default of this property depends on the implementation. There is no way to detect white space inside an internal document type definition (DTD). Therefore, DTDs are always indented.

Value Description
True  Provide white space for formatting.
False No additional indenting requirement.

The XSLProcessor object implements the indent property as follows:

- Setting this property at run time is not implemented. To control indentation in XSLT processor output, you must use the <xs1:output> element within a XSLT style sheet assigned to the processor.
- When read, returns the setting (either True or False) that was set using the <xs1:output> element in the XSLT style sheet assigned to the processor. If the output method is HTML, then the default setting is VARIANT_TRUE. If the output method is XML, then the default setting is VARIANT_FALSE. Note that if the output method is unspecified, the processor cannot know until run time what the true setting will be. Therefore, the probable default will be VARIANT_FALSE.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
Applies to: IMXWriter Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**omitXMLDeclaration Property**

Forces the IMXWriter to skip the XML declaration. This can be useful when you want to output fragments of larger XML documents that can be assembled into a single XML document later.

[Visual Basic]

**Implementation Syntax**

```vbnet
Property Let IMXWriter_omitXMLDeclaration(ByVal RHS As Boolean)
Property Get IMXWriter_omitXMLDeclaration() As Boolean
```

**Usage Syntax**

```vb
oMXXMLWriter.omitXMLDeclaration= boolValue
boolValue = oMXXMLWriter.omitXMLDeclaration
```

**Remarks**

Read/write. A Boolean expression (True/False) specifying whether the feature is on or off. The omitXMLDeclaration property has no effect on DOM output.

[C/C++]

**Syntax**

```c
[propput]
HRESULT omitXMLDeclaration (  
    [in] VARIANT_BOOL fValue);

[propget]
HRESULT omitXMLDeclaration (  
    [out, retval] VARIANT_BOOL * fValue);
```

**Parameters**
fValue
  Boolean.

Return Values

S_OK
  The value returned if no errors are reported.

Remarks

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Omit XML declaration.</td>
</tr>
<tr>
<td>False</td>
<td>Provide XML declaration.</td>
</tr>
</tbody>
</table>

The omitXMLDeclaration property has no effect on DOM output.

The IXSLProcessor object implements the omitXMLDeclaration property as follows:

- When set, overrides the omit-xml-declaration setting used in the active XSLT style sheet assigned to the processor. When XML is the output method, this setting controls whether an XML declaration such as <xml version="1.0"?> appears in processor output.
- When read, returns the current omit-xml-declaration setting, either VARIANT_TRUE (True) or VARIANT_FALSE (False). If omit-xml-declaration was not specified in the XSLT style sheet, a value of VARIANT_FALSE (False) is returned.

[Visual Basic]

Example

The following example loads the books.xml file and generates an XML document without the XML declaration.

XML File

Use books.xml.
Visual Basic File (Form1.frm)

Private Sub Command1_Click()
    Dim rdr As New Msxml2.SAXXMLReader50
    Dim cnth As IVBSAXContentHandler
    Dim wrt As New MXXMLWriter50
    wrt.indent = True
    wrt.omitXMLDeclaration = True
    wrt.byteOrderMark = False
    Set cnth = wrt
    Set rdr.contentHandler = cnth
    rdr.parseURL("books.xml")
    MsgBox wrt.output
End Sub

Try It!

1. Go to books.xml. Copy it to a folder on your computer (such as c:\temp).
2. Open Visual Basic and create a new Standard EXE project.
3. Create a reference in the project to Microsoft XML, v5.0.
4. Add a command button (Command1) to the application form (Form1).
5. From the View menu, click Code.
6. Use copy and paste to replace any code in the form with the code provided above.
7. Save the project and form to the folder on your computer where you saved books.xml in Step 1.
8. From the Run menu, click Start.
9. In the test application, click Command1 (the command button).

Output

When run, the sample Visual Basic application should display the books.xml with the XML declaration omitted from the output.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

MXHTMLWriter CoClass | MXXMLWriter CoClass | Sample XML File (books.xml)
Applies to: IMXWriter Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
output Property

Sets the destination and the type of output for IMXWriter. By default, the output property is a BSTR (string) value. Output can be:

- any implementation of the IStream interface. (The resulting document will be written into the provided IStream).
- String.
- DOMDocument object.

For more information about setting the output property to a DOMDocument object, see Create a DOMDocument Object from SAX Events.

Setting this property to the empty value (VT_EMPTY or "") will return output to the internal string and reset it.

[Visual Basic]

Implementation Syntax

Private Property Let IMXWriter_output(ByVal RHS As Variant)
Private Property Get IMXWriter_output() As Variant

Usage Syntax

oMXXMLWriter.output = varDestination
varDestination = oMXXMLWriter.output

Remarks

Read/write. Variant destination.

Example

The following example creates an instance of MXXMLWriter and sets it to the ContentHandler. Method calls on the ContentHandler are used to manually
generate a document. In the sample, a DOMDocument object is created, and the
writer output is set to the DOMDocument object. After the contents of the output
property are loaded into a DOMDocument object, they are then displayed in
message box.
XML File
Use books.xml.
Visual Basic File (Form1.frm)
Private Sub Command1_Click()
Dim wrt As New MXXMLWriter50
Dim cnth As IVBSAXContentHandler
Dim atrs As New SAXAttributes50
Dim xmlDoc As New DOMDocument50
Dim currNode As IXMLDOMNode
Set cnth = wrt
wrt.output = xmlDoc
cnth.startDocument
cnth.startElement "", "", "catalog", atrs
atrs.addAttribute "", "", "id", "CDATA", "bk101"
cnth.startElement "", "", "book", atrs
atrs.Clear
cnth.startElement "", "", "author", atrs
cnth.characters "Gambardella, Matthew"
cnth.endElement "", "", "author"
cnth.startElement "", "", "title", atrs
cnth.characters "XML Developer's Guide"
cnth.endElement "", "", "title"
cnth.startElement "", "", "description", atrs
cnth.characters "An in-depth look at creating applications with XML"
cnth.endElement "", "", "description"
cnth.endElement "", "", "book"
cnth.endElement "", "", "catalog"
'Ends the document.
cnth.endDocument
Set currNode = xmlDoc.documentElement
MsgBox currNode.xml


Try It!

1. Open Visual Basic and create a new **Standard EXE** project.
2. Create a reference in the project to **Microsoft XML, v5.0**.
3. Add a command button (**Command1**) to the application form (**Form1**).
4. From the **View** menu, click **Code**.
5. Use copy and paste to replace any code in the form with the code provided above.
6. From the **Run** menu, click **Start**.
7. In the test application, click **Command1** (the command button).

Output

When run, the sample Visual Basic application should display in a message box the contents of the XML document element (the `<catalog>` element and all its node children) that was created using SAX **ContentHandler** method calls.

The output text should be this:

```xml
<catalog>
    <book id="bk101">
        <author>Gambardella, Matthew</author>
        <title>An in-depth look at creating applications with XML</title>
        <description>An in-depth look at creating applications with XML</description>
    </book>
</catalog>
```

[C/C++]

Syntax

```csharp
[propput]
HRESULT output (  
    [in] VARIANT varDestination);

[propget]
HRESULT output (  
    [out, retval] VARIANT * varDestination);
```

Parameters

```csharp
varDestination
    The variant destination.
```
Return Values

S_OK
   The value returned if no errors are reported.

Remarks

The default is VT_EMPTY to output to string.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

MXHTMLWriter CoClass | MXXMLWriter CoClass

Applies to: IMXWriter Interface
standalone Property

Sets the value of the standalone attribute in the XML declaration to "yes" or "no".

[Visual Basic]

**Implementation Syntax**

```vbnet
Property Let IMXWriter_standalone(ByVal RHS As Boolean)
Property Get IMXWriter_standalone() As Boolean
```

**Usage Syntax**

```vbnet
oMXXMLWriter.standalone = boolValue
boolValue = oMXXMLWriter.standalone
```

**Remarks**

Boolean. Read/write.

**Example**

The following example reads **books.xml** and writes the value of the standalone attribute to "yes" in the XML declaration.

**XML File**

Use **books.xml**.

**Visual Basic File (Form1.frm)**

```vbnet
Private Sub Command1_Click()
    Dim rdr As New Msxml2.SAXXMLReader50
    Dim cnth As IVBSAXContentHandler
    Dim wrt As New MXXMLWriter50
    wrt.indent = True
    wrt.standalone = True
```
Set cnth = wrt
Set rdr.contentHandler = cnth
rdr.parseURL ("books.xml")
MsgBox wrt.output
End Sub

Try It!

1. Go to books.xml. Copy it to a folder on your computer (such as c:\temp).
2. Open Visual Basic and create a new Standard EXE project.
3. Create a reference in the project to Microsoft XML, v5.0.
4. Add a command button (Command1) to the application form (Form1).
5. From the View menu, click Code.
6. Use copy and paste to replace any code in the form with the code provided above.
7. Save the project and form to the folder on your computer where you saved books.xml in Step 1.
8. From the Run menu, click Start.
9. In the test application, click Command1 (the command button).

Output

When run, the sample Visual Basic application should display the contents of books.xml in a message box. The file XML declaration should have its standalone attribute set to "yes", like this:

<?xml version="1.0" encoding="UTF-16" standalone="yes"?>

[C/C++]

Syntax

[propput]
HRESULT standalone (  
    [in] VARIANT_BOOL fValue);

[propget]
HRESULT standalone (  
    [out, retval] VARIANT_BOOL * fValue);

Parameters
**fValue**

Boolean.

**Return Values**

S_OK

The value returned if no errors are reported.

**Remarks**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Sets the value of the standalone attribute to &quot;yes&quot;.</td>
</tr>
<tr>
<td>False (default)</td>
<td>Sets the value of the standalone attribute to &quot;no&quot;.</td>
</tr>
</tbody>
</table>

The XSLProcessor object implements the standalone property as follows:

- Setting this property at run time is not implemented.
- When read, this property returns the current value (if specified by the user) for `<xs1:output>` in the XSLT style sheet used by the processor. Otherwise, a default value of VARIANT_FALSE is returned.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[MXHTMLWriter CoClass](#) | [MXXMLWriter CoClass](#) | [Sample XML File (books.xml)](#)

Applies to: [IMXWriter Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
version Property

Specifies the version to include in XML declarations, for example, <?xml version="1.0">.

[Visual Basic]

Implementation Syntax

Property Let IMXWriter_version(ByVal RHS As String)
Property Get IMXWriter_version() As String

Usage Syntax

oMXXMLWriter.version = strVersion
strVersion = oMXXMLWriter.version

[C/C++]

Syntax

[propput]
HRESULT version ( 
    [in] BSTR strVersion);

[propget]
HRESULT version ( 
    [out, retval] BSTR * strVersion);

Parameters

strVersion
    The BSTR of version.

Return Values

S_OK
    The value returned if no errors are reported.
Remarks

Read/write. A string representing the version. The default value is "1.0". The XSLProcessor object does not implement setting of this property at run time.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

MXHTMLWriter CoClass | MXXMLWriter CoClass

Applies to: IMXWriter Interface
MXHTMLWriter CoClass

The `MXHTMLWriter` CoClass generates HTML output from Simple API for XML (SAX) events and handles many of the details of outputting content in HTML format. When connected to `SAXXMLReader`, and set as a `ContentHandler`, `MXHTMLWriter` accumulates content passed by events thrown by the reader. This content is stored in the output buffer of `MXHTMLWriter`. The `MXXMLWriter` CoClass implements the `IMXWriter` interface.

For MSXML 5.0, the version-dependent CoClass is `MXHTMLWriter50`, which uses the following programming class ID information:

(CLSID_MXHTMLWriter: 88d969c9-f192-11d4-a65f-0040963251e5)

Results generated by `MXHTMLWriter` are exposed through the `MXHTMLWriter.output` property. `MXHTMLWriter` provides several properties that enable you to modify the results generated by it. For example, to enable "pretty printing," set the `MXHTMLWriter.indent` property to True.

To enable `MXHTMLWriter` to catch handler events passed by `SAXXMLReader`, you must set `MXHTMLWriter` as the `ContentHandler`, and optionally as any of the other handlers. The following table shows the handler interfaces implemented by `MXHTMLWriter`.

<table>
<thead>
<tr>
<th>Visual Basic Interfaces</th>
<th>C++ Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVBSAXContentHandler</td>
<td>ISAXContentHandler</td>
</tr>
<tr>
<td>IVBSAXDeclHandler</td>
<td>ISAXDeclHandler</td>
</tr>
<tr>
<td>IVBSAXDTDHandler</td>
<td>ISAXDTDHandler</td>
</tr>
<tr>
<td>IVBSAXErrorHandler</td>
<td>ISAXErrorHandler</td>
</tr>
<tr>
<td>IVBSAXLexicalHandler</td>
<td>ISAXLexicalHandler</td>
</tr>
</tbody>
</table>

With `MXHTMLWriter`, HTML is output according to the HTML output method, as defined in the XSL Transformations (XSLT) Version 1.0 recommendation.

In general, `MXHTMLWriter` outputs HTML content like XML, with the following exceptions:
The following elements are recognized as empty elements. They are recognized in any combination of uppercase and lowercase. These elements are output with a start tag but no end tag.

```html
<area>  <frame>  <links>
;base>  <hr>  <meta>
;basefont>  <img>  <param>
<br>  <input>
<col>  <isindex>
```

- The `<script>` and `<style>` elements do not require escaping of special characters. In the textual content of these elements, a "<" character will be output as "<", not as "&lt;".
- HTML attributes that have a URI value, such as the `href` attribute of the `<a>` element or the `src` attribute of the `<image>` element, are recognized. Special characters within a URI are escaped as specified in the HTML specification. For example, non-ASCII characters in the URI should be represented by converting each byte of the UTF-8 representation "%hh" where `hh` represents the byte value in hexadecimal. For example, spaces are represented as "%20".
- `MXHTMLWriter` may output a character using a character entity reference if one is defined for it in the version of HTML that is specified.
- Processing instructions are terminated with "">
  rather than "?".
- Attributes that are written with a keyword only, and no value, will be recognized and output in this form. For example, in HTML, `<OPTION SELECTED>` shorthand is acceptable, but in XML it must be written as `<OPTION SELECTED="SELECTED">`. The HTML output method will normally use the abbreviated form, as this is the only form recognized by earlier HTML browsers.
- The special use of the ampersand character in dynamic HTML attributes is recognized. For example, although the tag `<TD HEIGHT=&{height};>` is valid HTML, it would not be valid XML because of the ampersand character.

**Properties**

`byteOrderMark` Determines whether or not to write the Byte Order Mark (BOM).
**disableOutputEscaping**
Determine whether disableOutputEscaping is on or off. When set to True, special symbols such as "&" are passed through literally.

**encoding**
Sets encoding for the output.

**indent**
Sets whether to indent output. Useful for "pretty printing."

**omitXMLDeclaration**
Not supported by MXHTMLWriter.

**output**
Sets the destination for the output generated by MXHTMLWriter.

**standalone**
Not supported by MXHTMLWriter.

**version**
Not supported by MXHTMLWriter.

**Methods**

**flush**
Flushes the object's internal buffer to its destination IStream/string.

**Requirements**

[Visual Basic, C/C++]

**Implementation:** msxml5.dll, msxml2.lib

**Header and LIB files:** msxml2.h, msxml2.lib

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**Other Resources**

W3C XSL Transformations (XSLT) Version 1.0 Recommendation
flush Method

Flushes the object's internal buffer to its destination IStream/string. This flushing occurs when one of the following conditions is met:

- The output property is set to a destination.
- The endDocument() event is called.
- The flush() method is called.

[Visual Basic]

Usage Syntax

oMXHTMLWriter.flush

[C/C++]

Syntax

HRESULT flush ( );

Return Values

S_OK

Remarks

Avoid handling the underlying IStream object during output (for example, direct writing). When using the flush method, the exact position within the generated document is not generally known. If the flush method is called, all output that has been processed is put into the underlying IStream/ISequentialStream object.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

output Property | IMXWriter Interface | MXXMLWriter CoClass

Applies to: MXHTMLWriter CoClass
byteOrderMark Property

Determines whether to write the Byte Order Mark (BOM) to the MXHTMLWriter output. Setting the byteOrderMark property to False prohibits the MXHTMLWriter from putting the Byte Order Mark (BOM) into the resulting HTML document.

[Visual Basic]

Usage Syntax

```
OMXMLWriter.byteOrderMark = boolValue
boolValue = OMXMLWriter.byteOrderMark
```

Remarks

Boolean. Read/write. The default is True. MXHTMLWriter never generates a BOM if the output is set to a string.

[C/C++]

Syntax

```
[propput]
HRESULT byteOrderMark (  
    [in] VARIANT_BOOL fWriteByteOrderMark);

[propget]
HRESULT byteOrderMark (  
    [out, retval] VARIANT_BOOL * fWriteByteOrderMark);
```

Parameters

fWriteByteOrderMark
A Boolean expression (True/False) specifying whether the feature is on or off.

Return Values
**S_OK**

The value returned if no errors are reported.

**Remarks**

The default is True. **MXMLWriter** never outputs BOM if the output is set to a string.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td><strong>MXMLWriter</strong> follows HTML 4.0 specifications and outputs BOM for encoding.</td>
</tr>
<tr>
<td>False</td>
<td><strong>MXMLWriter</strong> does not write the BOM into the resulting HTML output.</td>
</tr>
</tbody>
</table>

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[IMXWriter Interface](#) | [MXMLWriter CoClass](#)

Applies to: **MXMLWriter CoClass**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
disableOutputEscaping Property

Specifies whether the disableOutputEscaping is on or off. When set to True, special symbols, such as "&", are passed through literally. When disableOutputEscaping is set to False, special symbols, (also known as built-in entities), such as "&", are escaped to &amp;.

[Visual Basic]

Usage Syntax

oMXXMLWriter.disableOutputEscaping = boolValue
boolValue = oMXXMLWriter.disableOutputEscaping

Remarks

Boolean. Read/write. The default value is False. Normally, when you try to output a special character such as "<" or "&", the special character will be escaped in the output file using the standard XML escaping mechanism. For example, by default, MXHTMLWriter writes the "&" symbol as "&amp;". If disableOutputEscaping is set to True, the XML parser does not escape the special symbol and writes the "&" symbol as the literal "&".

Example

For elements such as <script> and <style>, MXHTMLWriter passes through special characters. For others, you may want to set disableOutputEscaping property to True, as shown in the following example.

Dim wrt As New MXHTMLWriter
Dim cnth As IVBSAXContentHandler
Dim atrs As New SAXAttributes
Set cnth = wrt

cnth.startDocument

cnth.startElement "", "", "html", atrs
cnth.startElement "", "", "script", atrs
cnth.characters "var a = 0;"
Example Output

The preceding example code passes through the "<" character as a literal within the <script> tags. However, to pass through the "&" character in the company name GG&G, you must set the disableOutputEscaping property to True. If disableOutputEscaping is set to False, the "&" character is escaped as &amp;.

<html><script>var a = 0; var b = 1; if (a < b) alert('hello')</script>
<body>GG&G</body>
</html>

[C/C++]

Syntax

[propput]
HRESULT disableOutputEscaping (  
    [in] VARIANT_BOOL fValue);

[propput]
HRESULT disableOutputEscaping (  
    [out, retval] VARIANT_BOOL * fValue);

Parameters

fValue
A Boolean expression (True/False) specifying whether the feature is on or off.

Return Values
S_OK
The value returned if no errors are reported.

Remarks
If this property is set to True, the XML output may not be well-formed or valid.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td><strong>MXXMLWriter</strong> does not escape special symbols (passing them through as literals) when writing output.</td>
</tr>
<tr>
<td>False (default)</td>
<td><strong>MXXMLWriter</strong> escapes special symbols when writing output.</td>
</tr>
</tbody>
</table>

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

[IMXWriter Interface](#) | [MXXMLWriter CoClass](#)

Applies to: [MXHTMLWriter CoClass](#)
encoding Property

Allows you to set the encoding according to HTML 4.0 specifications. MXHTMLWriter supports the same encoding as the Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office DOM. This parameter is ignored for output to a BSTR, but will take effect if output is changed in the middle of a document to a destination other than BSTR.

[Visual Basic]

Usage Syntax

oMXXMLWriter.encoding = strValue
strValue = oMXXMLWriter.encoding

Remarks

String. This property is read/write. The default is UTF-16. If your output is a string, the encoding parameter is ignored.

[C/C++]

Syntax

[propput]
HRESULT encoding ( 
    [in] BSTR strEncoding);

[proppget]
HRESULT encoding ( 
    [out, retval] BSTR * strEncoding);

Parameters

strEncoding
    The BSTR used to identify encoding.
Return Values

S_OK
The value returned if no errors are reported.

Remarks

The default string is UTF-16. If your output is a string, the encoding parameter is ignored.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Character Encoding, XML, and MSXML | IMXWriter Interface | MXXMLWriter CoClass

Applies to: MXHTMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
indent Property

Sets whether to indent output. When set to True, this property requires output to be indented (or "pretty printed") to reflect the nesting structure of the document.

[Visual Basic]

Usage Syntax

```
oMXXMLWriter.indent = boolValue
boolValue = oMXXMLWriter.indent
```

Remarks

Boolean. Read/write. The general rules of indentation are: XML headers and root elements start on a new line with a zero indent; element content, including leading and trailing white spaces, is not changed in any way.

Example

The following example creates a simple HTML document and indents it for "pretty-printing."

```
Dim wrt As New MXHTMLWriter50
Dim cnth As IVBSAXContentHandler
Dim atrs As New SAXAttributes50
Set cnth = wrt
wrt.indent = True
cnth.startDocument
cnth.startElement "", "", "html", atrs
cnth.startElement "", "", "body", atrs
cnth.characters "This is a test"
cnth.endElement "", "", "body"
cnth.endElement "", "", "html"
MsgBox wrt.output
```

Results

```
<html>
```
[C/C++]

Syntax

[propput]
HRESULT indent (  
    [in] VARIANT_BOOL fIndentMode);

[propget]
HRESULT indent (  
    [out, retval] VARIANT_BOOL * fIndentMode);

Parameters

fIndentMode
    A Boolean expression (True/False) specifying whether the feature is on or off.

Return Values

S_OK
    The value returned if no errors are reported.

Remarks

The default of this property depends on the implementation. The general rules of indentation are: XML headers and root elements start on a new line with a zero indent; element content, including leading and trailing white spaces, is not changed in any way.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

IMXWriter Interface | MXXMLWriter CoClass
Applies to: MXHTMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
omitXMLDeclaration Property

This property is not supported by MXHTMLWriter.
**output Property**

Determines the output for `MXXMLWriter`. By default, the output property is a BSTR (string) value. You can also set this property to any implementation of the IStream interface, and the resulting document will be written into the provided IStream. Setting this property to the empty value (VT_EMPTY or "") will return output to the internal string and reset it.

[Visual Basic]

**Usage Syntax**

```vbs
oMXXMLWriter.output = varDestination
varDestination = oMXXMLWriter.output
```

**Remarks**

Read/write. Variant destination.

**Example**

For elements such as `<script>` and `<style>`, MXHTMLWriter passes through special characters. In the following example, MXHTMLWriter passes through the "<" character as a literal within the `<script>` tags.

```vbs
Dim wrt As New MXHTMLWriter50
Dim cnth As IVBSAXContentHandler
Dim atrs As New SAXAttributes50
Set cnth = wrt
cnth.startDocument
cnth.startElement "", "", "html", atrs
cnth.startElement "", "", "script", atrs
cnth.characters "var a = 0;"
cnth.characters "var b = 1;"
cnth.characters "if (a < b) alert('hello')"
cnth.endElement "", "", "script"
cnth.startElement "", "", "body", atrs
wrt.disableOutputEscaping = True
cnth.characters "Sample HTML Page"
```
wrt.disableOutputEscaping = False
cnth.endElement "", "", "body"
cnth.endElement "", "", "html"
MsgBox wrt.output

Example Output

<html><script>var a = 0;var b = 1;if (a < b)
alert('hello')</script><body>Sample HTML Page</body></html>

[C/C++]

Syntax

[propput]
HRESULT output (  
    [in] VARIANT varDestination);

[propget]
HRESULT output (  
    [out, retval] VARIANT * varDestination);

Parameters

varDestination
    The variant destination.

Return Values

S_OK
    The value returned if no errors are reported.

Remarks

The default is VT_EMPTY to output to string.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
IMXWriter Interface | MXXMXMLWriter CoClass

Applies to: MXHTMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
standalone Property

This property is not supported by MXHTMLWriter.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
version Property

This property is not supported by MXHTMLWriter.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
MXXMLWriter CoClass

MXXMLWriter generates XML output that can be:

- Any implementation of the IStream interface. (The resulting document will be written into the provided IStream.)
- A string.
- A DOMDocument object. For more information about setting the output property to a DOMDocument object, see Create a DOMDocument Object from SAX Events

When connected to SAXXMLReader, and set as a ContentHandler, MXXMLWriter accumulates XML content passed by events thrown by the reader. This content is stored in the MXXMLWriter's output buffer. The MXXMLWriter CoClass implements the IMXWriter interface.

For MSXML 5.0, the version-dependent CoClass is MXMLWriter50, which uses the following programming class ID information:

(CLSID_MXXMLWriter50: 8d969c8-f192-11d4-a65f-0040963251e5)

In addition to generating XML, MXXMLWriter also handles the details of collecting data produced by SAX events. For example, rather than concatenating strings within each method call of the ContentHandler, and dealing with issues such as the manual insertion of opening and closing brackets for startElement and EndElement events respectively, you can set MXXMLWriter to the IVBSAXContentHandler\ISAXContentHandler interface, and then make method calls on the ContentHandler to push data to MXXMLWriter. MXXMLWriter then writes opening brackets for the startElement event and closing brackets for the endElement event.

Results generated by MXXMLWriter are exposed through the MXXMLWriter.output property. MXXMLWriter provides several properties that enable you to modify output collected by MXXMLWriter. For example, to create an XML document fragment, set the MXXMLWriter.omitXMLDeclaration to True. In a similar way, you can set the MXXMLWriter.indent to True to enable "pretty printing."
To enable `MXXMLWriter` to catch handler events passed by `SAXXMLReader`, you set `MXXMLWriter` as the `ContentHandler`, and optionally as any of the other handlers. For more information about how to register `MXXMLWriter` as a handler, see `Use MXXMLWriter`.

The following table shows the handler interfaces implemented by `MXXMLWriter`.

<table>
<thead>
<tr>
<th>Visual Basic Interfaces</th>
<th>C++ Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVBSAXContentHandler</td>
<td>ISAXContentHandler</td>
</tr>
<tr>
<td>IVBSAXDeclHandler</td>
<td>ISAXDeclHandler</td>
</tr>
<tr>
<td>IVBSAXDTDHandler</td>
<td>ISAXDTDHandler</td>
</tr>
<tr>
<td>IVBSAXErrorHandler</td>
<td>ISAXErrorHandler</td>
</tr>
<tr>
<td>IVBSAXLexicalHandler</td>
<td>ISAXLexicalHandler</td>
</tr>
</tbody>
</table>

**Properties**

- **byteOrderMark**
  - Determines whether to write the Byte Order Mark (BOM).
  - Turns `disableOutputEscaping` on or off. When `disableOutputEscaping` is set to True, special symbols such as "&" are passed through literally.

- **disableOutputEscaping**
  - Sets and gets encoding for the output.

- **encoding**
  - Sets whether to indent output. Useful for "pretty printing."

- **indent**
  - Forces the `MXXMLWriter` to omit the XML declaration. Useful for creating document fragments.

- **omitXMLDeclaration**
  - Determines the destination and the type of output for `MXXMLWriter`. Output can be stream, string, or DOM.

- **output**
  - Sets the value of the `standalone` attribute in the XML declaration to "yes" or "no".

- **standalone**
  - Specifies the version to include in XML declaration.

- **version**

**Methods**

- **flush**
  - Flushes the object's internal buffer to its destination `IStream/string`. 
Requirements

[Visual Basic]

**Implementation:** msxml5.dll, msxml2.lib

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
flush Method

Flushes the object's internal buffer to its destination IStream/string. This flushing occurs when one of the following conditions is met:

- The output property is set to a destination.
- The endDocument() event is called.
- The flush() method is called.

[Visual Basic]

Usage Syntax

oMXXMLWriter.flush

Remarks

Avoid handling the underlying IStream object during output (for example, direct writing). When using the flush method, the exact position within the generated document is not generally known. If the flush method is called, all output that has been processed is put into the underlying IStream/ISequentialStream object.

[C/C++]

Syntax

HRESULT flush ( );

Return Values

S_OK

Remarks
Avoid handling the underlying \texttt{IStream} object during output (for example, direct writing). When using the \texttt{flush} method, the exact position within the generated document is not generally known. If the \texttt{flush} method interrupts \texttt{MXXMLWriter} output, all output that has been processed is put into the underlying \texttt{IStream/ISequentialStream} object.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

\textbf{See Also}

\texttt{output Property}

Applies to: \texttt{MXXMLWriter CoClass}
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
byteOrderMark Property

Determines whether to write the Byte Order Mark (BOM) to MXXMLWriter output. Setting the byteOrderMark property to False prohibits the writer from putting the Byte Order Mark (BOM) into the resulting XML document or document fragment. This property is useful for creating document fragments. Setting this property to True makes MXXMLWriter follow XML 1.0 specifications and output the Byte Order Mark for appropriate encoding.

[Visual Basic]

Usage Syntax

```
oMXXMLWriter.byteOrderMark = boolValue
boolValue = oMXXMLWriter.byteOrderMark
```

Remarks

Boolean. Read/write. The default is True. MXXMLWriter never generates a BOM if the output is set to a string.

[C/C++]

Syntax

```
[proput]
HRESULT byteOrderMark (  
    [in] VARIANT_BOOL fwriteByteOrderMark);
```

```
[propget]
HRESULT byteOrderMark (  
    [out, retval] VARIANT_BOOL * fwriteByteOrderMark);
```

Parameters

fwriteByteOrderMark
    A Boolean expression (True/False) specifying whether the feature is on or
Return Values

S_OK

The value returned if no errors are reported.

Remarks

The default of this is True. MXXMLWriter never outputs BOM if the output is set to a string.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>MXXMLWriter follows XML 1.0 specifications and outputs BOM for encoding.</td>
</tr>
<tr>
<td>False</td>
<td>MXXMLWriter does not write the BOM into the resulting XML output.</td>
</tr>
</tbody>
</table>

Example

In some cases, you may want to merge an XML document fragment with an XML document. If the document fragment has not been parsed, the "&" character will be escaped as "&amp;" in the document fragment. However, if you merge the document fragment into an XML document, the "&amp;" will be escaped again when it is parsed. To avoid the situation in which an already escaped character is escaped again, set the disableOutputEscaping property to True. The following code sample manually generates an XML document to demonstrate how this works. For the first element created, disableOutputEscaping is left as False, resulting in a double-escaped character. However, in the second element generated, disableOutputEscaping is set to True, enabling the "GG&amp;G" value to be passed through as intended.

```vbscript
Dim wrt As New MXXMLWriter50
Dim cnth As IVBSAXContentHandler
Dim atrs As New SAXAttributes50
Set cnth = wrt

'Starts the document by adding the XML declaration.
```
'This section escapes the character.
cnth.startElement "", "", "company", atrs
cnth.characters "GG&amp;G"
cnth.endElement "", "", "company"

'This section passes the escaped character through as a literal.
wrt.disableOutputEscaping = True
cnth.startElement "", "", "company", atrs
cnth.characters "GG&amp;G"
cnth.endElement "", "", "company"

'Ends the document.
cnth.endDocument

'Sets the writer output to the TextResult.Text box.
MsgBox wrt.output

**Example Output**

The output in the message box is as follows. Notice that the "&" character has been escaped twice in the first <company> element and passed through as intended (shown in bold) in the second <company> element.

```xml
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<company>GG&amp;amp;G</company>

<company>GG&amp;G</company>
```

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [MXXMLWriter CoClass](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
disableOutputEscaping Property

Turns disableOutputEscaping on or off. When disableOutputEscaping is set to True, special symbols such as "&" are passed through literally. When disableOutputEscaping is set to False, special symbols (also known as built-in entities) such as & are escaped to &.

[Visual Basic]

Usage Syntax

oMXXMLWriter.disableOutputEscaping = boolValue
boolValue = oMXXMLWriter.disableOutputEscaping

Remarks

Read/write. The default value is False. When you try to output a special character such as "<" or "&", the special character will be escaped in the output file using the standard XML escaping mechanism. For example, by default, the MXXMLWriter writes the "&" symbol as "&amp;" unless disableOutputEscaping is set to True. If disableOutputEscaping is set to True, MXXMLWriter does not escape the special symbol and writes the "&" symbol as the literal "&".

[C/C++]

Syntax

[propput]
HRESULT disableOutputEscaping (  
    [in] VARIANT_BOOL fValue);

[propget]
HRESULT disableOutputEscaping (  
    [out, retval] VARIANT_BOOL * fValue);

Parameters
**fValue**

A Boolean expression (True/False) specifying whether the feature is on or off.

**Return Values**

S_OK

The value returned if no errors are reported.

**Remarks**

If this property is set to True, the XML output may not be well-formed or valid.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td><strong>MXXMLWriter</strong> does not escape special symbols (passing them through as literals) when writing output.</td>
</tr>
<tr>
<td>False (default)</td>
<td><strong>MXXMLWriter</strong> escapes special symbols when writing output.</td>
</tr>
</tbody>
</table>

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: **MXXMLWriter CoClass**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
encoding Property

Sets and gets encoding for the output.

[Visual Basic]

Usage Syntax

```plaintext
oMXXMLWriter.encoding = strValue
strValue = oMXXMLWriter.encoding
```

Remarks

String. Read/write. The default string is empty, that is, UTF-16. Microsoft® Visual Basic® strings are always UTF-16 encoded. If your output is a string, the encoding parameter is ignored.

[C/C++]

Syntax

```plaintext
[propput]
HRESULT encoding (
    [in] BSTR strEncoding);

[propget]
HRESULT encoding (
    [out, retval] BSTR * strEncoding);
```

Parameters

strEncoding
    The BSTR used to identify encoding.

Return Values

S_OK
The value returned if no errors are reported.

Remarks

The default string is empty, that is, UTF-16.

If your output is a string, the encoding parameter is ignored.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

[Character Encoding, XML, and MSXML](#)

Applies to: [MXXMWriter CoClass](#)
indent Property

Sets whether to indent output. When set to True, this property indents output for "pretty printing" to reflect the nesting structure of the document.

[Visual Basic]

Usage Syntax

```vbs
oMXXMLWriter.indent = boolValue
boolValue = oMXXMLWriter.indent
```

Remarks

Boolean. Read/write. The default is False. The general rules of indenting elements are: XML headers and root elements start on a new line with a zero indent; element content, including leading and trailing white spaces, is not changed in any way.

Example

The following example parses books.xml and indents the output.

```vbs
Dim rdr As New Msxml2.SAXXMLReader50
Dim cnth As IVBSAXContentHandler
Dim wrt As New MXXMLWriter50
wrt.indent = True
wrt.standalone = True
Set cnth = wrt
Set rdr.contentHandler = cnth
rdr.parseURL ("books.xml")
MsgBox wrt.output
```

Results

The following shows the beginning of the `books.xml` output with the elements indented for "pretty printing".

```xml
<?xml version="1.0" encoding="UTF-16" standalone="yes"?><
```
[C/C++]

Syntax

[propput]
HRESULT indent (    
    [in] VARIANT_BOOL fIndentMode);

[propget]
HRESULT indent (    
    [out, retval] VARIANT_BOOL * fIndentMode);

Parameters

fIndentMode
A Boolean expression (True/False) specifying whether the feature is on or off.

Return Values

S_OK
The value returned if no errors are reported.

Remarks

The default of this property is False. Because there is no way to detect white space inside an internal document type definition (DTD), they are always indented.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Provide white space for formatting.</td>
</tr>
</tbody>
</table>
False

No additional indenting requirement.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Sample XML File (books.xml)

Applies to: MXXMLWriter CoClass
omitXMLDeclaration Property

Forces the `MXXMLWriter` to omit the XML declaration. This property can be used to output fragments of larger XML documents that can be assembled into a single XML document later.

[Visual Basic]

Usage Syntax

```vbnet
oMXXMLWriter.omitXMLDeclaration = boolValue
boolValue = oMXXMLWriter.omitXMLDeclaration
```

Remarks

Boolean. Read/write.

Example

The following example parses `books.xml` and generates an XML document without the XML declaration.

```vbnet
Dim rdr As New Msxml2.SAXXMLReader50
Dim cnth As IVBSAXContentHandler
Dim wrt As New MXXMLWriter50
wrt.indent = True
wrt.omitXMLDeclaration = True
wrt.byteOrderMark = False
Set cnth = wrt
Set rdr.contentHandler = cnth
rdr.parseURL("books.xml")
MsgBox wrt.output
```

Results

The following shows a fragment of `books.xml` with the XML declaration omitted from the output.

```xml
<catalog>
```
[C/C++]

**Syntax**

```c
[propput]
HRESULT omitXMLDeclaration (  
    [in] VARIANT_BOOL fValue);

[propget]
HRESULT omitXMLDeclaration (  
    [out, retval] VARIANT_BOOL * fValue);
```

**Parameters**

*fValue*

Boolean. Read/write.

**Return Values**

*S_OK*

The value returned if no errors are reported.

**Remarks**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Omit XML declaration.</td>
</tr>
<tr>
<td>False (default)</td>
<td>Provide XML declaration.</td>
</tr>
</tbody>
</table>

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.
See Also

Sample XML File (books.xml)

Applies to: MXXXMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
output Property

Determines the destination and the type of output for MXXMLWriter. By default, the output property is a BSTR (string) value.

Output for IMXWriter can be set as one of the following:

- Any implementation of the IStream interface. (The resulting document will be written into the provided IStream.)
- A string.
- A DOMDocument object.

For more information about setting the output property to a DOMDocument object, see Create a DOMDocument Object from SAX Events

Setting the output property to the empty value (VT_EMPTY or "") will return output to the internal string and reset it.

[Visual Basic]

Usage Syntax

```vbs
oMXXMLWriter.output = varDestination
varDestination = oMXXMLWriter.output
```

Remarks

Read/write. Variant destination.

Example

The following example creates an instance of MXXMLWriter and sets it to the ContentHandler. Method calls on the ContentHandler are used to manually generate a document. In the sample, a DOMDocument object is created, and the writer output is set to the DOMDocument object. The example code produces three message boxes. The first box shows the DOMDocument object as it is being built
(For example, <catalog><book id="bk101"/></catalog>). The second message box shows the finished document. The third message box shows the text in the first <author> tag, Gambardella, Matthew.

Dim xmlDoc As New Msxml2.DOMDocument50
Dim nodeList As IXMLDOMNodeList
Dim wrt As New MXXMWriter50
Dim cnth As IVBSAXContentHandler
Dim atrs As New SAXAttributes50

Set cnth = wrt
wrt.output = xmlDoc

' sets standalone="yes" in the XML declaration
' wrt.standalone = True
' configures the writer to indent elements
wrt.indent = True

'starts the document by adding the XML declaration
cnth.startDocument

' adds the <catalog> element to the page
cnth.startElement "", "", "catalog", atrs

' adds the id attribute to the collection with the "bk0101" value.
atrs.addAttribute "", "", "id", "CDATA", "bk101"
' creates the <book id="bk101"> tag
cnth.startElement "", "", "book", atrs
' clears the attribute collection
atrs.Clear

MsgBox xmlDoc.xml

' creates the <author>Gambardella, Matthew</author> string
cnth.startElement "", "", "author", atrs
cnth.characters "Gambardella, Matthew"
cnth.endElement "", "", "author"

' creates the <title>XML Developer's Guide</title> string
cnth.startElement "", "", "title", atrs
cnth.characters "XML Developer's Guide"
cnth.endElement "", "", "title"

' creates the <description>An in-depth look at...</description> string
cnth.startElement "", "", "description", atrs
cnth.characters "An in-depth look at creating applications with XML"
cnth.endElement "", "", "description"
'adds closing tags for <book> and <catalog> elements
ctn.endElement "", "", "book"
ctn.endElement "", "", "catalog"

'ends the document
ctn.endDocument

'displays the author's name in a message box
MsgBox xmlDoc.xml
Set objNodeList = xmlDoc.getElementsByTagName("author")
MsgBox objNodeList.Item(0).Text

[C/C++]

Syntax

[propput]
HRESULT output (  
    [in] VARIANT varDestination);

[propget]
HRESULT output (  
    [out, retval] VARIANT * varDestination);

Parameters

varDestination
    The variant destination.

Return Values

S_OK
    The value returned if no errors are reported.

Remarks

The default is output to string.

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
Create a DOMDocument Object from SAX Events | IMXWriter Interface | MXHTMLWriter CoClass

Applies to: MXXMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**standalone Property**

Sets the value of the standalone attribute in the XML declaration to "yes" or "no". If set to True, the XML declaration appears as the following in XML output:

```xml
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
```

[Visual Basic]

**Usage Syntax**

```vba
oMXXMXMLWriter.standalone = boolValue
boolValue = oMXXMXMLWriter.standalone
```

**Remarks**

Boolean. Read/write. The default value is False. The standalone declaration indicates if a document relies on information from an external source, such as external document type definition (DTD), for its content.

If set to False, the XML parser will accept external resources, if there are any, without reporting an error. If set to True, the parser reports an error if the document references an external DTD or external entities.

**Example**

The following example reads **books.xml** and writes the value of the standalone attribute to "yes" in the XML declaration.

```vba
Dim rdr As New Msxml2.SAXXMLReader50
Dim cnth As IVBSAXContentHandler
Dim wrt As New MXXMXMLWriter50
wrt.indent = True
wrt.standalone = True
Set cnth = wrt
Set rdr.contentHandler = cnth
rdr.parseURL ("books.xml")
MsgBox wrt.output
```
[C/C++]

Syntax

[propget]
HRESULT standalone (
    [in] VARIANT_BOOL fValue);

[propput]
HRESULT standalone (
    [out, retval] VARIANT_BOOL *fValue);

Parameters

fValue
A Boolean expression (True/False) specifying whether the feature is on or off.

Return Values

S_OK
The value returned if no errors are reported.

Remarks

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The value of standalone is set to &quot;yes&quot;.</td>
</tr>
<tr>
<td>False (default)</td>
<td>The value of standalone is set to &quot;no&quot;.</td>
</tr>
</tbody>
</table>

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: MXXMXMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Reference
**version Property**

Specifies the version to include in the XML declaration. The `version` property value defaults to "1.0". As of this writing, there have been no plans for any other version of the XML specification. However, if another version is created in the future, the version number in the XML specification will be used to identify which version of the specification the document supports.

[Visual Basic]

**Usage Syntax**

```
oMXXMLWriter.version = strVersion
strVersion = oMXXMLWriter.version
```

[C/C++]

**Syntax**

```
[propput]
HRESULT version (    
    [in] BSTR strVersion);

[propget]
HRESULT version (    
    [out, retval] BSTR * strVersion);
```

**Parameters**

*strVersion*

  The BSTR of version.

**Return Values**

*S_OK*

  The value returned if no errors are reported.
Remarks

Read/write. A string representing the version. The default value is "1.0".

To view reference material for Visual Basic or C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: MXMLWriter CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXAttributes Interface

The ISAXAttributes/IVBSAXAttributes interface allows access to a list of attributes by:

- attribute index
- namespace-qualified name
- qualified (prefixed) name

Attributes that are declared #IMPLIED, but not specified in the start tag, are not in the list. In addition, attributes used as namespace declarations (xmlns*) are not in the list unless the "http://xml.org/sax/features/namespace-prefixes" feature is set to True. (It is False by default.)

If the namespace-prefixes feature is False, access by qualified name (QName) may not be available.

If the "http://xml.org/sax/features/namespaces" feature is False, access by namespace-qualified names may not be available.

**Note**  Information provided during an event call is valid only during the invocation of that event. This applies to both strings and interfaces. Performance would decrease significantly otherwise.

**Methods**

- `getIndexFromName` Finds an attribute by name and returns the (zero-based) index for the attribute.
- `getIndexFromQName` Finds an attribute by XML 1.0 qualified name and returns the (zero-based) index for the attribute.
- `getLocalName` Finds an attribute's local name by index.
- `getQName` Finds an attribute's qualified name (QName) by index.
- `getType` Finds an attribute by index and returns the attribute's type, as declared in the document type definition (DTD) or schema.
  Finds an attribute by namespace name and returns the
**getTypeFromName**
attribute's type, as declared in the DTD or schema.

**getTypeFromQName**
Finds an attribute by XML 1.0 qualified name and returns the attribute's type, as declared in the DTD or schema.

**getURI**
Finds an attribute's namespace URI by index.

**getValue**
Returns the text value of an attribute.

**getValueFromName**
Finds the attribute by namespace name and returns the attribute's text value.

**getValueFromQName**
Finds the attribute by XML 1.0 qualified name and returns the attribute's text value.

[Visual Basic]
The VBSAXAttributes interface is implemented by the Simple API for XML (SAX) reader.

**Property**

In addition to the preceding methods, the following property also applies to the IVBSAXAttributes interface.

**length**
Returns the count of attributes for an element.

[C/C++]

**Methods**

In addition to the preceding methods, the following methods also apply to the ISAXAttributes interface.

**getLength**
Returns the count of attributes.

**getName**
Returns all information related to the name of an attribute at a given index.

**Versioning**
MSXML 3.0 and later

Requirements

Implementation: msxml5.dll

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Reference
getIndexFromName Method

Finds an attribute by name and returns the (zero-based) index for the attribute. For ambiguous names, this method returns the name from the first namespace.

[Visual Basic]

**Implementation Syntax**

Function getIndexFromName(ByVal strURI As String, ByVal strLocalName As String) As Long

**Usage Syntax**

StrVal = oIVBSAXAttributes.getIndexFromName(strURI, strLocalName)

**Parameters**

strURI
   The namespace URI or, if the name has no namespace URI, an empty string.
strLocalName
   The attribute's local name.

**Return Values**

If failed, raises a trappable error.

[C/C++]

**Syntax**

HRESULT getIndexFromName(
   [in] const wchar_t * pwchUri,
   [in] int cchUri,
   [in] const wchar_t * pwchLocalName,
   [in] int cchLocalName
)
Parameters

`pwchUri` [in]
The namespace URI or, if the name has no namespace URI, an empty string.
`cchUri` [in]
The length of the namespace URI string.
`pwchLocalName` [in]
The attribute's local name.
`cchLocalName` [in]
The length of the local name string.
`pnIndex` [out, retval]
The returned index value.

Return Values

S_OK
The value returned if the index value is returned successfully.
E_INVALIDARG
The value returned if no matching attribute is found.
E_FAIL
The value returned if an internal error occurs.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

`getIndexFromQName Method` | `getLength Method (C/C++)` | `length Property (Visual Basic)`

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getIndexFromQName Method

Finds an attribute by XML 1.0 qualified name (QName) and returns the (zero-based) index for the attribute. For ambiguous names, this method returns the name from the first namespace.

[Visual Basic]

Implementation Syntax

Function getIndexFromQName(ByVal strQName As String) As Long

Usage Syntax

StrVal = oIVBSAXAttributes.getIndexFromQName(strQName)

Parameters

strQName
The qualified name for the attribute.

Return Values

If failed, raises a trappable error.

[C/C++]

Syntax

HRESULT getIndexFromQName(
    [in] const wchar_t * pwchQName,
    [in] int cchQName,
    [out, retval] int * pnIndex);

Parameters
pwchQName [in]
The qualified name for the attribute.

cchQName [in]
The length of the qualified name string.

pnIndex [out, retval]
The returned index value.

**Return Values**

S_OK
The value returned if the index value is returned successfully.

E_INVALIDARG
The value returned if no matching attribute is found.

E_FAIL
The value returned if an internal error occurs.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

getLength Method (C/C++) | length Property (Visual Basic)

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getLength Method (C/C++)

Returns the count of attributes. After you know the number, you can iterate through the list of attributes.

Syntax

```c
HRESULT getLength(
    [out, retval] int * pnLength);
```

Parameters

`pnLength [out, retval]`
The count of attributes in a list.

Return Values

```c
S_OK
    The value returned if the count is returned successfully.
E_FAIL
    The value returned if an internal error occurs.
```

See Also

getURI Method | getLocalName Method | getQName Method | getName Method (C/C++) | getValue Method | getValueFromName Method | getValueFromQName Method

Applies to: ISAXAttributes Interface
**getLocalName Method**

Finds an attribute's local name by index. For "xmlns", this value is empty.

[Visual Basic]

**Implementation Syntax**

Function getLocalName(ByVal nIndex As Long) As String

**Usage Syntax**

StrVal = oIVBSAXAttributes.getLocalName(nIndex)

**Parameters**

`nIndex`  
The attribute's index value.

**Return Values**

If failed, raises a trappable error.

[C/C++]

**Syntax**

HRESULT getLocalName(  
    [in] int nIndex,  
    [out] const wchar_t ** ppwchLocalName,  
    [out] int * pcchLocalName);

**Parameters**

`nIndex` [in]  
The attribute's index value.
ppwchLocalName [out]
The attribute's local name.

pcchLocalName [out]
The length of the local name string.

**Return Values**

S_OK
The value returned if the local name is returned successfully.

E_FAIL
The value returned if an internal error occurs.

E_INVALIDARG
The value returned if the local name cannot be found.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

getLength Method (C/C++) | length Property (Visual Basic)

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getName Method (C/C++)**

Returns all information related to the name of an attribute at a given index.

**Note** MSXML 4.0 and later does not check for Null values in pointers used as output parameters with this method. For more information, see Remarks, below.

**Syntax**

```c
HRESULT getName(
    [in] int nIndex,
    [out] const wchar_t ** ppwchUri,
    [out] int * pcchUri,
    [out] const wchar_t ** ppwchLocalName,
    [out] int * pcchLocalName,
    [out] const wchar_t ** ppwchQName,
    [out] int * pcchQName);
```

**Parameters**

- **nIndex [in]**
  The attribute's index value.
- **ppwchUri [out]**
  The attribute's URI.
- **pcchUri [out]**
  The length of the URI string.
- **ppwchLocalName [out]**
  The attribute's local name.
- **pcchLocalName [out]**
  The length of the local name string.
- **ppwchQName [out]**
  The qualified prefixed name of the attribute.
- **pcchQName [out]**
  The length of the qualified name string.

**Return Values**
S_OK

The value returned if successful. Returns the results as a combination of getURI, getLocalName, and getQName, with the appropriate parameters either filled in or empty.

E_FAIL

The value returned if an internal error occurs.

E_INVALIDARG

The value returned if an invalid index is provided.

Remarks

In previous versions of MSXML, a pointer used in an output parameter could have a value of Null. In some cases, this allowed you to limit or scope the output to return only a particular output parameter when other output parameters were not of interest. Beginning with MSXML 4.0, pointers to output parameters used with this method are no longer checked for Null values, and you must always get all three output parameter values (i.e., the URI, local name, and QName).

With the current design, a value of Null is treated the same as any other pointer value, which might be incorrect (for example, in the case of a pointer to address 1). It is now left to the individual application to ensure that any pointer used as an output parameter is correct. When writing your application code, ensure that any pointer passed as an output parameter is not Null.

See Also

getAddress Method (C/C++) | getURI Method | getLocalName Method | getQName Method

Applies to: ISAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getQName Method

Finds an attribute's qualified name (QName) by index. For the "xmlns" namespace, the qname value is empty.

[Visual Basic]

Implementation Syntax

Function getQName(ByVal nIndex As Long) As String

Usage Syntax

StrVal = oIVBSAXAttributes.GetQName(nIndex)

Parameters

nIndex
The attribute's index value.

Return Values

If failed, raises a trappable error.

[C/C++]

Syntax

HRESULT getQName(
   [in] int nIndex,
   [out] const wchar_t ** ppwchQName,
   [out] int * ppchQName);

Parameters

nIndex [in]
The attribute's index value.

`ppwchQName [out]`  
The attribute's qualified name.

`ppchQName [out]`  
The length of the qualified name string or –1 (if zero-terminated).

**Return Values**

S_OK  
The value returned if the QName is returned successfully.

E_FAIL  
The value returned if an internal error is reported.

E_INVALIDARG  
The value returned if QName is not found.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[getLength Method (C/C++)](#) | [length Property (Visual Basic)](#)

Applies to: [ISAXAttributes/IVBSAXAttributes Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getTipo Method**

Finds an attribute by index and returns the attribute's type, as declared in the document type definition (DTD) or schema. If there is no DTD or schema or the schema does not map directly to a DTD attribute type, the return value is "CDATA". For ambiguous names, the type from the first namespace is returned.

[Visual Basic]

**Implementation Syntax**

Function getType(ByVal nIndex As Long) As String

**Usage Syntax**

StrVal = oIVBSAXAttributes.getType(nIndex)

**Parameters**

*nIndex
The attribute's index value.

**Return Values**

If the type is unknown, returns "CDATA".

If failed, raises a trappable error.

[C/C++]

**Syntax**

HRESULT getType(
    [in] int nIndex,
    [out] const wchar_t ** ppwchType
    [out] int * pcchType);
**Parameters**

*nIndex* [in]
   The attribute's index value.

*ppwchType* [out]
   The returned type of the attribute.

*pcchType* [out]
   The length of the returned type attribute string.

**Return Values**

*S_OK*
   The value returned if the attribute type is returned successfully.

*E_INVALIDARG*
   The value returned for an invalid index or if no matching attribute is found.

*E_FAIL*
   The value returned if an internal error occurs.

**Remarks**

The returned attribute type can be (but is not limited to) one of the following: CDATA, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

*getTypeFromName Method | getTypeFromQName Method*

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getTypeFromName Method**

Finds an attribute by namespace name and returns the attribute's type, as declared in the document type definition (DTD) or schema. If a DTD or schema does not exist or the schema type does not map directly to a DTD attribute type, the return value is "CDATA".

[Visual Basic]

**Implementation Syntax**

Function getTypeFromName(ByVal strURI As String, ByVal strLocalName As String) As String

**Usage Syntax**

StrVal = oIVB SAXAttributes.getTypeFromName(strURI, strLocalName)

**Parameters**

strURI
The namespace URI or, if the namespace has no URI, an empty string.

strLocalName
The local name of the attribute.

**Return Values**

If the type is unknown, returns CDATA.

[C/C++]

**Syntax**

HRESULT getTypeFromName(
    [in] const wchar_t * pwchUri,
    [in] int cchUri,
[in] const wchar_t * pwchLocalName,
[in] int cchLocalName,
[out] const wchar_t ** ppwchType,
[out] int * pcchType);

Parameters

pwchUri [in]
The namespace URI or, if the namespace has no URI, an empty string.
cchUri [in]
The length of the namespace URI string.
pwchLocalName [in]
The local name of the attribute.
cchLocalName [in]
The length of the local name string.
ppwchType [out]
The returned type of the attribute.
pcchType [out]
The length of the attribute-type string.

Return Values

S_OK
The value returned if the attribute type is returned successfully.
E_INVALIDARG
The value returned if an invalid index or no matching attribute is found.
E_FAIL
The value returned if an internal error is reported.

Remarks

The returned attribute type can be (but is not limited to) one of the following:
CDATA, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also
**get\(\text{Type}\) Method** | **get\(\text{TypeFromName}\) Method**

Applies to: **ISAXAttributes/IVBSAXAttributes Interface**
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getTypeFromQName Method**

Finds an attribute by XML 1.0 qualified name (QName) and returns the attribute's type, as declared in the document type definition (DTD) or schema. If a DTD or schema does not exist or the schema type does not map directly to a DTD attribute type, the return value is "CDATA".

[Visual Basic]

**Implementation Syntax**

Function getTypeFromQName(ByVal strQName As String) As String

**Usage Syntax**

StrVal = oIVBSAXAttributes.getTypeFromQName(strQName)

**Parameters**

*strQName*

The qualified name.

**Return Values**

If the type is unknown, returns CDATA.

[C/C++]

**Syntax**

HRESULT getTypeFromQName(
    [in] const wchar_t * pwchQName,
    [in] int cchQName,
    [out] const wchar_t ** ppwchType,
    [out] int * pcchType
);
Parameters

\textit{pwchQName [in]}
  The qualified name.
\textit{cchQName [in]}
  The length of the qualified name string.
\textit{ppwchType [out]}
  The returned type of the attribute.
\textit{pcchType [out]}
  The length of the attribute-type string.

Return Values

\textbf{S\_OK}
  The value returned if the attribute type is returned successfully.
\textbf{E\_INVALIDARG}
  The value returned for an invalid index or if no matching attribute is found.
\textbf{E\_FAIL}
  The value returned if an internal error occurs.

Remarks

The returned attribute type can be (but is not limited to) one of the following: CDATA, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ▼ in the upper-left corner of the page.

See Also

\texttt{getType Method}

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getURI Method

Finds an attribute's namespace URI by index.

[Visual Basic]

**Implementation Syntax**

Function getURI(ByVal nIndex As Long) As String

**Usage Syntax**

StrVal = oIVBSAXAttributes.getURI(nIndex)

**Parameters**

*nIndex*

The attribute's index (zero-based).

**Return Values**

If a namespace URI is not available, returns an empty string.

If the index is out of range, returns a trappable error message.

[C/C++]

Returns all the information related to the name of the attribute available at the index. Each return string can be disabled by passing Null in for the *text out* parameter. It is an error to pass Null for the character count unless Null is also passed for the corresponding *text out* parameter.

**Syntax**

```c
HRESULT getURI(
    [in] int nIndex,
```
[out] const wchar_t ** ppwchUri,
[out] int * pcchUri);

Parameters

nIndex [in]
The attribute's index (zero-based).

ppwchUri [out]
The namespace URI.

pcchUri [out]
The length of the namespace URI.

Return Values

S_OK
The value returned if the URI is returned successfully.

E_INVALIDARG
The value returned for an invalid index or if no matching attribute is found.

E_FAIL
The value returned if an internal error occurs.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

getLength Method (C/C++) | length Property (Visual Basic)

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getValue Method

Returns the text value of the attribute.

[Visual Basic]

If the index is out of range, this method returns a trappable error.

Implementation Syntax

Function getValue(ByVal nIndex As Long) As String

Usage Syntax

StrVal = oIVBSAXAttributes.getValue(nIndex)

Parameters

nIndex
   The attribute index (zero-based).

Return Values

If failed, raises a trappable error if the index is out of range.

[C/C++]

If the index is out of range, this method returns a Null value.

Syntax

HRESULT getValue(
   [in] int nIndex,
   [out] const wchar_t ** ppwchValue,
   [out] int * pcchValue);
Parameters

nIndex [in]
The attribute index (zero-based).

ppwchValue [out]
The string value of the attribute.

pcchValue [out]
The length of the attribute string.

Return Values

S_OK
The value returned if the text value is returned successfully.

E_INVALIDARG
The value returned if an invalid index or no matching attribute is reported.

E_FAIL
The value returned if an internal error occurs.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ☰ in the upper-left corner of the page.

See Also

gGetLength Method (C/C++) | length Property (Visual Basic)

Applies to: ISAXAttributes/IVBSAXAttributes Interface
getValueFromName Method

Finds the attribute by namespace name and returns the attribute's text value. For ambiguous names, this method returns the value from the first namespace.

[Visual Basic]

**Implementation Syntax**

```vbnet
Function getValueFromName(ByVal strURI As String,_
    ByVal strLocalName As String) As String
```

**Usage Syntax**

```vbnet
StrVal = oIVBSAXAttributes.getValueFromName(strURI,strLocalName)
```

**Parameters**

- **strURI**
  The namespace URI or, if the name has no namespace URI, an empty string.
- **strLocalName**
  The local name of the attribute.

**Return Values**

If failed, raises a trappable error.

[C/C++]

**Syntax**

```c
HRESULT getValueFromName(
    [in] const wchar_t * pwchUri,
    [in] int cchUri,
    [in] const wchar_t * pwchLocalName,
    [in] int cchLocalName,
```
[out] const wchar_t ** ppwchValue,
[out] int * pcchValue);

**Parameters**

*pwchUri* [in]
The namespace URI or, if the namespace has no URI, an empty string.

*cchUri* [in]
The length of the URI string.

*pwchLocalName* [in]
The local name of the attribute.

*cchLocalName* [in]
The length of the local name string.

*ppwchValue* [out]
The string value of the attribute.

*pcchValue* [out]
The length of the attribute's string value.

**Return Values**

**S_OK**
The value returned if the text value is returned successfully.

**E_INVALIDARG**
The value returned for an invalid index or if no matching attribute is reported.

**E_FAIL**
The value returned if an internal error occurs.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

getLength Method (C/C++) | length Property (Visual Basic)

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getValueFromQName Method

Finds the attribute by XML 1.0 qualified name and returns the attribute's text value.

[Visual Basic]

Implementation Syntax

Function getValueFromQName(ByVal strQName As String) As String

Usage Syntax

StrValue = oIVBSAXAttributes.getValueFromQName(strQName)

Parameters

strQName
The XML 1.0 qualified name.

Return Values

If failed, raises aappable error.

[C/C++]

Syntax

HRESULT getValueFromQName(
    [in] const wchar_t * pwchQName,  
    [in] int cchQName,  
    [out] const wchar_t ** pwchValue,  
    [out] int * pcchValue);

Parameters
The XML 1.0 qualified name.

The length of the qualified name string.

The string value of the qualified name.

The length of the qualified name.

Return Values

S_OK
The value returned if the text value is returned successfully.

E_INVALIDARG
The value returned for an invalid index or if no matching attribute is found.

E_FAIL
The value returned if an internal error occurs.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

getLength Method (C/C++) | length Property (Visual Basic)

Applies to: ISAXAttributes/IVBSAXAttributes Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
length Property (Visual Basic)

Returns the count of attributes for an element. After you know the number, you can iterate through the list of attributes.

**Implementation Syntax**

```
Property Get IVBSAXAttributes_length() As Long
```

**Usage Syntax**

```
StrVal = oIVBSAXAttributes.length
```

**Parameters**

`length`

The count of attributes in a list.

**Return Values**

If there are no attributes in the collection, returns a "0".

If failed, raises a trappable error.

**See Also**

- [getURI Method](#)
- [getLocalName Method](#)
- [getValue Method](#)
- [getValueFromName Method](#)
- [getValueFromQName Method](#)

Applies to: [IVBSAXAttributes Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXContentHandler Interface

The ISAXContentHandler/IVBSAXContentHandler interface receives notification of the logical content of a document. This is the main interface implemented by most Simple API for XML (SAX) applications.

**Note** Only one ContentHandler can be registered at a time.

**Methods**

- **characters** Receives notification of character data.
- **endDocument** Receives notification of the end of a document.
- **startDocument** Receives notification of the beginning of a document.
- **endElement** Receives notification of the end of an element.
- **startElement** Receives notification of the beginning of an element.
- **ignorableWhitespace** Receives notification of ignorable white space in element content. This method is not called in the MSXML 4.0 and later, because the SAX2 implementation is nonvalidating.
- **endPrefixMapping** Indicates the end of a namespace prefix that maps to a URI.
- **startPrefixMapping** Indicates the beginning of a namespace prefix that maps to a URI.
- **processingInstruction** Receives notification of a processing instruction.
- **skippedEntity** Receives notification of a skipped entity.

[Visual Basic]

A SAX application must be informed of basic parsing events. To provide this information, it implements the IVBSAXContentHandler interface and uses the contentHandler property to register an instance with the SAXXML Reader. The reader then uses this instance to report basic document-related events, such as
the start of elements, the end of elements, and character data.

The **IVBSAXContentHandler** interface handles events passed by the **SAXXMLReader**. The interface handles these events in the order of the information within the document being parsed. For example, **IVBSAXContentHandler** handles the **startElement** event, then all of the content of an element, and then the **endElement** event. The content of an element can include character data, processing instructions, and subelements.

The interface includes a series of methods that the reader invokes during the parsing operation. The reader passes the appropriate information to the parameters of the method. To add functionality to a method, you add code to the method in the **ContentHandler** implementation. In most cases, you will not call these methods yourself from within the **ContentHandler**. Therefore, the syntax shown for the **IVBSAXContentHandler** interface focuses on the implementation syntax, rather than the usage syntax.

**Important** The SAX2 reader passes string values to **IVBSAXContentHandler** methods by reference, rather than by value.

**Properties**

In addition to the preceding methods, the following property also applies to the **IVBSAXContentHandler** interface.

```c/c++

**documentLocator**

Receives an interface pointer to the **IVBSAXLocator** interface, which provides methods for returning the column number, line number, public ID, or system ID for a current document event.

[C/C++]

If a SAX application must be informed of basic parsing events, it implements the **ISAXContentHandler** interface and uses the **putContentHandler** method to register an instance with the **SAXXMLReader**. The reader then uses this instance to report basic document-related events, such as the start of elements, the end of elements, and character data.

The **ISAXContentHandler** interface handles events passed by the **SAXXMLReader**.
The interface handles these events in the order of the information within the document being parsed. For example, ISAXContentHandler handles the startElement event, then all of the content of an element, and then the endElement event. The content of an element can include character data, processing instructions, and subelements.

**Methods**

In addition to the preceding methods, the following method also applies to the ISAXContentHandler interface.

**putDocumentLocator** Receives an interface for locating the origin of SAX document events.

**Versioning**

MSXML 3.0 and later

**Requirements**

**Implementation:** msxml5.dll

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

colorHandler Property (Visual Basic) | putContentHandler Method (C/C++) | IVBSSAXLocator Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
characters Method

Receives notification of character data.

[Visual Basic]

The reader calls the characters method and passes it character data. The reader can pass all contiguous character data in a single chunk or split it into several chunks. All characters in a single event, however, must come from the same external entity, so that the Locator object provides useful information. The ContentHandler must not attempt to read from the array outside the specified range. The reader calls the characters method between the startElement and endElement methods. The reader passes strings to the characters method by reference (ByRef), not by value (ByVal). This improves performance and avoids extra string copying.

Implementation Syntax

Sub characters(strChars As String)

Usage Syntax

Dim strChars As String
strChars = "Some character data as content for the reader to handle.
oContentHandler.characters strChars

Parameters

strChars
The character data.

Return Values

If failed, the ContentHandler raises a trappable error.

Example
The following example uses the characters method to insert a carriage return (CR) character for each line feed (LF) character in the character data. The string that results is then appended to the current text of a text box (Text1) on the main application form (Form1).

Private Sub IVBSAXContentHandler_characters(strChars As String)
    strChars = Replace(strChars, vbLf, vbCrLf)
End Sub

Note This code technique can be used to preserve any CRs in your XML document input when your Microsoft® Visual Basic® application displays or prints it as output. Otherwise, carriage return/line feed (CR/LF) occurrences are converted to LF-only data by the reader.

[C/C++]

The reader calls the characters method to report each chunk of character data. The reader can return all contiguous character data in a single chunk or split it into several chunks. All characters in a single event, however, must come from the same external entity so that the Locator object can provide useful information. The application must not attempt to read from the array outside the specified range.

Syntax

HRESULT characters(
    [in] const wchar_t * pwchChars,
    [in] int cchChars);

Parameters

pwchChars [in]
    The character data.

cchChars [in]
    The length of the character string.

Return Values

S_OK
The value returned if the character data is returned successfully.

E_FAIL
The value returned if an internal error occurs.

Remarks

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
endDocument Method

Receives notification of the end of a document. The reader invokes the endDocument method only once. The endDocument method is the last method invoked during the parse. The reader does not invoke this method until it has either abandoned parsing upon encountering an unrecoverable error or has reached the end of the document.

[Visual Basic]

Implementation Syntax

Sub endDocument()

Usage Syntax

oContentHandler.endDocument

Return Values

If failed, the ContentHandler might raise an exception. This can be any Simple API for XML (SAX) exception. The exception might wrap another exception.

Example

The following example uses the endDocument method to add a line to the current contents of a text box (Text1) on the main application form (Form1). The new text indicates that the reader has finished the document.

Private Sub IVBSAXContentHandler_endDocument()
End Sub

[C/C++]

Syntax
HRESULT endDocument();

**Return Values**

S_OK
   The value returned if no errors occur.
E_FAIL
   The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[startDocument Method](#)

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
startDocument Method

Receives notification of the beginning of a document. The reader invokes the startDocument method only once.

[Visual Basic]

The startDocument method is invoked before any other methods in the IVBSAXContentHandler interface or in the IVBSAXDTDHandler interface. Only the documentLocator property is invoked before the startDocument method.

Implementation Syntax

Sub startDocument()

Usage Syntax

oContentHandler.startDocument

Return Values

If failed, the ContentHandler raises a trappable error.

Example

The following example uses the startDocument method to set the text property for a text box (Text1) on the main application form (Form1). The new text indicates that the reader has begun reading the document.

Private Sub IVBSAXContentHandler_startDocument()
    Form1.Text1.Text = "Starting document." & vbCrLf
End Sub

[C/C++]

The startDocument method is invoked before any other methods in the ISAXContentHandler interface or in the ISAXDTDHandler interface, with the
exception of the putDocumentLocator method.

Syntax

HRESULT startDocument();

Return Values

S_OK
The value returned if successful.

E_FAIL
The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

documentLocator Property (Visual Basic) | endDocument Method | ISAXDTDHandler Interface | putDocumentLocator Method (C/C++)

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
endElement Method

Receives notification of the end of an element. The reader invokes this method at the end of every element in the XML document.

[Visual Basic]

A corresponding startElement method is invoked for every endElement method, even when the element is empty.

Implementation Syntax

Sub endElement(strNamespaceURI As String, strLocalName As String, strQName As String)

Usage Syntax

oContentHandler.endElement "xsl", "stylesheet", "xsl:stylesheet"

Parameters

strNamespaceURI
   The namespace URI or an empty string (if the element has no namespace URI, or if namespaces are not being processed).
strLocalName
   The local name, without prefix, or an empty string (if namespace processing is not being performed).
strQName
   The XML 1.0 qualified name (QName), with prefix, or an empty string (if the QName is not available).

Return Values

If failed, the ContentHandler raises a trappable error.

Example
The following example uses the endElement method to set the text property for a text box (Text1) on the main application form (Form1). The new text indicates that the reader has finished reading the current element.

```vbnet
Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As String,
   strLocalName As String, strQName As String)
        strQName & vbCrLf
End Sub
```

Remarks

This method allows up to three names for each element:

- The namespace URI.
- The local name.
- The QName, if available.

Which of these names are specified depends on the values of the "http://xml.org/sax/features/namespaces" and the "http://xml.org/sax/features/namespace-prefixes" features. The namespace URI and local name are required when the "http://xml.org/sax/features/namespaces" feature is True (the default). These values are optional when this feature is set to False.

Note Neither the URI nor the local name value can be specified alone. Both values are required if either is used.

The QName is required when the "http://xml.org/sax/features/namespaces-prefixes feature" is set to True and is optional when this feature is set to False (the default).

[C/C++]

A corresponding startElement event occurs for every endElement event, even when the element is empty.

This event allows up to three name components for each element:

- The namespace URI.
- The local name.
• The qualified XML 1.0 name (QName), with prefix, or an empty string if the QName is not available.

Any or all of these name components may be provided, depending on the values of the "http://xml.org/sax/features/namespaces" and the "http://xml.org/sax/features/namespace-prefixes" features. The namespace URI and local name are required when the "http://xml.org/sax/features/namespaces" feature is set to True (the default). These values are optional when this feature is set to False.

**Note** Neither the URI nor the local name value can be specified alone. Both values are required if either is used.

The QName is required when the "http://xml.org/sax/features/namespaces-prefixes feature" is set to True and is optional when this feature is set to False (the default).

**Syntax**

```csharp
HRESULT endElement(
    [in] const wchar_t * pwchNamespaceUri,
    [in] int cchNamespaceUri,
    [in] const wchar_t * pwchLocalName,
    [in] int cchLocalName
    [in] const wchar_t * pwchQName,
    [in] int cchQName);
```

**Parameters**

`pwchNamespaceUri [in]`
  The namespace URI.

`cchNamespaceUri [in]`
  The length of the namespace URI string.

`pwchLocalName [in]`
  The local name string.

`cchLocalName [in]`
  The length of the local name string.

`pwchQName [in]`
  The QName, with prefix, or an empty string (if QNames are not available).

`cchQName [in]`
The length of the QName string.

**Return Values**

S_OK
   The value returned if no errors occur.
E_FAIL
   The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[startElement Method](#)

Applies to: [ISAXContentHandler/IVBSAXContentHandler Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
startElement Method

Receives notification of the beginning of an element. The reader invokes the startElement method at the beginning of every element in the XML document.

[Visual Basic]

A corresponding endElement method is called for every startElement method, even when the element is empty.

Implementation Syntax

Sub startElement(strNamespaceURI As String, strLocalName As String, strQName As String, oAttributes As IVBSAXAttributes)

Usage Syntax

oContentHandler.startElement "xsl", "stylesheet", "xsl:stylesheet", oAttributes

Parameters

strNamespaceURI
   The namespace URI.
strLocalName
   The local name string.
strQName
   The XML 1.0 qualified name (QName), with prefix, or an empty string (if QNames are not available).
oAttributes
   The attributes collection attached to the element.

Return Values

If failed, the ContentHandler raises a trappable error.
Example

The following example uses the `startElement` method to set the text property for a text box (Text1) on the main application form (Form1). The new text indicates that the reader has begun reading the current element.

```vbnet
Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As String, strLocalName As String, strQName As String, oAttributes As IVBSAXAttributes)
End Sub
```

Remarks

This method allows up to three names for each element:

- The namespace URI.
- The local name.
- The QName, if available.

Any or all of these name components may be specified, depending on the values of the "http://xml.org/sax/features/namespaces" and the "http://xml.org/sax/features/namespace-prefixes" features. The namespace URI and local name are required when the "http://xml.org/sax/features/namespaces" feature is set to True (the default). These values are optional when this feature is set to False.

**Note**  Neither the URI nor local name value can be specified on its own. Both values are required if either is used.

The QName is required when the "http://xml.org/sax/features/namespace-prefixes feature" is set to True and is optional when this feature is set to False (the default).

The attribute list provided contains only attributes with explicit values (specified or default). The attribute list omits #IMPLIED attributes. Furthermore, the attribute list contains attributes used for namespace declarations (`xmlns*` attributes) only if the "http://xml.org/sax/features/namespace-prefixes" feature is set to True. This feature is set to False by default.
A corresponding `endElement` method is called for every `startElement` event, even when the element is empty.

The `startElement` event allows up to three name components for each element:

- The namespace URI.
- The local name.
- The qualified XML 1.0 name (QName).

Any or all of these name components may be specified, depending on the values of the "http://xml.org/sax/features/namespaces" and the "http://xml.org/sax/features/namespace-prefixes" features. The namespace URI and local name are required when the "http://xml.org/sax/features/namespaces" feature is set to True (the default). These values are optional when this feature is set to False.

**Note**  Neither the URI nor local name value can be specified alone. Both values are required if either is used.

The QName is required when the "http://xml.org/sax/features/namespace-prefixes feature" is set to True and is optional when this feature is set to False (the default).

The attribute list provided contains only attributes with explicit values (specified or default). The attribute list omits #IMPLIED attributes. Furthermore, the attribute list contains attributes used for namespace declarations (xmlns* attributes) only if the "http://xml.org/sax/features/namespace-prefixes" feature is set to True. This feature is set to False by default.

**Syntax**

```c
HRESULT startElement(
    [in] const wchar_t * pwchNamespaceUri,
    [in] int cchNamespaceUri,
    [in] const wchar_t * pwchLocalName,
    [in] int cchLocalName
    [in] const wchar_t * pwchQName
    [in] int cchQName);
```

Parameters

pwchNamespaceUri [in]
The namespace URI.
cchNamespaceUri [in]
The length of the namespace URI.
pwchLocalName [in]
The local name string.
cchLocalName [in]
The length of the local name.
pwchQName [in]
The QName, with prefix, or, an empty string (if QNames are not available).
cchQName [in]
The length of the QName.
pAttributes [in]
The attributes attached to the element.

Return Values

S_OK
The value returned if no errors occur.
E_FAIL
The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

endElement Method

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**ignoreWhitespace Method**

This method is not called in the current implementation because the Simple API for XML (SAX2) implementation is nonvalidating.

[Visual Basic]

You can manually call the `ignoreWhitespace` method to pass white space characters such as space, carriage return (CR), tab, or line feed (LF) to the internal buffer of the `MXXMLWriter` object.

**Implementation Syntax**

```vbscript
Sub ignoreWhitespace(strChars As String)
```

**Usage Syntax**

```vbscript
Dim stringOfXml As String
stringOfXml = xmlDoc.xml
oContentHandler.ignoreWhitespace stringOfXml
```

**Parameters**

`strChars`

The character data.

**Return Values**

If failed, the `ContentHandler` raises a trappable error.

**Example**

The following example shows how to manually call the `ignoreWhitespace` method. This method finds ignorable white space characters in character data and replaces them with two characters of white space. In this case, the tab character is replaced.
Private Sub IVBSAXContentHandler_characters(strChars As String)
    If strChars Like vbTab Or strChars Like "*[vbTab]*" Then
        Call IVBSAXContentHandler_ignorableWhitespace(strChars)
    End If
    strChars = Replace(strChars, vbLf, vbCrLf)
    Form1.TextBox1.Text = Form1.TextBox1.Text & strChars
End Sub

Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars As String)
    strChars = Replace(strChars, vbTab, "\t\t")
End Sub

[C/C++]

You can manually call the ignorableWhitespace method to pass ignorable white space characters to MXXMLWriter.

Syntax

HRESULT ignorableWhitespace(
    [in] const wchar_t * pwchChars,
    [in] int cchChars);

Parameters

pwchChars [in]
    The character data.

cchChars [in]
    The length of the character string.

Return Values

S_OK
    The value returned if the character data is returned successfully.
E_FAIL
    The value returned if an internal error occurs.

Remarks

The ignorableWhitespace method does not convert CR/LF("\r\n") to the Java LF("\n") implementation.
To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Use MXXMLWriter

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
endPrefixMapping Method

Ends the scope of the prefix-URI namespace mapping.

[Visual Basic]
The reader calls this method after the corresponding endElement method.

Implementation Syntax

Sub endPrefixMapping(strPrefix As String)

Parameters

strPrefix
   The prefix being mapped.

Return Values

If failed, the ContentHandler raises a trappable error.

Example

The following example shows how to use the endPrefixMapping method to indicate that a namespace prefix has been ended or discontinued in the document.

Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As String)
   Dim msg As String
   msg = "End prefix mapping." & vbCrLf & _
         "strPrefix: " & strPrefix & vbCrLf
   Form1.TextBox1.Text = Form1.TextBox1.Text & msg
End Sub

[C/C++]

This event always occurs after the corresponding endElement event, but the
order of the endPrefixMapping events is not otherwise guaranteed.

Syntax

HRESULT endPrefixMapping(
    [in] const wchar_t * pwchPrefix,
    [in] int cchPrefix);

Parameters

pwchPrefix [in]
    The prefix being mapped.

ccchPrefix [in]
    The length of the prefix string.

Return Values

S_OK
    The value returned if no errors occur.

E_FAIL
    The value returned if the parse operation should be aborted.

Remarks

The information from this event is not necessary for normal namespace processing. The reader automatically replaces prefixes for element and attribute names when the "http://xml.org/sax/features/namespaces" feature is set to True (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values and they cannot safely be expanded automatically. In these cases, the startPrefixMapping and endPrefixMapping methods can supply the information needed to expand prefixes.

There is no guarantee that the startPrefixMapping and endPrefixMapping events are properly nested relative to each other. All startPrefixMapping events occur before the corresponding startElement event, and all endPrefixMapping events occur after the corresponding endElement event, but
their order is not otherwise guaranteed.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

endElement Method | startElement Method | startPrefixMapping Method

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**startPrefixMapping Method**

Starts the scope of the prefix-URI namespace mapping.

[Visual Basic]

This method is always invoked before the corresponding startElement method.

**Implementation Syntax**

```vbs
Sub startPrefixMapping(strPrefix As String, strURI As String)

Parameters

**strPrefix**
- The namespace prefix being declared.

**strURI**
- The namespace URI to which the prefix is mapped.

**Return Values**

If failed, the `ContentHandler` raises a trappable error.

**Example**

The following example shows how to use the `startPrefixMapping` method to indicate each time a namespace prefix occurs in the document.

```vbs
Private Sub IVBSAXContentHandler_startPrefixMapping(strPrefix As String, strURI As String)
    Dim msg As String
    msg = "Start prefix mapping." & vbCrLf & _
        "strPrefix: " & strPrefix & vbCrLf & _
        "strURI: " & strURI & vbCrLf
    Form1.TextBox1.Text = Form1.TextBox1.Text & msg
End Sub
```

[C/C++]
This method is always invoked before the corresponding startElement event, but the order of the startPrefixMapping events is not otherwise guaranteed.

Syntax

```c
HRESULT startPrefixMapping(
    [in] const wchar_t * pwchPrefix,
    [in] int cchPrefix,
    [in] const wchar_t * pwchUri
    [in] int cchUri);
```

Parameters

- `pwchPrefix` [in]
  The prefix being mapped.
- `cchPrefix` [in]
  The length of the prefix string, or –1 (if zero-terminated).
- `pwchUri` [in]
  The namespace URI to which the prefix is mapped.
- `cchUri` [in]
  The length of the namespace URI string.

Return Values

- `S_OK`
  The value returned if no errors occur.
- `E_FAIL`
  The value returned if the parse operation should be aborted.

Remarks

The information from this event is not necessary for normal namespace processing. The reader automatically replaces prefixes for element and attribute names when the "http://xml.org/sax/features/namespaces" feature is set to True (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values and they cannot safely be expanded automatically. In these cases, the startPrefixMapping and endPrefixMapping methods can
supply the information needed to expand prefixes.

There is no guarantee that the startPrefixMapping and endPrefixMapping events are properly nested relative to each other. All startPrefixMapping events occur before the corresponding startElement event, and all endPrefixMapping events occur after the corresponding endElement event, but their order is not otherwise guaranteed.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[endElement Method](#) | [endPrefixMapping Method](#) | [startElement Method](#)

Applies to: [ISAXContentHandler/IVBSAXContentHandler Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
processingInstruction Method

Receives notification of a processing instruction. The reader invokes the processingInstruction method once for each processing instruction found. Processing instructions can occur before or after the main document element.

[Visual Basic]

**Implementation Syntax**

Sub processingInstruction(strTarget As String, strData As String)

**Parameters**

*strTarget*

The target for the processing instruction. The target is the first piece of the processing instruction. For example, xml-stylesheet is the target for the following processing instruction: `<?xml-stylesheet type="text/css" href="sample.css"?>`.

*strData*

The processing instruction data, or Null (if none is supplied). For example, `type="text/css" href="sample.css"` is the data for the following processing instruction: `<?xml-stylesheet type="text/css" href="sample.css"?>`

**Return Values**

If failed, the ContentHandler raises a trappable error.

**Example**

The following example shows how to use the processingInstruction method to pass a processing instruction, unchanged, into a text box (Text1) on the application form (Form1) that is used to display the output of the reader.

Private Sub IVBSAXContentHandler_processingInstruction(strTarget As
   "<?" & strTarget & " " & strData & "?>" & vbCrLf
End Sub

[C/C++]

Syntax

HRESULT processingInstruction(
   [in] const wchar_t * pwchTarget,
   [in] int cchTarget,
   [in] const wchar_t * pwchData,
   [in] int cchData);

Parameters

pwchTarget [in]
The target for the processing instruction. The target is the first piece of the processing instruction. For example, xml-stylesheet is the target for the following processing instruction: <?xml-stylesheet href="sample.css" type="text/css" title="Sample" ?>.

cchTarget [in]
The length of the target string.

pwchData [in]
The processing instruction data, or Null (if none is supplied).

ccchData [in]
The length of the processing instruction string.

Return Values

S_OK
The value returned if no errors occur.

E_FAIL
The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ▶ in the upper-left corner of the page.

See Also
Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
putDocumentLocator Method
(C/C++)

Receives an interface for locating the origin of Simple API for XML (SAX) document events. The reader supplies a locator to the application by invoking this method before invoking any other methods in the ISAXContentHandler interface. This locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application uses this information to report its own errors (for example, if the character content does not match the business rules of the application).

Syntax

HRESULT putDocumentLocator(
    [in] ISAXLocator * pLocator);

Parameters

pLocator [in]
    A pointer to a valid instance of the ISAXLocator interface.

Return Values

S_OK
    The value returned if no errors occur.
E_FAIL
    The value returned if the parse operation should be aborted.

Remarks

The information returned by the Locator object can sometimes be insufficient for use with a search engine. The locator returns the correct information only during the invocation of events in this interface. The application should not
attempt to use this method at any other time.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[ISAXLocator Interface] | [startDocument Method]

Applies to: [ISAXContentHandler Interface]
skippedEntity Method

Receives notification of a skipped entity. The skippedEntity method is invoked for each entity skipped. The reader may skip entities if they have not been seen in the declarations. An example of this is when an entity is declared in an external DTD subset. The reader may skip external entities, depending on the values of the "http://xml.org/sax/features/external-general-entities" and "http://xml.org/sax/features/external-parameter-entities" features.

The skippedEntity method is also called in the case of an external DTD declaration. For example:

```xml
<!DOCTYPE xyz SYSTEM "http://...">
```

[Visual Basic]

**Implementation Syntax**

```vbscript
Sub skippedEntity(strName As String)

*Parameters*

*strName*

The name of the skipped entity. If the entity is a parameter entity, the name begins with '%'.

*Return Values*

If failed, the `ContentHandler` raises a trappable error.

**Example**

The following example shows how to use the skippedEntity method to pass log event information to a text box (Text2) on a form (Form1). This informs the user that entity references in the document were skipped and not resolved by the
reader during parsing.

```vbscript
Private Sub IVBSAXContentHandler_skippedEntity(strName As String)
    Dim msg As String
    Select Case strName
        Case "[dtd]",
            msg = "Reference to external DTD skipped." & vbCrLf
        Case Else
            msg = "Entity reference {" & strName & "} skipped." & vbCrLf
    End Select
End Sub
```

[C/C++]

**Syntax**

```
HRESULT skippedEntity(
    [in] const wchar_t * pwchName,
    [in] int cchName);
```

**Parameters**

- **pwchName [in]**
  The name of the skipped entity. If the entity is a parameter entity, the name begins with `%`.
- **cchName [in]**
  The number of characters in the `pwchName` string.

**Return Values**

- **S_OK**
  The value returned if no errors occur.
- **E_FAIL**
  The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**
putFeature Method

Applies to: ISAXContentHandler/IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
documentLocator Property (Visual Basic)

Receives an interface pointer to the IVBSAXLocator interface, which provides methods for returning the column number, line number, public ID, or system ID for a current document event.

The reader supplies a Locator object to the application by invoking this property before invoking any other methods in the IVBSAXContentHandler interface. This locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application uses this information for reporting its own errors (for example, if the character content does not match the business rules of the application).

**Implementation Syntax**

Property documentLocator As IVBSAXLocator

**Return Values**

If failed, the ContentHandler raises a trappable error.

**Example**

The following example shows how to set and use the documentLocator property in a typical ContentHandler class implementation. The line and column information can be used to support the implementation of log event details by the reader.

Implements IVBSAXContentHandler
Implements IVBSAXLocator

'Declare a variable for setting the locator.
Private oLocator As IVBSAXLocator

Private Property Set IVBSAXContentHandler_documentLocator(ByVal RHS
MSXML2.IVBSAXLocator)
  'Invoke the locator by setting it here before the document is read.
  Set oLocator = RHS
End Property

Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As String,
  strLocalName As String, strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttributes)
  'When the element starts, write its line/column position in file back to the text box on the application form.
  Form1.Text1.Text = Form1.Text1.Text + _
    "Starting <" + strLocalName + " > at " + _
    "Line " & Str(oLocator.lineNumber) + _
    ", Column " & Str(oLocator.columnNumber) + vbCrLf
End Sub

Remarks

The Locator object returns the correct information only during the invocation of events in this interface. The application should not attempt to use this method at any other time.

See Also

IVBSAXLocator Interface | startDocument Method

Applies to: IVBSAXContentHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXDeclHandler Interface

The ISAXDeclHandler/IVBSAXDeclHandler interface allows you to access declaration information about the following:

- Element and attribute types
- Internal and external entities

If you are using the DeclHandler with a LexicalHandler, all of the events occur between the startDTD and the endDTD events.

To set the DeclHandler for a SAX2 reader, use the putProperty method with the property ID, "http://xml.org/sax/properties/declaration-handler".

Methods

- `attributeDecl` Receives/reports an attribute type declaration.
- `elementDecl` Receives/reports an element type declaration.
- `externalEntityDecl` Receives/reports a parsed external entity declaration.
- `internalEntityDecl` Receives/reports a parsed internal entity declaration.

[Visual Basic]

The IVBSAXDeclHandler interface enables a Simple API for XML (SAX2) application to implement an optional extension handler that receives information about DTD declarations in an XML document. SAX2 readers are not required to support this handler, and this handler is not included in the core SAX2 distribution.

Data-related DTD declarations (unparsed entities and notations) are already reported through the IVBSAXDTDHandler interface.

The IVBSAXDeclHandler interface includes a series of methods that are invoked by the reader during the parsing operation. The reader passes the appropriate information to the parameters of the method. To perform some type of
processing for a method, you add code to the method in the DeclHandler implementation. In general, you will rarely call these methods yourself from within the DeclHandler. Therefore, the syntax shown for the IVBSAXDeclHandler interface focuses on the implementation syntax, rather than the usage syntax.

**Important** The SAX2 reader passes string values to IVBSAXDeclHandler methods by reference, rather than by value.

[C/C++]

This is an optional extension handler for SAX2 that provides information about DTD declarations in an XML document. Data-related DTD declarations (unparsed entities and notations) are already reported through the ISAXDTDHandler interface.

The following code shows how to use the putProperty method with the property ID, `http://xml.org/sax/properties/declaration-handler`, to set the DeclHandler for a SAX2 reader.

```c
d ... SAXXMLReader * r = ... SAXDeclHandler * dh = new SAXDeclHandler(); r->putProperty("http://xml.org/sax/properties/declaration-handler",
```

**Versioning**

MSXML 3.0 and later

**Requirements**

**Implementation:** msxml5.dll

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**
endDTD Method | ISAXDTDHandler Interface | putProperty Method | startDTD Method
attributeDecl Method

Receives/reports an attribute type declaration. Only the first declaration for an attribute is reported.

[Visual Basic]

The attribute type is one of these three choices:

- One of the following strings: "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", or "ENTITIES".
- A parenthesized token group with the separator "|" and all white space removed.
- The word "NOTATION", followed by a space, followed by a parenthesized token group with all white space removed.

Any parameter entities in the attribute value are expanded. General entities are not expanded.

Implementation Syntax

Sub attributeDecl(strElementName As String, strAttributeName As String, strType As String, strValueDefault As String, strValue As String)

Parameters

strElementName
   The name of the associated element.
strAttributeName
   The name of the attribute.
strType
   A string representing the attribute type.
strValueDefault
   A string representing the attribute default ("#IMPLIED", "#REQUIRED", or "#FIXED"), or Null (if none of these applies).
strValue
   A string representing the default value of the attribute, or Null (if there is
Return Values

If failed, the DeclHandler raises a trappable error.

[C/C++]

The attribute type is one of these two choices:

- One of the following strings: "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION".
- A parenthesized token group with the separator "|" and all white space removed.

Syntax

HRESULT attributeDecl(
    [in] const wchar_t * pwchElementName,
    [in] int cchElementName,
    [in] const wchar_t * pwchAttributeName,
    [in] int cchAttributeName,
    [in] const wchar_t * pwchType,
    [in] int cchType,
    [in] const wchar_t * pwchValueDefault,
    [in] int cchValueDefault,
    [in] const wchar_t * pwchValue,
    [in] int cchValue);

Parameters

pwchElementName [in]
    The name of the associated element.

cchElementName [in]
    The length of the element name string.

pwchAttributeName [in]
    The name of the attribute.

cchAttributeName [in]
    The length of the attribute name string.
pwchType [in]
A string representing the attribute type.
cchType [in]
The length of the attribute type string.
pwchValueDefault [in]
A string representing the attribute default ("#IMPLIED", "#REQUIRED" or "#FIXED"), or Null (if none of these apply).
cchValueDefault [in]
The length of the default string of the attribute.
pwchValue [in]
A string representing the default value of the attribute, or Null (if there is none).
cchValue [in]
The length of the default value string of the attribute.

Return Values

S_OK
The value returned if no errors are reported.
E_FAIL
The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

Applies to: ISAXDeclHandler/IVBSAXDeclHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
elementDecl Method

Receives/reports an element type declaration. The content model consists of one of the following:

- The string "EMPTY"
- The string "ANY"
- A group of values in parentheses. This group is optionally followed by an occurrence indicator (for example, `<!ELEMENT DOCUMENT (TITLE, AUTHOR+, SUMMARY*, NOTE?)> ).

The content model is normalized so that all parameter entities are fully resolved, all white space is removed, and enclosing parentheses are included. Other normalization (such as removing redundant parentheses or simplifying occurrence indicators) depends on the parser.

[Visual Basic]

Implementation Syntax

Sub elementDecl(strName As String, strModel As String)

Parameters

strName
- The element type name.
strModel
- The content model as a normalized string.

Return Values

If failed, the DeclHandler raises a trappable error.

[C/C++]

Syntax
HRESULT elementDecl(
    [in] const wchar_t * pwchName,
    [in] int cchName,
    [in] const wchar_t * pwchModel,
    [in] int cchModel);

**Parameters**

*pwchName [in]*
   The element type name.

*cchName [in]*
   The length of the element type name.

*pwchModel [in]*
   The content model as a normalized string.

*cchModel [in]*
   The length of the content model normalized string.

**Return Values**

S_OK
   The value returned if no errors are returned.

E_FAIL
   The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button ☑ in the upper-left corner of the page.

**See Also**

attributeDecl Method

Applies to: ISAXDeclHandler/IVBSAXDeclHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
externalEntityDecl Method

 Receives/reports a parsed external entity declaration. Only the effective (first) declaration for each entity is reported.

[Visual Basic]

**Implementation Syntax**

Sub externalEntityDecl(strName As String, strPublicId As String, _
strSystemId As String)

**Parameters**

strName
  The name of the entity. If it is a parameter entity, the name begins with '%'.

strPublicId
  The declared public identifier of the entity, or Null (if none was declared).

strSystemId
  The declared system identifier of the entity.

**Return Values**

If failed, the DeclHandler raises a trappable error.

[C/C++]

**Syntax**

HRESULT externalEntityDecl();
  [in] const wchar_t * pwchName,
  [in] int cchName,
  [in] const wchar_t * pwchPublicId,
  [in] int cchPublicId,
  [in] const wchar_t pwchSystemId,
  [in] int cchSystemId);
Parameters

\texttt{pwchName} [in]
   The name of the entity. If it is a parameter entity, the name begins with a "%".
\texttt{cchName} [in]
   The length of the name string of the entity.
\texttt{pwchPublicId} [in]
   The declared public identifier of the entity, or Null (if none is declared).
\texttt{cchPublicId} [in]
   The length of the public identifier string.
\texttt{pwchSystemId} [in]
   The declared system identifier of the entity.
\texttt{cchSystemId} [in]
   The length of the system identifier string.

Return Values

\texttt{S\_OK}
   The value returned if no errors are reported.
\texttt{E\_FAIL}
   The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

\texttt{internalEntityDecl Method}

Applies to: ISAXDeclHandler/IVBSAXDeclHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
internalEntityDecl Method

Receives/reports a parsed internal entity declaration. Only the effective (first) declaration for each entity is reported.

[Visual Basic]

All parameter entities in the value are expanded. General entities are not expanded.

Implementation Syntax

Sub internalEntityDecl(strName As String, strValue As String)

Parameters

strName
The name of the entity. If it is a parameter entity, the name begins with '\%'.

strValue
The replacement text of the entity.

Return Values

If failed, the DeclHandler raises a trappable error.

[C/C++]

Syntax

HRESULT internalEntityDecl(
   [in] const wchar_t * pwchName,
   [in] int cchName,
   [in] const wchar_t * pwchValue,
   [in] int cchValue);

Parameters
**pwchName [in]**

The name of the entity. If it is a parameter entity, the name begins with a "%".

**cchName [in]**

The length of the name string of the entity.

**pwchValue [in]**

The replacement text of the entity.

**cchValue [in]**

The length of the replacement text string.

**Return Values**

**S_OK**

The value returned if no errors are reported.

**E_FAIL**

The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[externalEntityDecl Method](#)

Applies to: [ISAXDeclHandler/IVBSAXDeclHandler Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Reference
ISAXDTDHandler Interface

The ISAXDTDHandler/IVBSAXDTDHandler interface receives notification of basic document type definition (DTD)-related events.

Methods

- notationDecl
  Receives notification of a notation declaration event.
- unparsedEntityDecl
  Receives notification of an unparsed entity declaration event.

[Visual Basic]

If a Simple API for XML (SAX) application requires information about notations and unparsed entities, the application implements this interface and registers an instance with the SAX2 reader using the dtdHandler property. This DTDHandler instance reports notation and unparsed entity declarations to the application.

The IVBSAXDTDHandler interface includes methods that are invoked by the reader during the parsing operation. The reader passes the appropriate information to the method's parameters. To perform some type of processing for a method, you simply add code to the method in the DTDHandler implementation. In general, it's rare that you would call these methods yourself from within the DTDHandler. As such, the syntax shown for the IVBSAXDTDHandler interface focuses on the implementation syntax, rather than usage syntax.

**Important** The SAX2 reader passes string values to IVBSAXDTDHandler methods by reference, rather than by value.

[C++]

If a Simple API for XML (SAX) application requires information about notations and unparsed entities, the application implements this interface and registers an instance with the SAX2 reader using the putDTDHandler method.
The instance of the DTDHandler reports notation and unparsed entity declarations to the application.

**Remarks**

This interface includes only those DTD events that the XML recommendation requires processors to report: notation and unparsed entity declarations.

The reader can report these events in any order, regardless of the order in which the notations and unparsed entities were declared. However, all DTD events must be reported after the ContentHandler's startDocument event, and before the first startElement event.

It is up to the application to store the information for future use (perhaps in a hash table or object tree). If the application encounters attributes of type NOTATION, ENTITY, or ENTITIES, it can use the information that it obtained through this interface to find the entity or notation corresponding to the attribute value.

**Versioning**

MSXML 3.0 and later

**Requirements**

**Implementation:** msxml5.dll

[C++]

**Header and LIB files:** msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

dtdHandler Property (Visual Basic) | putDTDHandler Method (C/C++) | startDocument Method | startElement Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
notationDecl Method

Receives notification of a notation declaration event.

[Visual Basic]

If necessary, the application can record the notation for later reference. At least one of the PublicId or SystemId values must be available. If a system entity is present and it is a URL, the reader must resolve it before passing it to the application through the notationDecl method. There is no guarantee that the notation declaration will be reported before any unparsed entities that use it.

Implementation Syntax

Sub notationDecl(strName As String, strPublicId As String,_
                 strSystemId As String)

Parameters

strName
   The notation's name.
strPublicId
   The notation's public identifier or Null (if none was given).
strSystemId
   The notation's system identifier (required).

Return Value

If failed, the DTDHandler raises a trappable error.

[C/C++]

It is the responsibility of the application to record the notation for later reference. If a system identifier is present and it is a URL, the reader must resolve it fully before passing it to the application.
Syntax

HRESULT notationDecl(
    [in] const wchar_t * pwchName,
    [in] int cchName,
    [in] const wchar_t * pwchPublicId,
    [in] int cchPublicId
    [in] const wchar_t * pwchSystemId
    [in] int cchSystemId);

Parameters

pwchName [in]
    The notation's name.

cchName [in]
    The length of the notation.

pwchPublicId [in]
    The notation's public identifier or Null (if none was given).

ccPublicId [in]
    The length of the public identifier.

pwchSystemId [in]
    The notation's system identifier or Null (if none was given).

cchSystemId [in]
    The length of the system identifier.

Return Values

S_OK
    The value returned if no errors occur.

E_FAIL
    The value returned if the parse should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

unparsedEntityDecl Method
Applies to: ISAXDTDHandler/IVBSAXDTDHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
unparsedEntityDecl Method

Receives notification of an unparsed entity declaration event. The notation name corresponds to a notation reported by the notationDecl method or event. It is up to the application to record the entity for later reference, if necessary. If the system identifier is a URL, the reader must resolve it before passing it to the application.

[Visual Basic]

Implementation Syntax

Sub unparsedEntityDecl(strName As String, strPublicId As String,__
    strSystemId As String, strNotationName As String)

Parameters

strName     
The unparsed entity's name.
strPublicId  
The entity's public identifier or Null (if none was given).
strSystemId  
The entity's system identifier (required).
strNotationName  
The name of the associated notation.

Return Values

If failed, the DTDHandler raises a trappable error.

[C/C++]

Syntax

HRESULT unparsedEntityDecl(  
    [in] const wchar_t * pwchName,
Parameters

\textit{pwchName} [in]
   The unparsed entity's name.
\textit{cchName} [in]
   The length of the entity.
\textit{pwchPublicId} [in]
   The entity's public identifier or Null (if none was given).
\textit{cchPublicId} [in]
   The length of the public identifier.
\textit{pwchSystemId} [in]
   The entity's system identifier (required).
\textit{cchSystemId} [in]
   The length of the system identifier.
\textit{pwchNotationName} [in]
   The name of the associated notation.
\textit{cchNotationName} [in]
   The length of the notation.

Return Values

\texttt{S\_OK}
   The value returned if no errors occur.
\texttt{E\_FAIL}
   The value returned if the parse should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button \(\Box\) in the upper-left corner of the page.

See Also
notationDecl Method

Applies to: ISAXDTDHandler/IVBSAXDTDHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXEntityResolver Interface

The ISAXEntityResolver/IVBSAXEntityResolver interface implements a handler to the resolveEntity event.

Methods

resolveEntity

Allows the parser to resolve external entities.

Versioning

MSXML 3.0 and later

Requirements

Implementation: msxml5.dll

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

Other Resources

XML Industry Portal
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**resolveEntity Method**

Allows the parser to resolve external entities.

[Visual Basic]

**Implementation Syntax**

Function resolveEntity(strPublicId As String, strSystemId As String)

**Parameters**

`strPublicId`  
The public identifier of the external entity being referenced, or Null (if none was supplied).

`strSystemId`  
The system identifier of the external entity being referenced.

[C/C++]

**Syntax**

HRESULT resolveEntity(
    [in] const wchar_t * pwchPublicId,
    [in] const wchar_t * pwchSystemId,
    [out, retval] VARIANT * pvarInput);

**Parameters**

`pwchPublicId` [in]  
The public identifier of the external entity being referenced, or Null (if none was supplied).

`pwchSystemId` [in]  
The system identifier of the external entity being referenced.

`pvarInput` [out, retval]  
The returned text of the entity. Acceptable input types are the same as for
the parse method.

Return Values

S_OK
The value returned if no errors are reported.
E_FAIL
The value returned if the parse operation should be aborted.

Remarks

Allow the application to resolve external entities.

When ISAXEntityResolver or IVBSAXEntityResolver is implemented, the SAX reader will call this method before opening any external entity except the top-level document entity. External entities handled by this method can include the following: the external DTD subset, external entities referenced within the DTD, and external entities referenced within the document element. The application can request that the parser resolve the entity itself, that it use an alternative URI, or that it use an entirely different input source.

Application writers can use the resolveEntity method to perform the following tasks:

- To redirect external system identifiers to secure and/or local URIs.
- To look up public identifiers in a catalogue.
- To read an entity from a database or other input source for example, a dialog box.

If the system identifier is a URL, the SAX reader must resolve it fully before reporting it to the application.

Note  For this method, the Java implementation requires the InputSource interface, which is not included in this implementation. Instead, variant is used as the data type for method output. Output currently can represent be either a binary stream (IStream) or, potentially, string data. User feedback is appreciated during prerelease. If important usage cases are found, InputSource might be included in future releases.
To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[parse Method](#) | [unparsedEntityDecl Method](#)

Applies to: [ISAXEntityResolver Interface](#)

**Other Resources**

[XML Industry Portal](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXErrorHandler Interface

The ISAXErrorHandler/IVBSAXErrorHandler interface provides the basic interface for handling parsing errors.

Methods

- **error** Receives notification of a recoverable error.
- **fatalError** Receives notification of a nonrecoverable error.
- **ignoreableWarning** Receives notification of a warning.

[Visual Basic]

To implement customized error handling, the application must implement this interface and then register an instance with the SAX2 reader, using the `errorHandler` property. The reader then reports all errors and warnings through this interface. If an application does not register an `ErrorHandler`, XML parsing errors are unreported and unexpected behavior can occur.

In the original Java SAX2 implementation, all methods in this interface have only one parameter: `SAXParseException`. However, because exceptions are handled differently in COM, `SAXException` and derived classes are not implemented. Instead, the single `SAXParseException` parameter is replaced with the following methods for SAX2.

If an error occurs during the parse operation, the reader invokes the methods in the `ErrorHandler` class and passes the method the appropriate error information. In general, it's rare that you would call these methods yourself. As such, the syntax shown for the `IVBSAXErrorHandler` interface focuses on implementation syntax, rather than usage syntax.

**Important** The SAX2 reader passes string values to `IVBSAXErrorHandler` methods by reference, rather than by value.

[C/C++]
To implement customized error handling, the application must implement this interface and then register an instance with the SAX2 reader using the putErrorHandler method. The reader then reports all errors and warnings through this interface. If an application does not register an ErrorHandler, XML parsing errors are unreported and unexpected behavior can occur.

In the original Java SAX2 implementation, all methods in this interface have only one parameter: SAXParserException. However, because exceptions are handled differently in COM, SAXException and derived classes are not implemented. For this reason the single SAXParserException parameter is replaced with the following input parameters.

[in] ISAXLocator * pLocator The Locator object, which contains line and column numbers of the error. SystemID and PublicID may be unavailable for particular data sources, such as character buffer.

[in] const wchar_t * pwchErrorMessage The property inherited from the class, org.xml.sax.SAXException. In a COM implementation, provides error information.

[in] HRESULT hrErrorCode The HRESULT code for the reason of the error.

All these methods can return either S_OK code (which matches the Java 'void xyz()' and still aborts parsing for fatal errors), or E_FAIL (which matches throwing a Java exception) and means that parsing should be aborted no matter what level.

Versioning

MSXML 3.0 and later

Requirements

Implementation: msxml5.dll

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib
Remarks

When you use SAX validation, error events are thrown when content is invalid according to the XSD schema used for validation. If the "exhaustive-errors" feature on the SAX reader is set to False (the default), the reader ignores the return value for the error and aborts parsing. If the "exhaustive-errors" feature on the SAX reader is set to True, parsing can continue if S_OK is returned in an error event.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

errorHandler Property (Visual Basic) | putErrorHandler Method (C/C++)
**error Method**

Receives notification of a recoverable error.

[Visual Basic]

**Implementation Syntax**

```
Sub error(oLocator As IVBSAXLocator, strError As String,_
nErrorCode As Long)
```

**Parameters**

`oLocator`
   The Locator object, which contains line and column numbers of the error.
   SystemID and PublicID may be unavailable for particular data sources,
   such as character buffer.

`strError`
   The error information.

`nErrorCode`
   The error code.

[C/C++]

**Syntax**

```
HRESULT error(   
    [in] ISAXLocator * pLocator, 
    [in] const wchar_t * pwchErrorMessage 
    [in] HRESULT hrErrorCode);
```

**Parameters**

`pLocator [in]`
   The Locator object, which contains line and column numbers of the error.
   SystemID and PublicID may be unavailable for particular data sources,
   such as character buffer.
**pwchErrorMessage [in]**

The error information.

**hrErrorCode [in]**

The error code identifying the reason for the error.

### Return Values

**S_OK**

The value returned when the error is handled and parsing may proceed.

**Other**

The value returned if parsing should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

### See Also

[errorHandler Property](#) | [putErrorHandler Method](#) | [ISAXLocator Interface](#)

Applies to: [ISAXErrorHandler/IVBSAXErrorHandler Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
fatalError Method

Receives notification of a nonrecoverable error.

[Visual Basic]

Implementation Syntax

Sub fatalError(oLocator As IVBSAXLocator, strError As String,_
nErrorCode As Long)

Parameters

oLocator
    The Locator object, which contains line and column numbers of the error.
    SystemID and PublicID may be unavailable for particular data sources,
    such as character buffer.
strError
    The error information.
nErrorCode
    The error code.

Remarks

This notification of a nonrecoverable error corresponds to the definition of a
fatal error in section 1.2 of the Worldwide Web Consortium (W3C) XML 1.0
Recommendation. For example, a parser would use this method to report the
violation of a well-formedness constraint. The application must assume that the
document is unusable after the parser has invoked this method. Parsing should
continue only to collect additional error messages. In addition, the parser is free
to stop reporting any other events once this method has been invoked.

[C/C++]

Syntax
HRESULT fatalError(
    [in] ISAXLocator * pLocator,
    [in] const wchar_t * pwchErrorMessage,
    [in] HRESULT hrErrorCode);

Parameters

pLocator [in]
The Locator object, which contains line and column numbers of the error. SystemID and PublicID may be unavailable for particular data sources, such as character buffer.
pwchErrorMessage [in]
The error information.
hrErrorCode [in]
The error code identifying the reason for the error.

Return Values

S_OK
The value returned when the error is handled, but parsing stops for fatal errors.
Other
The value returned if parsing should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

errorHandler Property (Visual Basic) | putErrorHandler Method (C/C++) | ISAXLocator/IVBSAXLocator Interface

Applies to: ISAXErrorHandler/IVBSAXErrorHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ignorableWarning Method

Receives notification of a warning.

**Note** This method is not called in the current Simple API for XML (SAX2) implementation.

[Visual Basic]

**Implementation Syntax**

```vbnet
Sub ignorableWarning(oLocator As IVBSAXLocator, _
    strErrorMessage As String, nErrorCode As Long)
```

**Parameters**

- **oLocator**
  The Locator object, which contains line and column numbers of the error. SystemID and PublicID may be unavailable for particular data sources, such as character buffer.
- **strErrorMessage**
  The error information.
- **nErrorCode**
  The error code.

**Remarks**

The reader uses this method to report conditions that are not errors as defined by the XML 1.0 Recommendation. The default behavior is to take no action. The reader must continue to provide normal parsing events after invoking this method. It should still be possible for the application to process the document through to the end.

[C/C++]

**Syntax**

```c
void ignorableWarning(IVBSAXLocator *oLocator, _
    const char *strErrorMessage, int nErrorCode);
```
HRESULT ignorableWarning(
    [in] ISAXLocator * pLocator,
    [in] const wchar_t * pwchErrorMessage,
    [in] HRESULT hrErrorCode);

Parameters

pLocator [in]
The Locator object, which contains line and column numbers of the error.
SystemID and PublicID may be unavailable for particular data sources,
such as character buffer.
pwchErrorMessage [in]
The error information.
hrErrorCode [in]
The error code identifying the reason for the error.

Return Values

S_OK
The value returned when the error is handled and parsing may proceed.
Other
The value returned if parsing should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

errorHandler Property (Visual Basic) | putErrorHandler Method (C/C++) | ISAXLocator/IVBSAXLocator Interface

Applies to: ISAXErrorHandler/IVBSAXErrorHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXLexicalHandler Interface

The ISAXLexicalHandler/IVBSAXLexicalHandler interface enables a Simple API for XML (SAX2) application to implement an optional extension handler for receiving information from the SAX2 reader, such as comments, the document type declaration, CDATA sections, and the start and end of an entity within a document. The events in the LexicalHandler apply to the entire document, not just to the document elements. All LexicalHandler events must appear between the startDocument and endDocument events of the ContentHandler.

To set the LexicalHandler for the reader, use the putProperty method with the property ID "http://xml.org/sax/properties/lexical-handler".

Methods

- **comment** Receives XML comments from either inside or outside the document element, including comments in the external document type definition (DTD) or schema subset.

- **endCDATA** Receives the end of a character data (CDATA) section.

- **startCDATA** Receives the beginning of a CDATA section.

- **endDTD** Receives the end of a document type declaration.

- **startDTD** Receives the beginning of a document type declaration.

- **endEntity** Indicates that the reader has opened and finished processing an externally defined entity in the content of the document.

- **startEntity** Indicates that the reader has opened and started to process an internally or externally defined entity in the content of the document.

[Visual Basic]

The IVBSAXLexicalHandler interface includes a series of methods that are invoked by the reader during the parsing operation. The reader passes the
appropriate information to the method's parameters. To perform processing for a method, you simply add code to the method in the LexicalHandler implementation. In general, it's rare that you would call these methods yourself from within the LexicalHandler. The syntax shown for the IVBSAXLexicalHandler interface focuses on implementation rather than usage.

**Important**  The SAX2 reader passes string values to IVBSAXContentHandler methods by reference, rather than by value.

[C/C++]

The code to register the LexicalHandler may look like the following.

```c++
... SAXXMLReader * r = ... SAXLexicalHandler * lh = new SAXLexicalHandler (); r->putProperty("http://xml.org/sax/properties/lexical-handler", _vari
```

**Versioning**

MSXML 3.0 and later

**Requirements**

**Implementation:** msxml5.dll

[C/C++]

**Header and LIB files:** msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

endDocument Method | putProperty Method | startDocument Method
comment Method

Receives XML comments from either inside or outside the document element, including comments in the external document type definition (DTD) or schema subset.

Comments in the DTD must be properly nested inside startDTD/endDTD and startEntity/endEntity events (if used).

[Visual Basic]

Implementation Syntax

Sub comment(strChars As String)

Parameters

strChars
   The comment data.

Return Values

If failed, the LexicalHandler raises a trappable error.

[C/C++]

XML comments use the familiar HTML comment notation.

<!-- comments go here. -->

Syntax

HRESULT comment(
   [in] const wchar_t * pwchChars,
   [in] int cchChars);

Parameters
\textit{pwchChars} [in]  
The comment data.

\textit{cchChars} [in]  
The length of the comment string.

**Return Values**

S_OK  
The value returned if the comment data is returned successfully.

E_FAIL  
The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

Applies to: [ISAXLexicalHandler/IVBSAXLexicalHandler Interface]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
endCDATA Method

Receives the end of a character data (CDATA) section.

[Visual Basic]

Implementation Syntax

Sub endCDATA()

Return Values

If failed, the LexicalHandler raises a trappable error.

[C/C++]

The contents of the CDATA section are reported through the characters event. The endCDATA event reports only the end boundary of the CDATA section.

A CDATA section provides a way to include characters such as "<", ">", and "&" in an XML document that would normally interfere with processing. A CDATA section uses the following syntax:

<![CDATA[text goes here]]>.

Syntax

HRESULT endCDATA();

Return Values

S_OK
    The value returned if no errors occur.
E_FAIL
    The value returned if the parse operation should be aborted.
To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

characters Method | startCDATA Method

Applies to: ISAXLexicalHandler/IVBSAXLexicalHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
startCDATA Method

Receives the beginning of a character data (CDATA) section. The contents of the CDATA section are reported through the regular characters event.

[Visual Basic]

This event is intended only to report the boundary.

Implementation Syntax

Sub startCDATA()

Return Values

If successful, returns void.

If failed, the application may raise an exception.

[C/C++]

The startCDATA event reports only the end boundary of the CDATA section.

A CDATA section provides a way to include characters such as "<", ">", and "&" in an XML document that would normally interfere with processing. A CDATA section uses the following syntax:

<![[CDATA[text goes here]]>.

Syntax

HRESULT startCDATA();

Return Values

S_OK
   The value returned if successful.
E_FAIL
   The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

characters Method | endCDATA Method

Applies to: ISAXLexicalHandler/IVBSAXLexicalHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
endDTD Method

Receives the end of a document type declaration. If the document has no such declaration, this method will not be invoked.

[Visual Basic]

Implementation Syntax

Sub endDTD()

Return Values

If failed, the LexicalHandler raises a trappable error.

[C/C++]

A document type declaration connects a document to its document type definition (DTD), which defines the rules of the document, such as what tags can go in your document, what tags can contain other tags, and the number and sequence of the tags. This declaration always begins with "<!DOCTYPE", followed by the name of the DTD, followed by a declaration of the DTD or a link to where the DTD can be found. Finally, it ends with a ">" to close the declaration. For example:

<!DOCTYPE NAME [
  ...DTD content goes here.
  ]>

Syntax

HRESULT endDTD();

Return Values

S_OK
  The value returned if no errors occur.
E_FAIL

The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

startDTD Method

Applies to: ISAXLexicalHandler/IVBSAXLexicalHandler Interface
startDTD Method

Receives the beginning of a document type declaration. If the document has no such declaration, this method will not be invoked.

[Visual Basic]

All declarations reported through DTDHandler or DeclHandler events must appear between the startDTD and endDTD events. Declarations are assumed to belong to the internal document type definition (DTD) subset unless they appear between the startEntity and endEntity events. Comments and processing instructions from the DTD should also be reported between the startDTD and endDTD events.

Note  The startDTD/endDTD events appear within the startDocument/endDocument events of the ContentHandler and before the first startElement event.

Implementation Syntax

Sub startDTD(strName As String, strPublicId As String, strSystemId As String)

Parameters

strName  The document type name.
strPublicId  The declared public identifier for the external DTD subset or Null (if none was declared).
strSystemId  The declared system identifier for the external DTD subset or Null (if none was declared).

Return Values
If failed, the **LexicalHandler** raises a trappable error.

[C/C++]

**Note** Any declarations are assumed to be in the internal subset unless otherwise indicated by a **startEntity** event.

The **startDTD** method is called within the **startDocument** and **endDocument** events from the **ContentHandler** and before the first **startElement** event.

A document type declaration connects a document to its document type definition (DTD), which defines the rules of the document, such as what tags can go in your document, what tags can contain other tags, and the number and sequence of the tags. This declaration always begins with "<!DOCTYPE", followed by the name of the DTD, followed by a declaration of the DTD or a link to where the DTD can be found. Finally, it ends with a ">" to close the declaration. For example:

```
<!DOCTYPE NAME [ ...DTD content goes here. ]>
```

**Syntax**

```plaintext
HRESULT startDTD(
    [in] const wchar_t * pwchName,
    [in] int cchName,
    [in] const wchar_t * pwchPublicId,
    [in] int cchPublicId,
    [in] const wchar_t * pwchSystemId,
    [in] int cchSystemId);
```

**Parameters**

- **pwchName [in]**
  The document type name.
- **cchName [in]**
  The length of the document type's name.
- **pwchPublicId [in]**
  The declared public identifier for the external DTD subset or Null (if none was declared).
cchPublicId [in]
The length of the public identifier.

pwchSystemId [in]
The declared system identifier for the external DTD subset or Null (if none was declared).

ccchSystemId [in]
The length of the system identifier.

**Return Values**

S_OK
The value returned if no errors occur.

E_FAIL
The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

endDocument Method | endDTD Method | startDocument | startElement Method | startEntity Method

Applies to: ISAXLexicalHandler/IVBSAXLexicalHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
endEntity Method

Indicates that the reader has opened and finished processing an externally defined entity in the content of the document.

[Visual Basic]

Implementation Syntax

Sub endEntity(strName As String)

Parameters

strName
The name of the entity.

Return Values

If failed, the LexicalHandler raises a trappable error.

[C/C++]

Syntax

HRESULT EndEntity(
    [in] const wchar_t * pwchName,
    [in] int cchName);

Parameters

pwchName [in]
The name of the entity. For parameter entities, the name begins with "\%".

cchName [in]
The length of the entity name.

Return Values
S_OK
   The value returned if no errors occur.

E_FAIL
   The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

startEntity Method

Applies to: ISAXLexicalHandler/IVBSAXLexicalHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
startEntity Method

Indicates that the reader has opened and started to process an internally or externally defined entity in the content of the document.

[Visual Basic]

The reporting of parameter entities, including any external document type definition (DTD) subset, is optional. SAX2 (Simple API for XML) drivers that support a LexicalHandler may not support reporting of entities. Use the "http://xml.org/sax/features/lexical-handler/parameter-entities" feature to query or control the reporting of parameter entities.

The reader reports general entities with their regular names, parameter entities with '%' prepended to their names, and any external DTD subset with the pseudo-entity name "[dtd]".

When a reader driver provides these events, all other events must be properly nested within startEntity/endEntity events. There is no additional requirement that events from DeclHandler or DTDHandler be properly ordered.

**Note** The skippedEntity event, which is part of the ContentHandler interface, reports skipped entities.

Because of the streaming event model that SAX uses, some entity boundaries cannot be reported under any circumstances:

- General entities within attribute values.
- Parameter entities within declarations.

These are silently expanded, with no indication of where the original entity boundaries were.

Furthermore, the boundaries of character references (which are not really entities anyway) are not reported.

**Implementation Syntax**
Sub startEntity(strName As String)

**Parameters**

*strName*

The name of the entity. If it is a parameter entity, the name begins with '%'. If the parameter is an external DTD subset, it uses "[dtd]".

**Return Values**

If failed, the LexicalHandler raises a trappable error.

[C/C++]

The reader never reports entity references in attribute values, as well as the start and end of the document entity. In addition, the reader reports the start and end of the external document type definition (DTD) subset using the pseudo-name "[dtd]". All other events must be properly nested within the startEntity/endEntity events. Note that skipped entities will be reported through the skippedEntity event.

Entities are references to other content, which can be a piece of text or a large external file. Declaring an entity is done within a DTD. Entities always have two parts, the declaration and the reference. For example, consider the following entity reference:

```xml
<!ENTITY CompName "Microsoft® Corporation">
```

This entity can be referenced as follows:

```xml
&amp;CompName; is an industry leader.
```

Which is interpreted as:

```xml
Microsoft® Corporation is an industry leader.
```

XML offers two kinds of entities — general entities and parameter entities. General entities, defined in DTDs, substitute the entity value within a document, as shown in the preceding example. Parameter entities link the content of additional DTD files to a DTD, or act as an abbreviation for frequently repeated
declarations. To distinguish between these two types of entities, general entities use an ampersand (&) while parameter entities use the percent sign (%).

The reporting of parameter entities (including the DTD subset) is strictly optional. To allow the reader to recognize external entities, you use the `putFeature` method of the `ISAXXMLReader` interface to set the "http://xml.org/sax/features/lexical-handler/external-general-entities" and "http://xml.org/sax/features/lexical-handler/external-parameter-entities" features. The reader reports the general entities with their regular names, parameter entities have "%" prepended to their names, and the external DTD subset has the pseudo-entity name "[dtd]". When the reader reports these events, all other events must be properly nested within `startEntity` and `endEntity` events.

**Syntax**

```c
HRESULT startEntity(
    [in] const wchar_t * pwchName,
    [in] int cchName);
```

**Parameters**

- `pwchName [in]` The name of the entity. For parameter entities, the name begins with "%".
- `cchName [in]` The length of the entity name.

**Return Values**

- `S_OK` The value returned if no errors occur.
- `E_FAIL` The value returned if the parse operation should be aborted.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**
endEntity Method | ISAXXMLReader/IVBSAXXMLReader Interface | putFeature Method

Applies to: ISAXLexicalHandler/IVBSAXLexicalHandler Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXLocator Interface

The ISAXLocator/IVBSAXLocator interface associates a SAX2 event with a document location.

[Visual Basic]

The SAX2 reader provides location information by implementing this interface and passing an instance to the application, using the documentLocator property of the IVBSAXContentHandler interface. The application can use the Locator object to get the property values defined for this interface. These values describe location information associated with the current event, such as system ID, public ID, line number, and column number of any ContentHandler event in the XML source document.

Properties

- **columnNumber**
  Returns the column number where the current document event ends.

- **lineNumber**
  Returns the line number where the current document event ends.

- **publicId**
  Returns the public identifier for the current document event.

- **systemId**
  Returns the system identifier for the current document event.

[C/C++]

The SAX2 reader provides location information by implementing this interface and passing an instance to the application using the putDocumentLocator method of the ISAXContentHandler interface. The application can use the Locator object to obtain the location of any other document handler event in the XML source document.

The Locator object has a one-to-one relationship with the reader. As a result, each new event updates the information inside the Locator object. For example,
an application saves a reference to the Locator object for handler call A. After that, the application tries to access the Locator object in handler call B. The information provided by the Locator object is now specific to handler call B because the object updates with each event. Correct usage for the Locator object is to save a reference to it within the ContentHandler and access it from there. For ErrorHandler methods, the Locator object is passed as a parameter.

**Methods**

- **getColumnNumber**: Returns the column number where the current document event ends.
- **getLineNumber**: Returns the line number where the current document event ends.
- **getPublicId**: Returns the public identifier for the current document event.
- **getSystemId**: Returns the system identifier for the current document event.

**Versioning**

MSXML 3.0 and later

**Requirements**

**Implementation**: msxml5.dll

[C/C++]

**Header and LIB files**: msxml2.h, msxml2.lib

**Remarks**

Results returned by the object are valid only during the scope of each document handler method. The application receives unpredictable results if it attempts to use the locator at any other time. If the reader supplies a locator, it must do so before reporting any other document events. If no locator is set by the time the application receives the startDocument method event, the application should
assume that a locator is not available.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

documentLocator Property (Visual Basic) | putDocumentLocator Method (C/C++) | ISAXContentHandler Interface | startDocument Method
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**columnNumber Property (Visual Basic)**

Returns the column number where the current document event ends.

This is the number of the first character following the current document event. The first column in a line is position 1.

**Implementation Syntax**

```
Property columnNumber As Long
```

**Usage Syntax**

```
NumVal=IVBSAXLocator.columnNumber
```

**Return Values**

If failed, returns -1 and raises a trappable error.

**Remarks**

The `columnNumber` property is read-only.

**See Also**

[IFnBSAXProperty (Visual Basic)]

Applies to: [IVBSAXLocator Interface]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**lineNumber Property (Visual Basic)**

Returns the line number where the current document event ends.

This is the number of the line in which the first character following the current document event appears.

**Implementation Syntax**

```
Property lineNumber As Long
```

**Usage Syntax**

```
NumVal = IVBSAXLocator.lineNumber
```

**Return Values**

If failed, returns -1 and raises a trappable error.

**Remarks**

The `lineNumber` property is read-only.

**See Also**

[columnNumber Property (Visual Basic)]

Applies to: [IVBSAXLocator Interface]
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**publicId Property (Visual Basic)**

Returns the public identifier for the current document event. A public identifier identifies a public domain file located in a publicly accessible place.

**Implementation Syntax**

Property publicId As String

**Usage Syntax**

StrVal = IVBSAXLocator.publicId

**Return Values**

Null if a public identifier is not available.

**Remarks**

The publicID property is read-only.

**See Also**

systemId Property (Visual Basic)

Applies to: IVBSAXLocator Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**systemId Property (Visual Basic)**

Returns the system identifier for the current document event. A system identifier represents a file located on the local file system or network.

**Implementation Syntax**

Property `systemId` As String

**Usage Syntax**

```vbscript
StrVal = IVBSAXLocator.publicId
```

**Return Values**

Null if a system identifier is not available.

**Remarks**

The `systemId` property is read-only.

**See Also**

- [publicId Property (Visual Basic)](#)

Applies to: [IVBSAXLocator Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getColumnName Method (C/C++)**

Returns the column number where the current document event ends.

This is the number of the first character after the text associated with the document event. The `getColumnName` method is permitted to show position within the text associated with the text event.

**Syntax**

```c
HRESULT getColumnNumber(
    [out, retval] int * pnColumn);
```

**Parameters**

`pnColumn` [out, retval]

The column number (the first column in a line is #1). The `pnColumn` is –1 if the Locator object is not associated with some source input or a position within the source input.

**Return Values**

S_OK

The value returned if column number returned successfully.

**See Also**

[getLineNumber Method (C/C++)](#)

Applies to: [ISAXLocator Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getLineNumber Method (C/C++)

Returns the line number where the current document event ends.

This is the line position of the first character after the text associated with the document event. This method is permitted to show position within the text associated with the text event.

Syntax

```c
HRESULT getLineNumber(
    [out, retval] int * pnLine);
```

Parameters

`pnLine [out, retval]`
The line number. The `pnLine` is –1 if the Locator object is not associated with some source input or a position within the source input.

Return Values

S_OK
The value returned if line number returned successfully.

See Also

getColumnNumber Method (C/C++)

Applies to: ISAXLocator Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getPublicId Method (C/C++)**

Returns the public identifier for the current document event. A public identifier identifies a public domain file located in a publicly accessible place.

**Syntax**

```cpp
HRESULT getPublicId(
    [out, retval] const wchar_t ** ppwchPublicId);
```

**Parameters**

*ppwchPublicId* [out, retval]

The public identifier or Null (if none is available).

**Return Values**

*S_OK*

The value returned if the public identifier is returned successfully; Null if no public identifier is available.

**See Also**

*getSystemId Method (C/C++)*

Applies to: ISAXLocator Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getSystemId Method (C/C++)**

Returns the system identifier for the current document event. A system identifier represents a file located on the local file system or network.

**Syntax**

```c
HRESULT getSystemId(
    [out, retval] const wchar_t ** ppwchSystemId);
```

**Parameters**

- `ppwchSystemId [out, retval]`
  - The system identifier or Null (if none is available).

**Return Values**

- **S_OK**
  - The value returned if the system identifier is returned successfully; Null if no system identifier is available.

**See Also**

- [getPublicId Method (C/C++)](#)

Applies to: [ISAXLocator Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXXMLFilter Interface

The ISAXXMLFilter/IVBSAXXMLFilter interface provides a convenient base for creating an XML filter by passing on all ISAXEntityResolver/IVBSAXEntityResolver, ISAXDTDHandler/IVBSAXDTDHandler, ISAXContentHandler/IVBSAXContentHandler, and ISAXErrorHandler/IVBSAXErrorHandler events automatically. An XML filter is like an XML reader, except that it obtains its events from another XML reader rather than a primary source, such as an XML document or a database. Filters can modify a stream of events as they pass on to their final application.

[Visual Basic]

The IVBSAXXMLFilter interface is derived from the IDispatch interface, not the IVBSAXXMLReader interface.

Properties

parent

Sets or gets the instance of the reader that is the parent of the IVBSAXXMLFilter.

[C/C++]

Methods

getParent

Gets a reference to the instance of ISAXXMLReader that was set by the putParent method.

putParent

Sets the parent of the filter to an instance of ISAXXMLReader.

Versioning

MSXML 3.0, unimplemented
Requirements

Implementation: msxml5.dll

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISAXContentHandler | ISAXDTDHandler | ISAXEntityResolver | ISAXErrorHandler
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getParent Method (C/C++)**

Gets a reference to the instance of ISAXXMLReader that was set by the putParent method.

**Syntax**

```c
HRESULT getParent(
    [out, retval] ISAXXMLReader ** ppReader);
```

**Parameters**

`ppReader` [out, retval]
- The instance of the reader in use.

**Remarks**

The argument may not be Null.

**Return Values**

- **S_OK**
  - The value returned when the parent is returned as expected.

**See Also**

- [putParent Method (C/C++)](#)

Applies to: [ISAXXMLFilter Interface](#)
putParent Method (C/C++)

Sets the parent of the filter to an instance of ISAXXMLReader.

Syntax

HRESULT putParent(
    [in] ISAXXMLReader * pReader);

Parameters

*pReader [in]
    The instance of the ISAXXMLReader to use as the parent.

Return Values

S_OK
    The value returned when the instance of ISAXXMLReader is set as expected.

Remarks

This method allows the application to link the filter to a parent reader (which may be another filter). The argument may not be Null.

See Also

getParent Method (C/C++)

Applies to: ISAXXMLFilter Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**parent Property (Visual Basic)**

Sets or gets the instance of the reader that is the parent of the IVBSSAXXMLFilter.

**Implementation Syntax**

Property parent As SAXXMLReader

**Usage Syntax**

To set the parent property

Set oIVBSAXFilter.parent = oSAXXMLReader

To get the parent property

Set oSAXXMLReader = oIVBSAXFilter.parent

**Remarks**

The parent property is read/write.

**See Also**

Applies to: [IVBSAXXMLFilter Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
ISAXXMLReader Interface

The ISAXXMLReader/IVBSAXXMLReader interface allows an application to set and query features and properties in Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office, register event handling for document processing, and initiate a document parse. MSXML provides the following CoClass that implements this interface.

**SAXXMLReader50**
- The version-dependent CoClass ties the application to msxml5.dll.

(CLSID_SAXXMLReader50: 88d969c7-f192-11d4-a65f-0040963251e5)

With MSXML 4.0 and later, you can throw SAX events from a DOMDocument object by specifying the DOMDocument object as the input parameter for the parse method. For more information, see Convert DOM to SAX.

**Controlling Reader Behavior**

You can use the following methods to observe and control the behavior of the reader:

- **getFeature**
  - Allows an application to ask the reader if it supports a feature.
- **putFeature**
  - Allows an application to request that the reader turn a feature on or off.

**Supported Features and Properties**

The Microsoft COM/Visual Basic® implementation of SAX2 supports the following features (for example, a property whose value is Boolean).

- "exhaustive-errors"
- "http://xml.org/sax/features/external-general-entities"
- "http://xml.org/sax/features/external-parameter-entities"
- "http://xml.org/sax/features/lexical-handler/parameter-
The reader recognizes the "http://xml.org/sax/features/string-interning" and "schema-validation" features. However, you cannot set these features in the current SAX2 implementation.

The reader also provides methods (getProperty, putProperty) for getting and setting these properties.

- "http://xml.org/sax/properties/lexical-handler"
- "http://xml.org/sax/properties/declaration-handler"
- "http://xml.org/sax/properties/dom-node"
- "http://xml.org/sax/properties/xml-string"
- "charset"
- "schemas"
- "schema-declaration-handler"
- "xmldecl-encoding"
- "xmldecl-version"
- "xmldecl-standalone"

**Methods**

- `getFeature` Returns the Boolean value of a feature.
- `putFeature` Sets the value of a feature.
- `getProperty` Returns the value of a property (as a string), for any qualified Uniform Resource Identifier (URI).
- `putProperty` Sets the value of a property.
- `parse` Parses an XML document or a DOMDocument object.
- `parseURL` Parses an XML document from a system identifier.
The IVBSAXXMLReader interface is implemented in the Simple API for XML (SAX2) applications to manage and execute the parsing of an XML document.

**Creating an Instance of the Reader**

Typically, you create an instance of the reader in the Main form of a Visual Basic application. For example, first you declare a variable for holding a reference to the IVBSAXXMLReader50 interface. You then create an instance of the reader and assign the interface to the object variable as follows:

```vba
Dim reader As SAXXMLReader50
Set reader = New SAXXMLReader50
```

**Creating an Instance of a Handler**

After you create an instance of the reader, you can create an instance of a ContentHandler or ErrorHandler, and then set them as properties of the reader object using the contentHandler and errorHandler properties.

The following code sample declares variables for the ContentHandler and ErrorHandler implementations, creates instances of these two handlers, and registers these handlers with the reader object by defining them as properties of the reader.

```vba
Dim reader As SAXXMLReader50
Dim contentHandler As ContentHandlerImpl
Dim errorHandler As ErrorHandlerImpl

Set reader = New SAXXMLReader50
Set contentHandler = New ContentHandlerImpl
Set errorHandler = New ErrorHandlerImpl
Set reader.contentHandler = contentHandler
Set reader.errorHandler = errorHandler
```

**Properties**

- **baseURL**
  - Sets the base URL for the current document.
- **contentHandler**
  - Sets the current ContentHandler.
- **dtdHandler**
  - Sets the current DTDHandler.
- **entityResolver**
  - Sets the current EntityResolver.
**errorHandler** Registers or returns the current ErrorHandler.

**secureBaseURL** Sets the secure base URL for the document.

[C/C++]

**Registering Handlers**

An application can set and query features in the reader, register event handlers for document processing, and initiate a document parse. With this COM/C++ implementation of SAX2, the reader can use the following methods to register the corresponding handlers:

- **putContentHandler** Registers a ContentHandler.
- **putDTDHandler** Registers a DTDHandler.
- **putErrorHandler** Registers an ErrorHandler.

**Methods**

In addition to the preceding methods, the following also apply to the ISAXXMLReader interface.

- **getBaseURL** Returns the base URL for the document.
- **putBaseURL** Sets the base URL for the document.
- **getContentHandler** Returns the current ContentHandler.
- **putContentHandler** Allows an application to register a ContentHandler.
- **getDTDHandler** Returns the current DTDHandler.
- **putDTDHandler** Allows an application to register a DTDHandler.
- **getEntityResolver** Returns the current EntityResolver.
- **putEntityResolver** Allows an application to register an EntityResolver.
- **getErrorHandler** Returns the current ErrorHandler.
- **putErrorHandler** Allows an application to register an ErrorHandler.
- **getSecureBaseURL** Returns the secure base URL for the document.
- **putSecureBaseURL** Sets the secure base URL for the document.
Versioning

MSXML 3.0 and later

Requirements

Implementation: msxml5.dll, msxml2.lib

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

Version 5.0 Dependent ProgID: SAXXMLReader50

Version 5.0 Dependent CLSID: 88d969c7-f192-11d4-a65f-0040963251e5

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getFeature Method

Returns the Boolean value of a feature.

The Microsoft® COM/Visual Basic® and COM/C++ implementation support the following features:

- "exhaustive-errors"
- "http://xml.org/sax/features/external-general-entities"
- "http://xml.org/sax/features/external-parameter-entities"
- "http://xml.org/sax/features/lexical-handler/parameter-entities"
- "http://xml.org/sax/features/namespaces"
- "http://xml.org/sax/features/namespace-prefixes"
- "preserve-system-identifiers"
- "schema-validation"
- "server-http-request"

[Visual Basic]

The Boolean value indicates if a feature is enabled or disabled. For example, setting the "http://xml.org/sax/features/namespaces" feature to True enables namespace processing during parsing.

Usage Syntax

strFeature = oSAXXMLReader.getFeature(strName)

Parameters

strName
    The name of the current feature.

Return Values

The following table shows the current matrix of behavior for the getFeature
method.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>external-general-entities</td>
<td></td>
</tr>
<tr>
<td>external-parameter-entities</td>
<td></td>
</tr>
<tr>
<td>exhaustive-errors</td>
<td></td>
</tr>
<tr>
<td>namespaces</td>
<td></td>
</tr>
<tr>
<td>namespace-prefixes</td>
<td></td>
</tr>
<tr>
<td>parameter-entities</td>
<td></td>
</tr>
<tr>
<td>preserve-system-identifiers</td>
<td></td>
</tr>
<tr>
<td>schema-validation</td>
<td>Boolean (True/False). If successful, returns the value of the feature.</td>
</tr>
<tr>
<td>server-http-request</td>
<td></td>
</tr>
<tr>
<td>Other features</td>
<td>Trappable error.</td>
</tr>
</tbody>
</table>

**Remarks**

The IVBSAXXMLReader interface recognizes the following features:

"exhaustive-errors"

- **True** — Report all validation errors.
- **False** — (default) Report only basic validation errors.
- **Access** — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/external-general-entities"

- **True** — Include all external general (text) entities.
- **False** — (default) Do not include external general entities.
- **Access** — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/external-parameter-entities"

- **True** — Include all external parameter entities, including the external document type definition (DTD) subset.
False — (default) Do not include any external parameter entities, including the external DTD subset.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/lexical-handler/parameter-entities"

True — (default) For all parameter entities, all ISAXLexicalHandler event methods events are fired.

False — For all parameter entities, the following ISAXLexicalHandler event methods are not fired: startEntity, endEntity, skippedEntity. This includes parameter entities defined in external document type definition (DTD) subsets.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/namespaces"

True — (default) Makes namespace URIs and local names available during various method callbacks. For example, callbacks made using the startElement and endElement methods of the ISAXContentHandler interface, or the methods of the ISAXAttributes interface. In addition, namespace declarations are returned by the startPrefixMapping and endPrefixMapping methods of the ISAXContentHandler interface.

False — Does not make namespace URIs and local names available during various method callbacks.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/namespace-prefixes"

True — (default) Namespace declarations are ignored and namespace-prefixed attributes are treated the same as other attributes.

False — Namespace declarations are omitted from the attribute collection, provided that the namespace feature is True.

Access (parsing) Read-only. (not parsing) Read/write.
"preserve-system-identifiers"

True — System identifiers reported by the reader are not resolved. This applies to all system identifiers used with any lexical, declaration, or DTD handlers, as well as any entity resolvers that might be implemented. Internally, however, the system identifiers are still resolved.

False — (default) System identifiers reported by the reader are resolved.

Access — (parsing) Read-only; (not parsing) Read/write.

"schema-validation"

True — Include all external general (text) entities.

False — (default) Do not validate when parsing.

Access — (parsing) Read-only; (not parsing) Read/write.

"server-http-request"

True — Use the server-safe ServerXMLHTTP object for the parseURL method.

False — (default) Use WinInet component to get data through HTTP for parseURL.

Access — (parsing) Read-only; (not parsing) Read/write.

[C/C++]

The feature name is any fully-qualified Uniform Resource Identifier (URI). The values for the namespaces and namespace-prefixes features may be available only in a specific context, such as before, during, or after a parse.

Syntax

HRESULT getFeature(
    [in] const wchar_t * pwchName,
    [out, retval] VARIANT_BOOL * pvfValue);
Parameters

\textit{pwchName} [in]

The feature name (zero-terminated Unicode string).

\textit{pvfValue} [out, retval]

The value of the feature (True or False).

Return Values

The following table lists the return values for the \texttt{getFeature} method.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>exhaustive-errors</td>
<td>S_OK — Last successful putFeature argument.</td>
</tr>
<tr>
<td>external-general-entities</td>
<td></td>
</tr>
<tr>
<td>external-parameter-entities</td>
<td></td>
</tr>
<tr>
<td>namespaces</td>
<td>S_OK — False</td>
</tr>
<tr>
<td>namespace-prefixes</td>
<td></td>
</tr>
<tr>
<td>parameter-entities</td>
<td></td>
</tr>
<tr>
<td>preserve-system-identifiers</td>
<td>E_INVALIDARG</td>
</tr>
<tr>
<td>server-http-request</td>
<td></td>
</tr>
<tr>
<td>schema-validation</td>
<td></td>
</tr>
<tr>
<td>Other features</td>
<td></td>
</tr>
</tbody>
</table>

Remarks

The following describes the settings and access for the features:

"http://xml.org/sax/features/namespaces"

True — (default) Makes namespace URIs and local names available during various method callbacks. For example, callbacks made using the \texttt{startElement} and \texttt{endElement} methods of the \texttt{ISAXContentHandler} interface, or the methods of the \texttt{ISAXAttributes} interface. In addition, \texttt{namespace declarations} are returned by the \texttt{startPrefixMapping} and \texttt{endPrefixMapping} methods of the \texttt{ISAXContentHandler} interface.

False — Does not make namespace URIs and local names available during various method callbacks.

Access — (parsing) Read-only; (not parsing) Read/write.
"http://xml.org/sax/features/namespace-prefixes"

True — (default) Namespace declarations are ignored and namespace-prefixed attributes are treated the same as other attributes.

False — Namespace declarations are omitted from the attribute collection, provided that the namespace feature is True.

Access (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/external-general-entities"

True — Include all external general (text) entities.

False — (default) Do not include external general entities.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/external-parameter-entities"

True — Include all external parameter entities, including the external document type definition (DTD) subset.

False — (default) Do not include any external parameter entities, including the external DTD subset.

"exhaustive-errors"

True — Report all validation errors.

False — (default) Report only basic validation errors.

Access — (parsing) Read-only; (not parsing) Read/write.

"schema-validation"

True — Include all external general (text) entities.

False — (default) Do not validate when parsing.
Access — (parsing) Read-only; (not parsing) Read/write.

"server-http-request"

True — Use the server-safe ServerXMLHTTP object for the parseURL method.

False — (default) Use the WinInet component to get data through HTTP for parseURL.

Access — (parsing) Read-only; (not parsing) Read/write.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

putFeature Method

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
putFeature Method

Sets the value of a feature.

Allows an application to control the behavior of a reader by setting the state of the supported features. The feature name is any qualified Uniform Resource Identifier (URI). The ISAXXMLReader interface recognizes the following features:

- "exhaustive-errors"
- "http://xml.org/sax/features/external-general-entities"
- "http://xml.org/sax/features/external-parameter-entities"
- "http://xml.org/sax/features/lexical-handler/parameter-entities"
- "http://xml.org/sax/features/namespaces"
- "http://xml.org/sax/features/namespace-prefixes"
- "preserve-system-identifiers"
- "schema-validation"
- "server-http-request"

The values for the namespaces and namespace-prefixes features may be available only in a specific context, such as before, during, or after a parse.

[Visual Basic]

Usage Syntax

oSAXXMLReader.putFeature(strName, vfValue)

Parameters

strName
The name of the current feature.

vfValue
A Boolean expression (True/False) specifying whether the feature is on or off.
Return Values

The following table shows the return values for the `putFeature` method.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>exhaustive-errors</td>
<td></td>
</tr>
<tr>
<td>external-general-entities</td>
<td></td>
</tr>
<tr>
<td>external-parameter-entities</td>
<td></td>
</tr>
<tr>
<td>namespaces</td>
<td></td>
</tr>
<tr>
<td>namespace-prefixes</td>
<td>S_OK (not parsing and legal)</td>
</tr>
<tr>
<td>parameter-entities</td>
<td>E_FAIL if namespace-prefixes is False. (parsing or illegal)</td>
</tr>
<tr>
<td>preserve-system-identifiers</td>
<td></td>
</tr>
<tr>
<td>schema-validation</td>
<td></td>
</tr>
<tr>
<td>server-http-request</td>
<td></td>
</tr>
<tr>
<td>Other features</td>
<td>Trappable error</td>
</tr>
</tbody>
</table>

[C/C++]

Syntax

```c
HRESULT putFeature(
    [in] const wchar_t * pwchName,
    [in] VARIANT_BOOL vfValue);
```

Parameters

`pwchName` [in]
The name of the feature (zero-terminated Unicode string).

`vfValue` [in]
The current state of the feature (True or False).

Return Values

The following table lists the return values for the `putFeature` method.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaces</td>
<td>S_OK (not parsing and legal)</td>
</tr>
<tr>
<td></td>
<td>E_FAIL if namespace-prefixes is False. (parsing or illegal)</td>
</tr>
</tbody>
</table>
Remarks

The following describes the possible settings and access for features:

"exhaustive-errors"

True — Report all validation errors.

False — (default) Report only basic validation errors.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/external-general-entities"

True — Include all external general (text) entities.

False — (default) Do not include external general entities.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/external-parameter-entities"

True — Include all external parameter entities, including the external
document type definition (DTD) subset.

False — (default) Do not include any external parameter entities, including the external DTD subset.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/lexical-handler/parameter-entities"

True — (default) For all parameter entities, all ISAXLexicalHandler event methods events are fired.

False — For all parameter entities, the following ISAXLexicalHandler event methods are not fired: startEntity, endEntity, skippedEntity. This includes parameter entities defined in external document type definition (DTD) subsets.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/namespaces"

True — (default) Makes namespace URIs and local names available during various method callbacks. For example, callbacks made using the startElement and endElement methods of the ISAXContentHandler interface, or the methods of the ISAXAttributes interface. In addition, namespace declarations are returned by the startPrefixMapping and endPrefixMapping methods of the ISAXContentHandler interface.

False — Does not make namespace URIs and local names available during various method callbacks.

Access — (parsing) Read-only; (not parsing) Read/write.

"http://xml.org/sax/features/namespace-prefixes"

True — (default) Namespace declarations are ignored and namespace-prefixed attributes are treated the same as other attributes.

False — Namespace declarations are omitted from the attribute collection, provided that the namespace feature is True.
Access  (parsing) Read-only. (not parsing) Read/write.

"preserve-system-identifiers"

True — System identifiers reported by the reader are not resolved. This applies to all system identifiers used with any lexical, declaration, or DTD handlers, as well as any entity resolvers that might be implemented. Internally, however, system identifiers are still resolved.

False — (default) System identifiers reported by the reader are resolved.

Access — (parsing) Read-only; (not parsing) Read/write.

"schema-validation"

True — Include all external general (text) entities.

False — (default) Do not validate when parsing.

Access — (parsing) Read-only; (not parsing) Read/write.

"server-http-request"

True — Use the server-safe ServerXMLHTTP object for the parseURL method.

False — (default) Use the WinInet component to get data through HTTP for parseURL.

Access — (parsing) Read-only; (not parsing) Read/write.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

getFeature Method

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getProperty Method

Returns a property's value (as a string), for any qualified Uniform Resource Identifier (URI).

[Visual Basic]

Usage Syntax

varValue = oSAXXMLReader.getProperty(strName)

Parameters

varValue

The corresponding data type of the property.

Return Values

The following table shows the return values for the getProperty method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration-handler</td>
<td>Current value</td>
</tr>
<tr>
<td>lexical-handler</td>
<td></td>
</tr>
<tr>
<td>schemas</td>
<td></td>
</tr>
<tr>
<td>schema-declaration-handler</td>
<td></td>
</tr>
<tr>
<td>charset</td>
<td></td>
</tr>
<tr>
<td>dom-node</td>
<td></td>
</tr>
<tr>
<td>xmldecl-encoding</td>
<td>Either the current value (when parsing) or a trappable error (when not parsing).</td>
</tr>
<tr>
<td>xmldecl-version</td>
<td></td>
</tr>
<tr>
<td>xmldecl-standalone</td>
<td></td>
</tr>
<tr>
<td>xml-string</td>
<td></td>
</tr>
<tr>
<td>Other properties</td>
<td>Trappable error.</td>
</tr>
</tbody>
</table>

[C/C++]

Returns the value of a property. The property name is any qualified Uniform Resource Identifier (URI).
It is possible for a reader to recognize a property name but not be able to return its state. Although a core set of properties is defined, a reader is not required to recognize specific property names. Some property values may be available only in a specific context, such as before, during, or after a parse.

**Syntax**

```
HRESULT getProperty(
    [in] const wchar_t * pwchName,
    [out, retval] VARIANT * pvarValue);
```

**Parameters**

- **pwchName [in]**  
  The property's name (zero-terminated Unicode string).
- **pvarValue [out, retval]**  
  The corresponding data type for the property.

**Return Values**

The following table lists the return values for the `getProperty` method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration-handler</td>
<td>S_OK — The current value is returned.</td>
</tr>
<tr>
<td>lexical-handler</td>
<td></td>
</tr>
<tr>
<td>schemas</td>
<td>S_OK — The current value is returned.</td>
</tr>
<tr>
<td>schema-declaration-handler</td>
<td></td>
</tr>
<tr>
<td>charset</td>
<td>E_FAIL — Not parsing.</td>
</tr>
<tr>
<td>dom-node</td>
<td></td>
</tr>
<tr>
<td>xmldecl-encoding</td>
<td></td>
</tr>
<tr>
<td>xmldecl-version</td>
<td></td>
</tr>
<tr>
<td>xmldecl-standalone</td>
<td></td>
</tr>
<tr>
<td>xml-string</td>
<td></td>
</tr>
<tr>
<td>Other properties</td>
<td>E_INVALIDARG — The property is not recognized.</td>
</tr>
</tbody>
</table>

**Remarks**

The ISAXXMLReader interface recognizes the following properties:
"charset"

Externally defined encoding to use with an XML document. An example is the one used in the MIME-header. This setting takes priority over the default encoding, which is implicitly UTF-16, or over the encoding specified in the byte order mark (BOM) of the XML document header.

This property only affects documents that are parsed using the `parse` method. For any documents that are parsed using the `parseURL` method, this property is ignored. This property can be set at any time, but it only affects documents that are opened after it is set. For example, if it is set in the `resolveEntity` method (which is available when you implement a SAX entity resolver), it affects the external entity to be opened, but does not affect the main document, which is currently being parsed.

Datatype — BSTR* (The string value must be a recognized description of a character set that is valid for use with XML. For example, "US-ASCII", "UTF-8", etc.)

Default — Empty (uses either the BOM in XML header or the default encoding)

Access — Read/write.

"http://xml.org/sax/properties/lexical-handler"

This is an optional extension handler for lexical events like comments.

Datatype — ISAXXMLLexicalHandler.

Access — Read/write.

"http://xml.org/sax/properties/declaration-handler"

This is an optional extension handler for declaration events.

Datatype — ISAXXMLDeclHandler.

Access — Read/write.

"http://xml.org/sax/properties/dom-node"
Source DOM node for current SAX event. Only available when using parse method and SAX events are generated from an IXMLDOMNode object. Not available when using parseURL method for SAX parsing of a file.

Datatype — IXMLDOMNode.

Access — Read-only.

"http://xml.org/sax/properties/xml-string"

The literal string of characters in the XML source for the current SAX event.

Datatype — String (Visual Basic) / w_char* (C/C++)

Access — Read-only.

"schemas"

An option for registering schemas with the SAX reader.

Datatype — XMLSchemaCache.

Access — (parsing) Read-only; (not parsing) Read/Write.

"schema-declaration-handler"

An option for registering a schema-based declaration handler with the SAX reader.

Datatype — Event handler that implements the IMXSchemaDeclHandler interface.

Access — Read/write.

"xmldecl-encoding"

Encoding declared in the XML header of the currently parsed document. This data is unavailable until the reader parses the XML header.

Datatype — BSTR*.
Default — Empty (if there is no XML declaration or if a property is not found).

Access — (parsing) Read; (not parsing) None.

"xmldecl-version"

Version declared in the XML header of currently parsed document. This data is unavailable until the reader parses the XML header.

Datatype — BSTR*.

Default — Empty (if there is no XML declaration).

Access — (parsing) Read; (not parsing) None.

"xmldecl-standalone"

Declared in the XML header of currently parsed document. This data is unavailable until the reader parses the XML header.

Datatype — BSTR* ("yes" or "no").

Default — Empty (if there is no XML declaration or if a property is not found).

Access — (parsing) Read; (not parsing) None.

**Note**  The "xmldecl-encoding", "xmldecl-version", and "xmldecl-standalone" properties provide information about the presence and content of the XML header. However, XML header information was designed for low-level reader and parser use, not for applications.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

putProperty Method

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**putProperty Method**

Sets the value of a property. The property name is any qualified Uniform Resource Identifier (URI).

**[Visual Basic]**

**Usage Syntax**

oSAXXMLReader.putProperty(strName, varValue)

**Parameters**

*strName*  
The property's name.

*varValue*  
The corresponding data type of the property.

**Return Values**

The following table shows the return values for the putProperty method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>Sets property</td>
</tr>
<tr>
<td>schema-declaration-handler</td>
<td></td>
</tr>
<tr>
<td>declaration-handler</td>
<td>Trappable error (if unable to set property).</td>
</tr>
<tr>
<td>lexical-handler</td>
<td></td>
</tr>
<tr>
<td>schemas</td>
<td></td>
</tr>
<tr>
<td>dom-node</td>
<td></td>
</tr>
<tr>
<td>xmldecl-encoding</td>
<td></td>
</tr>
<tr>
<td>xmldecl-version</td>
<td></td>
</tr>
<tr>
<td>xmldecl-standalone</td>
<td></td>
</tr>
<tr>
<td>xml-string</td>
<td></td>
</tr>
<tr>
<td>Other properties</td>
<td>Trappable error</td>
</tr>
</tbody>
</table>

**[C/C++]**
Although a core set of properties is defined, a reader is not required to recognize any specific property names. Some property values may be available only in a specific context, such as before, during, or after a parse.

Syntax

```c
HRESULT putProperty(
    [in] const wchar_t * pwchName,
    [in] VARIANT varValue);
```

Parameters

`pwchName` [in]
The property's name (zero-terminated Unicode string).

`varValue` [in]
The corresponding data type of the property.

Return Values

The following table lists the return values for the `putProperty` method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>S_OK (regardless of parsing mode)</td>
</tr>
<tr>
<td>schema-declaration-handler</td>
<td>S_OK (not parsing)</td>
</tr>
<tr>
<td>declaration-handler</td>
<td>E_FAIL (parsing)</td>
</tr>
<tr>
<td>lexical-handler</td>
<td></td>
</tr>
<tr>
<td>schemas</td>
<td></td>
</tr>
<tr>
<td>dom-node</td>
<td></td>
</tr>
<tr>
<td>xmldecl-encoding</td>
<td>E_FAIL</td>
</tr>
<tr>
<td>xmldecl-version</td>
<td></td>
</tr>
<tr>
<td>xmldecl-standalone</td>
<td></td>
</tr>
<tr>
<td>xml-string</td>
<td></td>
</tr>
<tr>
<td>Other properties</td>
<td>E_INVALIDARG</td>
</tr>
</tbody>
</table>

Remarks

The `ISAXXMLReader` interface recognizes the following properties:
"http://xml.org/sax/properties/lexical-handler"

This is an optional extension handler for lexical events, like comments.

Datatype — ISAXXMLLexicalHandler.

Access — Read/write.

"http://xml.org/sax/properties/declaration-handler"

This is an optional extension handler for declaration events.

Datatype — ISAXXMLDeclHandler

Access — Read/write.

"charset"

Externally defined encoding to use with an XML document. An example is the one used in the MIME-header. This setting takes priority over the default encoding, which is implicitly UTF-16, or over the encoding specified in the byte order mark (BOM) of the XML document header.

This property only affects documents that are parsed using the parse method. For any documents that are parsed using the parseURL method, this property is ignored. This property can be set at any time, but it only affects documents that are opened after it is set. For example, if it is set in the resolveEntity method (which is available when you implement a SAX entity resolver), it affects the external entity to be opened, but does not affect the main document, which is currently being parsed.

Datatype — BSTR* (The string value must be a recognized description of a character set that is valid for use with XML. For example, "US-ASCII", "UTF-8", etc.)

Default — Empty (uses either the BOM in XML header or the default encoding)

Access — Read/write.

"http://xml.org/sax/properties/dom-node"
Source DOM node for current SAX event. Only available when using parse method and SAX events are generated from an IXMLDOMNode object. Not available when using parseURL method for SAX parsing of a file.

Datatype — IXMLDOMNode.

Access — Read-only.

"http://xml.org/sax/properties/xml-string"

The literal string of characters in the XML source for the current SAX event.

Datatype — String (Visual Basic) / w_char* (C/C++)

Access — Read-only.

"schemas"

An option for registering a schema with the SAX reader.

Datatype — XMLSchemaCache.

Access — (parsing) Read-only; (not parsing) Read/Write.

"schema-declaration-handler"

An option for registering a schema-based declaration handler with the SAX reader.

Datatype — Event handler that implements the IMXSchemaDeclHandler interface.

Access — Read/write.

"xmldecl-encoding"

Encoding declared in the XML header of the currently parsed document. This data is unavailable until the reader parses the XML header.

Datatype — BSTR.
Default — Empty (if there is no XML declaration or a property is not found).

Access — (parsing) Read; (not parsing) None.

"xmldecl-version"

Version declared in the XML header of currently parsed document. This data is unavailable until the reader parses the XML header.

Datatype — BSTR.

Default — Empty (if there is no XML declaration).

Access — (parsing) Read; (not parsing) None.

"xmldecl-standalone"

Declared in the XML header of currently parsed document. This data is unavailable until the reader parses the XML header.

Datatype — BSTR ("yes" or "no").

Default — Empty (if there is no XML declaration or if a property is not found).

Access — (parsing) Read; (not parsing) None.

Note  The "xmldecl-encoding", "xmldecl-version", and "xmldecl-standalone" properties provide information about the presence and content of the XML header. However, XML header information was designed for low-level reader and parser use, not for applications.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button [ ] in the upper-left corner of the page.

See Also

getProperty Method

Applies to: ISAXXMLReader Interface
parse Method

Parses an XML document or a DOMDocument object.

[Visual Basic]

An application can use this method to instruct the reader to begin parsing a document from any valid input source, including a DOMDocument object, character stream, byte stream, or Uniform Resource Identifier (URI). An application cannot invoke this method while a parsing is in progress. Instead, it should create a new instance of the reader for each nested XML document. After parsing is complete, an application can reuse the same reader object, possibly with a different input source.

Usage Syntax

oSAXXMLReader.parse(varInput)

Parameters

varInput
A variant (or a reference to one) containing the input source to parse from. The following VARIANT types are supported:
- VT_BSTR, which can contain a value of BSTR type.
- Either VT_ARRAY or VT_UI1, which can contain a value in the form of a safe array of bytes.
- VT_UNKNOWN or VT_DISPATCH, which can contain a value in the form of either an IStream, ISequentialStream, or IXMLDOMDocument.

When using references (VT_BYREF), more than one level of indirection is permitted.

An application cannot invoke this method while parsing is in progress. Instead, it should create a new instance of the reader for each nested XML document. After parsing is complete, an application can reuse the same reader object, possibly with a different input source.
Remarks

During parsing, the reader provides information about the XML document through the registered event handlers. If an error occurs during parsing, the reader calls the appropriate method of the ErrorHandler. The ErrorHandler always returns the internal error code to the parse method, even if the ErrorHandler raises a user-defined error.

Example

The following example loads books.xml into a DOMDocument object, specifies the object as a parameter of the parse method, and then displays the result in a message box.

Sub Main()
  'create the reader
  Dim rdr As New SAXXMLReader50
  'create the writer
  Dim wrt As New MXXMLWriter50

  On Error GoTo errorHandler

  wrt.byteOrderMark = True
  wrt.omitXMLDeclaration = False
  wrt.indent = True

  'set the writer to the content handler
  Set rdr.contentHandler = wrt
  Set rdr.dtdHandler = wrt
  Set rdr.errorHandler = wrt
  rdr.putProperty "http://xml.org/sax/properties/lexical-handler", wrt
  rdr.putProperty "http://xml.org/sax/properties/declaration-handler",

  'parse the XML
  rdr.parse "<catalog><book id='bk101'></book></catalog>">

  'show the results in a message box
  MsgBox wrt.output

  Exit Sub

errorHandler:
  HandleErrors
End Sub
Private Function HandleErrors()
    MsgBox Err.Description
End Function

[C/C++]

The application can use this method to instruct the reader to begin parsing a
document from a variety of sources. Supported VARIANT types are: VT_BSTR,
which can contain a value of BSTR type; either VT_ARRAY or VT_UI1, which
can contain a value in the form of a safe array of bytes; or VT_UNKNOWN,
which can contain a value in the form of either an IStream, ISequentialStream or
IXMLDOMDocument.

    VT_BYREF | VT_VARIANT -> VT_BSTR -> BSTR
    VT_BYREF | VT_VARIANT -> VT_UNKNOWN -> VT_DISPATCH -> IStream |
    VT_BYREF | VT_VARIANT -> VT_UNKNOWN -> VT_DISPATCH -> IXMLDOMDocument
    VT_BYREF | VT_VARIANT -> VT_ARRAY | VT_UI1 -> SAFEARRAY

When using references (VT_BYREF), more than one level of indirection is
permitted.

An application cannot invoke this method while parsing is in progress. Instead, it
should create a new instance of the reader for each nested XML document. After
parsing is complete, an application can reuse the same reader object, possibly
with a different input source.

Syntax

HRESULT parse (  
    [in] VARIANT varInput);

Parameters

varInput [in]
    Either a variant (VT_VARIANT) or a reference to one containing the input
    source to parse from. Supported VARIANT types and their data types for
    actual values are:

    VT_BSTR (BSTR)

    VT_ARRAY or VT_UI1 (SafeArray of bytes)
VT_UNKNOWN or VT_DISPATCH (IStream, ISequentialStream, or IXMLDOMDocument).

**Return Values**

S_OK
   The value returned if parsing is successful.
E_INVALIDARG
   The value returned if varInput is not a recognized input source.
Other
   The parser error code or the code returned by the custom ErrorHandler.

**Remarks**

During parsing, the reader provides information about the XML document through the registered event handlers. If an error occurs during parsing, the reader calls the appropriate method of the ISAXErrorHandler interface. If the ErrorHandler returns anything but S_OK, parsing is aborted and the parse method returns the internal error that stopped the parser. The ErrorHandler always returns the internal error code, even if the ErrorHandler returns a user-defined HRESULT to the parse method. The reader can return a number of return codes, including, but not limited to: E_ACCESSDENIED, INET_E_OBJECT_NOT_FOUND, INET_E_DOWNLOAD_FAILURE, MSG_E_BADSTARTNAMECHAR, or MSG_E_MISSINGSEMICOLON.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[Convert DOM to SAX](#) | [DOMDocument](#) | [ISAXErrorHandler Interface](#) | [contentHandler Property (Visual Basic)](#) | [dtdHandler Property (Visual Basic)](#) | [entityResolver Property (Visual Basic)](#) | [errorHandler Property (Visual Basic)](#) | [putContentHandler Method (C/C++)](#) | [putDTDHandler Method (C/C++)](#) | [putEntityResolver Method (C/C++)](#) | [putErrorHandler Method (C/C++)](#)

Applies to: [ISAXXMLReader Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
parseURL Method

Parses an XML document from a system identifier (URI).

**Note**  If you are using Microsoft® Internet Explorer 5.5, you may get an E_PENDING error when you attempt to load an XML file that has external entities. To correct this situation, upgrade to Internet Explorer 5.5, Service Pack 1.

[Visual Basic]

The `parseURL` method is a shortcut for reading a document from a system identifier. If the system identifier is a URL, the application must fully resolve the URL before passing it to the reader.

**Usage Syntax**

```vbscript
oSAXXMLReader.parseURL(strURL)
```

**Parameters**

`strURL`

The URL to load.

**Remarks**

While parsing, the reader provides information about the XML document through the registered event handlers. If an error occurs during parsing, the reader calls the appropriate method of the `ErrorHandler`. The `ErrorHandler` always returns the internal error code to the `parseURL` method, even if the `ErrorHandler` raises a user-defined error.

[C/C++]

The base URL and secure base URL can be set by the `putBaseURL` and `putSecureBaseURL` methods. These methods are shortcuts for reading a
document from a system identifier.

**Syntax**

```c
HRESULT ParseURL(
    [in] const wchar_t * pwchUrl);
```

**Parameters**

- **pwchUrl [in]**
  - The URL (zero-terminated Unicode string) to load from.

**Return Values**

- **S_OK**
  - The value returned if parsing is successful.
- **Other**
  - Either the parser error code or the code returned by the custom `ErrorHandler`. Unresolved URLs, security, and network errors are returned in the same way.

**Remarks**

During parsing, the reader provides information about the XML document through the registered event handlers. If an error occurs during parsing, the reader calls the appropriate method of the `ISAXErrorHandler : IUnknown` interface. If the `ErrorHandler` returns anything but S_OK, parsing is aborted and the `parseURL` method returns the internal error that stopped the parser. The `ErrorHandler` always returns the internal error code, even if the `ErrorHandler` returns a user-defined HRESULT to the `parseURL` method. The reader can return a number of return codes, including, but not limited to: E_ACCESSDENIED, INET_E_OBJECT_NOT_FOUND, INET_E_DOWNLOAD_FAILURE, MSG_E_BADSTARTNAMECHAR, or MSG_E_MISSINGSEMICOLON.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**
Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
baseURL Property (Visual Basic)

Sets the base URL for the current document. The baseURL property is stored as a string.

Usage Syntax

For getting the base URL:

strbaseURL = oSAXXMLReader.baseURL

For setting the base URL:

oSAXXMLReader.baseURL = strBaseURL

Return Values

A trappable error if the application tries to set the baseURL property while parsing is in progress.

Example

Dim strURL As String
reader.baseURL = "http://My.web.site/"
strURL = reader.baseURL

Remarks

If a URL does not end with a slash or backslash, the text following the backslash is considered the destination file name. For example, "http://www.microsoft.com/msdn/" sets the base URL to "http://www.microsoft.com/msdn" while "http://www.microsoft.com/msdn" sets the base URL to "http://www.microsoft.com/".

See Also
secureBaseURL Property (Visual Basic)

Applies to: IVBSAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
contentHandler Property (Visual Basic)

Sets the current ContentHandler.

Usage Syntax

For getting the ContentHandler:

Set objContentHandler = oSAXXMLReader.ContentHandler

For setting the ContentHandler:

Set oSAXXMLReader.ContentHandler = objCcontentHandler

Remarks

The contentHandler property gets or sets the current ContentHandler for the IVBSAXContentHandler interface.

See Also

Applies to: IVBSAXXMLReader Interface
**dtdHandler Property (Visual Basic)**

Sets the current DTDHandler.

**Usage Syntax**

For getting the DTDHandler:

Set objDTDHandler = oSAXXMLReader.dtdHandler

For setting the DTDHandler:

Set oSAXXMLReader.dtdHandler = objDTDHandler

**Remarks**

The `dtdHandler` property gets or sets the current DTDHandler for the IVBSAXDTDHandler interface.

**See Also**

Applies to: [IVBSAXXMLReader Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
entityResolver Property (Visual Basic)

Sets the current EntityResolver.

Usage Syntax

For getting the EntityResolver:

```vba
set objentityResolver = oSAXXMLReader.entityResolver
```

For setting the EntityResolver:

```vba
set oSAXXMLReader.entityResolver = objentityResolver
```

Remarks

The entityResolver property gets or sets the current EntityResolver for the IVBSAXEntityResolver interface.

See Also

Applies to: IVBSAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**errorHandler Property (Visual Basic)**

Registers or returns the current ErrorHandler.

**Usage Syntax**

For getting the ErrorHandler:

```vbnet
Set objErrorHandler = oSAXXMLReader.errorHandler
```

For setting the ErrorHandler:

```vbnet
Set oSAXXMLReader.errorHandler = objErrorHandler
```

**Remarks**

The `errorHandler` property gets or sets the current `ErrorHandler` for the `IVBSAXErrorHandler` interface.

**See Also**

Applies to: [IVBSAXXMLReader Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
secureBaseURL Property (Visual Basic)

Sets the secure base URL for the document. If the secure base URL is not set, full access is allowed. The secure base URL cannot be set during parsing. The underlying runtime environment determines the response to the security violation.

By setting the secureBaseURL, the application requests security enforcement by the Microsoft® Internet Explorer security settings, which control behavior on the user's computer. This means that the user, not the application, controls Internet security settings.

To understand what the secure base URL is, consider this example. When a parser reads an XML file from the site "www.a.com", both the base URL and the secure base URL are the same: "www.a.com". Now, consider that in the XML file there is an external reference to the site "www.b.com". When reading the externally referenced file, the base URL will be "www.b.com", while the secure base URL is still "www.a.com".

**Note**  For security reasons, the secure base URL cannot be set with script within the context of Internet Explorer.

**Usage Syntax**

For getting the secureBaseURL property:

```vbnet
strVal = oSAXXMLReader.secureBaseURL
```

For setting the secureBaseURL property:

```vbnet
oSAXXMLReader.secureBaseURL = strVal
```

**Return Values**

A trappable error if the application tries to set the secureBaseURL property
during the parse operation.

See Also

baseUrl Property (Visual Basic)

Applies to: IVBSAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getBaseURL Method (C/C++)**

Returns the base URL for the document.

**Syntax**

```c
HRESULT getBaseURL(
    [out] const wchar_t ** ppwchBaseUrl);
```

**Parameters**

`ppwchBaseUrl [out]`

The base URL (zero-terminated Unicode string) for the document.

**Return Codes**

S_OK

The value returned if the URL is successfully returned.

**Remarks**

The base URL represents the URL of the current XML document. The reader uses this base URL to resolve relative links (either for those links related to the `parseURL` method or for external entities in an XML document). To resolve these links, the reader uses the base URL in much the same way that an Internet browser uses a Web page's URL to resolve relative links contained on that page.

The base URL may be a complete URL. However, the reader only uses that portion of the URL up to and including the final backslash when resolving links. For instance, the base URL for a document is "http://msdn.microsoft.com/xml/test.xml". The reader would then only use "http://msdn.microsoft.com/xml/" when resolving links.

Here is an example of how the reader uses the base URL to resolve links. The base URL for a document is "http://msdn.microsoft.com/xml/test.xml". In that document, there is a link for "mytest.xml". The reader would resolve this link as
"http://msdn.microsoft.com/xml/mytest.xml."

**See Also**

putBaseURL Method (C/C++) | getSecureBaseURL Method (C/C++) | putSecureBaseURL Method (C/C++)

Applies to: [ISAXXMLReader Interface](http://msdn.microsoft.com/xml/mytest.xml)
**putBaseURL Method (C/C++)**

Sets the base URL for the document. The base URL cannot be set during parsing.

**Syntax**

```c
HRESULT putBaseURL(
    [in] const wchar_t * pwchBaseUrl);
```

**Parameters**

*pwchBaseUrl* [in]

The base URL (zero-terminated Unicode string) for the document.

**Return Codes**

*S_OK*

The value returned if the base URL is successfully set.

*E_FAIL*

The value returned if the application tries to set the base URL during parsing.

**Remarks**

The base URL represents the URL of the current XML document. The reader uses this base URL to resolve relative links (either for those links related to the parseURL method or for external entities in an XML document). To resolve these links, the reader uses the base URL in much the same way that an Internet browser uses a Web page's URL to resolve relative links contained on that page.

The base URL may be a complete URL. However, the reader only uses that portion of the URL up to and including the final backslash when resolving links. For instance, the base URL for a document is "http://msdn.microsoft.com/xml/test.xml". The reader would then only use "http://msdn.microsoft.com/xml/" when resolving links.
Here is an example of how the reader uses the base URL to resolve links. The base URL for a document is "http://msdn.microsoft.com/xml/test.xml". In that document, there is a link for "mytest.xml". The reader would resolve this link as "http://msdn.microsoft.com/xml/mytest.xml".

See Also

getBaseURL Method (C/C++) | getSecureBaseURL Method (C/C++) | putSecureBaseURL Method (C/C++)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getcontentHandler Method (C/C++)**

Returns the current ContentHandler. The ContentHandler is the class that is implemented to process the contents of a document.

**Syntax**

```c
HRESULT getContentHandler(
    [out, retval] ISAXContentHandler ** ppHandler);
```

**Parameters**

*ppHandler* [out, retval]
- The ContentHandler currently in use or Null (if a ContentHandler has not been set).

**Return Values**

*S_OK*
- The value returned if the ContentHandler is successfully returned.

**See Also**

- [putContentHandler Method (C/C++)](#)

Applies to: [ISAXXMLReader Interface](#)
**putContentHandler Method (C/C++)**

Allows an application to register a `ContentHandler`. If the application does not register a `ContentHandler`, all content events reported by the reader will be ignored. An application can register a new or different handler while in the middle of parsing a document, and the reader will immediately start using the new handler.

**Syntax**

```c
HRESULT putContentHandler(
    [in] ISAXContentHandler * pHandler);
```

**Parameters**

`pHandler` [in]  
The `ContentHandler` to register.

**Return Value**

S_OK  
The value returned if the `ContentHandler` is successfully registered.

**See Also**

`getContentHandler Method (C/C++)`

Applies to: `ISAXXMLReader Interface`
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getDTDHandler Method (C/C++)**

Returns the current DTDHandler.

**Syntax**

```c
HRESULT getDTDHandler(
    [out, retval] ISAXDTDHandler ** ppHandler);
```

**Parameters**

*ppHandler* [out, retval]

The DTDHandler currently in use or Null (if a DTDHandler has not been set).

**Return Values**

*S_OK*

The value returned if the DTDHandler is successfully returned.

**See Also**

[putDTDHandler Method (C/C++)](#)

Applies to: [ISAXXMLReader Interface](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
putDTDHandler Method (C/C++)

Allows an application to register a DTDHandler. If the application does not register a DTDHandler, all DTD events reported by the reader are ignored. An application can register a new or different handler while in the middle of parsing a document, and the reader will immediately start using the new handler.

Syntax

HRESULT putDTDHandler(
    [in] ISAXDTDHandler * pHandler);

Parameters

pHandler [in]
   The DTDHandler to register.

Return Values

S_OK
   The value returned if the DTDHandler is successfully registered.

See Also

getDTDHandler Method (C/C++)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**getEntityResolver Method (C/C++)**

Returns the current EntityResolver.

**Syntax**

```c
HRESULT getEntityResolver(
    [out, retval] ISAXEntityResolver ** ppResolver);
```

**Parameters**

*ppResolver* [out, retval]
Pointer to the EntityResolver currently in use.

**Return Values**

*S_OK*
The value returned.

**See Also**

[putEntityResolver Method (C/C++)](#)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**putEntityResolver Method (C/C++)**

Allows an application to register an EntityResolver. If the application does not register an EntityResolver, the reader performs its own default resolution. An application can register a new or different EntityResolver while in the middle of parsing a document, and the reader will immediately start using the new EntityResolver.

The ISAXEntityResolver interface provides a way to resolve references to external entities (data held in separate files identified by a system or public identifier). The reader automatically resolves internal entities, character references, and predefined entities. However, some applications build XML documents from databases and specialized input sources that use Uniform Resource Identifier (URI) types rather than URLs. When external entities cannot be found by URL, a SAX application should provide a class that implements the EntityResolver interface.

**Syntax**

```c
HRESULT putEntityResolver(
    [in] ISAXEntityResolver * pResolver);
```

**Parameters**

- **pResolver** [in]  The EntityResolver to register.

**Return Values**

- **E_FAIL**  The value returned if the application fails to set the EntityResolver property.

**See Also**

- [getEntityResolver Method (C/C++)](#) | [ISAXEntityResolver Interface](#)
Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getErrorHandler Method (C/C++)

Returns the current ErrorHandler.

Syntax

HRESULT getErrorHandler(
    [out, retval] ISAXErrorHandler ** ppHandler);

Parameters

ppHandler [out, retval]
    The current ErrorHandler or Null (if an ErrorHandler has not been set).

Return Values

S_OK
    The value returned if the ErrorHandler is successfully returned.

See Also

putErrorHandler Method (C/C++)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
putErrorHandler Method (C/C++)

Allows an application to register an ErrorHandler. If the application does not register an ErrorHandler, all error events reported by the reader are ignored. It is strongly recommended that all Simple API for XML (SAX2) applications implement an ErrorHandler to avoid unexpected bugs. An application can register a new or different handler while in the middle of parsing a document, and the reader will immediately start using the new handler.

Syntax

```c
HRESULT putErrorHandler(
    [in] ISAXErrorHandler * pHandler);
```

Parameters

`pHandler [in]`

The ErrorHandler to use.

Return Values

S_OK

The value returned if the ErrorHandler is successfully registered.

See Also

getErrorHandler Method (C/C++)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getSecureBaseUrl Method (C/C++)

Returns the secure base URL for the document. If secure base URL is not set, full access is allowed. The secure base URL cannot be set during parsing.

Syntax

HRESULT getSecureBaseUrl (
    [out, retval] const wchar_t ** ppwchSecureBaseUrl);

Parameters

ppwchSecureBaseUrl [out, retval]
    The secure base URL (zero-terminated Unicode string).

Return Values

S_OK
    The value returned if the secure base URL is successfully returned.

See Also

putSecureBaseUrl Method (C/C++)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
putSecureBaseUrl Method (C/C++)

Sets the secure base URL for the document. If the secure base URL is not set, full access is allowed. The secure base URL cannot be set during parsing. The underlying run-time environment determines the response to the security violation.

By setting the secure base URL, the application requests security enforcement by the Microsoft® Internet Explorer security settings, which control behavior on the user's computer. This means that the user, not the application, controls Internet security settings.

To understand what the secure base URL is, consider this example. When a parser reads an XML file from the site "www.a.com", both the base URL and the secure base URL are the same: "www.a.com". Now, consider that in the XML file there is an external reference to the site "www.b.com". When reading the externally referenced file, the base URL will be "www.b.com", while the secure base URL is still "www.a.com".

**Note** For security reasons, the secure base URL cannot be set with script within the context of Internet Explorer.

**Syntax**

```c
HRESULT putSecureBaseUrl (  
    [in] const wchar_t * pwchSecureBaseUrl
)
```

**Parameters**

- **pwchSecureBaseUrl** [in]
  The secure base URL (zero-terminated Unicode string).

**Return Values**

- **S_OK**
  The value returned if the secure base URL is successfully set.
E_FAIL
The value returned if the application tries to set the secure base URL during parsing.

See Also

getSecureBaseUrl Method (C/C++)

Applies to: ISAXXMLReader Interface
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
The **MXValidator CoClass** is a SAX XML filter that validates an XML stream. The MXValidator is inserted between the XML source and the XML consumer to validate the contents of the stream. The MXValidator validates the series of events generated by the ISAXContentHandler, and generates a new stream that includes all those events, plus some additional validation-related events. Input events are not buffered. Each input event generates at least one output event.

### Controlling Validator Behavior

You can use the following methods to observe and control the behavior of the validator:

- **getFeature**
  Gets the current state of a feature.
- **putFeature**
  Configures a feature.

### Supported Features and Properties

The **MXValidator CoClass** supports the following feature:

- "exhaustive-errors"

The **MXValidator CoClass** implements the following properties:

- "schemas"
- "schema-declaration-handler"

### Methods

- **getFeature**
  Returns a Boolean value that indicates whether a feature is enabled or disabled.
- **getProperty**
  Returns the current value of a property.
putFeature Enables or disables a feature.
putProperty Sets the value of a property.

Properties

contentHandler The current content handler.
dtdHandler The current DTD Handler.
entityResolver The current entity resolver.
errorHandler The current error handler.

Versioning

MSXML 5.0 and later

Requirements

Implementation: msxml5.dll

[C/C++]

Header and LIB files: msxml2.h, msxml2.lib

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

Example

See the putProperty Method for an example of using MXValidator.

See Also

ISAXXMLReader Interface
getFeature Method

Returns a Boolean value that indicates whether a feature is enabled or disabled.

The `mxValidator` CoClass supports the following feature:

- "exhaustive-errors"

[Visual Basic]

**Implementation Syntax**

Function getFeature(strName As String) As Boolean

**Usage Syntax**

`boolFeature = oMXValidator.getFeature(strName)`

**Parameters**

`strName`  
A string representing the name of the feature.

**Return Values**

`boolFeature`  
A Boolean (True/False) value that indicates whether the feature is enabled or disabled.

[C/C++]

**Syntax**

`HRESULT getFeature(`

  [in] BSTR strName,

  [out, retval] VARIANT_BOOL *fValue);`
Parameters

strName [in]
The name of the feature.
fValue [out, retval]
Set to indicate whether the feature is enabled (TRUE) or disabled (FALSE)

Return Values

S_OK
The fValue parameter contains a Boolean value indicating whether the feature specified in strName is enabled (TRUE) or disabled (FALSE).
E_INVALIDARG
The value returned if the argument provided for this feature is not recognized.

Remarks

The "exhaustive-errors" feature is the only feature and is not configurable. It is always enabled and returns True.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

putFeature Method

Applies to: MXValidator CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
getProperty Method

Returns the current value of a property.

The MXValidator CoClass implements the following properties:

- "schemas"
- "schema-declaration-handler"

[Visual Basic]

**Implementation Syntax**

Function getProperty(strName As String) As Variant

**Usage Syntax**

varValue = oMXValidator.getProperty(strName)

**Return Values**

A variant containing the current value of the property. The following table describes the properties defined for the MXValidator object.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;schemas&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;schema-declaration-handler&quot;</td>
<td>Current value of the property.</td>
</tr>
<tr>
<td>Other properties</td>
<td>Trappable error.</td>
</tr>
</tbody>
</table>

[C/C++]

**Syntax**

HRESULT getProperty(
    [in] BSTR strName,
[out, retval] VARIANT * varValue);

**Return Values**

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;schemas&quot;</td>
<td>S_OK — The varValue parameter contains the current value of the property.</td>
</tr>
<tr>
<td>&quot;schema-declaration-handler&quot;</td>
<td>E_INVALIDARG — The property is not recognized.</td>
</tr>
</tbody>
</table>

Other properties

**Remarks**

The following properties are defined for the **MXValidator**:

"schemas"
An option for registering schemas with the SAX reader.

Datatype — **XMLSchemaCache**.

Access — (parsing) Read-only; (not parsing) Read/Write.

"schema-declaration-handler"
An option for registering a schema-based declaration handler with the SAX reader.

Datatype — Event handler that implements the **IMXSchemaDeclHandler** interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

**See Also**

[putProperty Method](#)

Applies to: **MXValidator CoClass**
putFeature Method

Configures a feature.

The `MXValidator` CoClass only supports the following feature:

- "exhaustive-errors"

[Visual Basic]

**Implementation Syntax**

Sub putFeature(strName As String, varValue as Variant)

**Usage Syntax**

`oMXValidator.putFeature(strName, varValue)`

[C/C++]

**Syntax**

```c
HRESULT putFeature(
    [in] BSTR strName,
    [in] VARIANT_BOOL fValue);
```

**Return Values**

S_OK
- The `fValue` parameter is TRUE (not parsing).
E_FAIL
- The `fValue` parameter is FALSE (parsing).
E_INVALIDARG
- The value returned if the argument provided for this feature is not recognized.

**Remarks**
If the putFeature method fails, either because the feature setting could not be changed, or because an invalid feature name was passed to the method, a trappable error is raised.

The "exhaustive-errors" feature reports all validation errors. It is always enabled and only returns True.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

getFeature Method

Applies to: MXValidator CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**putProperty Method**

Sets the value of a property.

[Visual Basic]

**Implementation Syntax**

Sub putProperty(strName As String, varValue As Variant)

**Usage Syntax**

oMXValidator.putProperty(strName, varValue)

**Parameters**

- **strName**
  A string representing the name of the property.
- **varValue**
  The value that will be assigned to the property.

**Return Values**

The following table describes the results of calling the putProperty method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemas</td>
<td>Raises a trappable error if the property value cannot be set.</td>
</tr>
<tr>
<td>schema-declaration-handler</td>
<td>Sets the value of the property.</td>
</tr>
<tr>
<td>Other properties</td>
<td>Trappable error.</td>
</tr>
</tbody>
</table>

**Example**

This example illustrates how to use the MXValidator CoClass. The sample code performs the following tasks.
1. Creates a standalone SAX validator using an instance of MXValidator.
2. Loads and assigns **books.xsd** as the XSD schema for use by the validator.
3. Synthetically generates an XML document using SAX.
4. Uses the standalone SAX validator to validate the document.
5. Returns information on validation success or failure as output.

The example uses a resource file, **books.xsd**. We've provided source files for the sample in Visual Basic.

- **Resource File (books.xsd)**
- **Visual Basic Code (Mxvld.frm)**
- **Output for the MXValidator putProperty Example**

[C/C++]

**Syntax**

```c
HRESULT putProperty(
    [in] BSTR strName,
    [in] VARIANT varValue);
```

**Parameters**

- **strName [in]**
  A string representing the name of the property.
- **varValue [in]**
  The value that will be assigned to the property.

**Return Values**

The following table describes the results of calling the `putProperty` method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;schemas&quot;</td>
<td>S_OK (Not parsing.)</td>
</tr>
<tr>
<td>&quot;schema-declaration-handler&quot;</td>
<td>E_FAIL (Parsing.)</td>
</tr>
<tr>
<td></td>
<td>S_OK (Regardless of parsing mode.)</td>
</tr>
</tbody>
</table>
Other properties

E_FAIL

Remarks

The following properties are defined for the MXValidator:

"schemas"

 Gets or sets the collection of one or more schemas to be used by the MXValidator when it validates the XML stream.

 Datatype — XMLSchemaCache.

 Access — (parsing) Read-only; (not parsing) Read/Write.

"schema-declaration-handler"

 Gets or sets the schema-based declaration handler to be used by the MXValidator.

 Datatype — Event handler that implements the IMXSchemaDeclHandler interface.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

getProperty Property Method

Applies to: MXValidator CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
Resource: books.xsd

Use the following XSD schema definition file to validate the document created using the putProperty method of the MXValidator CoClass.

XSD schema definition file (books.xsd)

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="author" type="xsd:string"/>
              <xsd:element name="title" type="xsd:string"/>
              <xsd:element name="genre" type="xsd:string"/>
              <xsd:element name="price" type="xsd:float"/>
              <xsd:element name="publish_date" type="xsd:date"/>
              <xsd:element name="description" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Visual Basic Source: Mxvld.frm

Option Explicit

Private Function HandleErrors()
    MsgBox Err.Description
End Function

Private Sub Form_Resize()
    ' Resize text box to size of form.
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750
End Sub

Private Sub Form_Load()
    ' Resize text box to size of form.
    Text1 = ""
    Text1.Top = 100
    Text1.Left = 100
    Text1.Width = Form1.Width - 350
    Text1.Height = Form1.Height - 750

    ' Create standalone SAX validator.
    Dim vld As New MXValidator50
    Dim cnth As MSXML2.IVBSAXContentHandler
    Dim attrs As New SAXAttributes50

    ' Create the writer.
    Dim wrt As New MXXMLWriter50
    Set vld.errorHandler = wrt

    ' Create a schema collection to use in validation.
    Dim Sch As New XMLSchemaCache50

    ' Load schemas into the schema collection, Sch.
    Sch.Add "", App.Path & "\books.xsd"
    vld.putProperty "schemas", Sch
    Set cnth = vld

    On Error GoTo errorHandler

    ' Configure the writer to indent elements.
    wrt.indent = True
'Start the document.
cnth.startDocument

' Add the XML declaration.
cnth.processingInstruction "xml", "version='1.0'"

'Add the <catalog> element to the page.
cnth.startElement "", "catalog", "catalog", attrs

' Add the id attribute to the collection with the "bk101" value
attrs.addAttribute "", "", "id", "CDATA", "bk101"
' Create the <book id="bk101"> tag.
cnth.startElement "", "book", "book", attrs
' Clear the attribute collection.
attrs.Clear

' Create the <author>Gambardella, Matthew</author> string.
cnth.startElement "", "author", "author", attrs
cnth.characters "Gambardella, Matthew"
cnth.endElement "", "author", "author"

' Create the <title>XML Developer's Guide</title> string.
cnth.startElement "", "title", "title", attrs
cnth.characters "XML Developer's Guide"
cnth.endElement "", "title", "title"

' Create the <genre>Computer</genre> string.
cnth.startElement "", "genre", "genre", attrs
cnth.characters "Computer"
cnth.endElement "", "genre", "genre"

' Create the <cost>44.95</cost> string.
' This element will be invalid according to books.xsd.
cnth.startElement "", "cost", "cost", attrs
cnth.characters "44.95"
cnth.endElement "", "cost", "cost"

' Create the <publish_date>2000-10-01</publish_date> string.
cnth.startElement "", "publish_date", "publish_date", attrs
cnth.characters "2000-10-01"
cnth.endElement "", "publish_date", "publish_date"

' Create the <description>An in-depth look at...</description> string.
cnth.startElement "", "description", "description", attrs
cnth.characters "An in-depth look at creating applications with"
cnth.endElement "", "description", "description"

' Add closing tags for the <book> and <catalog> elements.
cnth.endElement "", "book", "book"
Try It!

1. Copy the XML schema definition file from Resource: books.xsd, and paste it into a text file. Save the file as books.xsd.
2. Create a Standard EXE project in Visual Basic. Save the empty project as Mxvld.vbp to the same directory where you saved books.xsd. Name the form file Mxvld.frm.
3. Create a reference to MSXML 5.0. To do this, select References... from the Project menu, and then check the box for Microsoft XML, v5.0.
4. Double click on the TextBox icon from the tools menu. A TextBox control will appear on the project's form named Text1. In the properties for Text1, set the ScrollBars property to use "2 - Vertical" as the value and set the MultiLine property to True.
5. Copy the Visual Basic code listing above, and paste it into the Visual Basic code editor to replace whatever code is already there.
6. Execute the code by selecting Start from the Run menu.
7. Verify that your output is similar to that listed in the Output topic. To view the contents of the output more easily, resize the application form as needed.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
Output

When you build and run the MXValidator putProperty example, the output should appear as follows. This is because the correct and valid name for the element in use at this location in the SAX document created would be <price/>, not <cost/>:

Failure:

Error:
Description: Element 'cost' is unexpected according to content model
Expecting: price.

Error Code: -2147467259
contentHandler Property

Gets or sets the current content handler.

[Visual Basic]

Usage Syntax

To get the current content handler:

Set oContentHandler = oMXValidator.contentHandler

To set the current content handler:

Set oMXValidator.contentHandler = oContentHandler

[C/C++]

Syntax

[propget]
HRESULT contentHandler(
    [out, retval] IUnknown ** oHandler);
[propputref]
HRESULT contentHandler(
    [in] IUnknown *oHandler);

oHandler [out, retval]
    A reference to the current content handler of the MXValidator, either an IVBSAXContentHandler or an ISAXContentHandler.

oHandler [in]
    Either an IVBSAXContentHandler or an ISAXContentHandler.

Return Values

S_OK
    The value returned if no errors occur.
E_FAIL
The value returned if a reference to the content handler could not be set or retrieved.

Remarks

The `contentHandler` property can accept either an `IVBSAXContentHandler` or an `ISAXContentHandler`.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

[ISAXContentHandler/IVBSAXContentHandler Interface](#)

Applies to: [MXValidator CoClass](#)
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
dtdHandler Property

Gets or sets the current DTD handler.

[Visual Basic]

Usage Syntax

Not supported on mxValidator. If you attempt to use `getProperty` to read this property, it returns NULL and S_OK. If you attempt to set it using `putProperty`, it returns E_FAIL.

[C/C++]

Syntax

[propget] HRESULT dtdHandler(
   [out, retval] IUnknown ** oHandler);
[propputref] HRESULT dtdHandler(
   [in] IUnknown * oHandler);

Parameters

`oHandler` [out, retval]
   This parameter is ignored. Property always returns E_FAIL if set.
`oHandler` [in]
   This parameter is ignored. Property always returns E_FAIL if set.

Return Values

S_OK
   No errors occurred.
E_FAIL
   The property is not implemented.

To view reference information for Visual Basic or C/C++ only, click the
Language Filter button in the upper-left corner of the page.

See Also

Applies to: MXValidator CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
entityResolver Property

Gets or sets the current entity resolver.

[Visual Basic]

Implementation Syntax

Not supported on mxValidator. If you attempt to use **get**Property** to read this property, it returns NULL and S_OK. If you attempt to set it using **put**Property, it returns E_FAIL.

[C/C++]

Syntax

[propget] HRESULT entityResolver(
   [out, retval] IUnknown ** oResolver);
[propputref] HRESULT entityResolver(
   [in] IUnknown * oResolver)

Parameters

 oResolver [out, retval]
   This parameter is ignored. Property always returns E_FAIL if set.
 oResolver [in]
   This parameter is ignored. Property always returns E_FAIL if set.

Return Values

 S_OK
   No errors occurred.
 E_FAIL
   The property is not implemented.

To view reference information for Visual Basic or C/C++ only, click the
Language Filter button in the upper-left corner of the page.

See Also

Applies to: MXValidator CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
**errorHandler Property**

Gets or sets the current error handler.

[Visual Basic]

**Implementation Syntax**

To get the current error handler:

Set oErrorHandler = oMXValidator.errorHandler

To set the current error handler:

Set oMXValidator.errorHandler = oErrorHandler

[C/C++]

**Syntax**

[propget] HRESULT errorHandler(
    [out, retval] IUnknown ** oHandler);
[propputref] HRESULT errorHandler(
    [in] IUnknown * oHandler);

**Parameters**

*oHandler [out, retval]*

A reference to the current error handler of the MXValidator, either an IVBSAXErrorHandler or an ISAXErrorHandler.

*oHandler [in]*

Either an IVBSAXErrorHandler or an ISAXErrorHandler.

**Return Values**

S_OK

The value returned if no errors occur.
E_FAIL
The value returned if a reference to the error handler could not be set or retrieved.

To view reference information for Visual Basic or C/C++ only, click the Language Filter button in the upper-left corner of the page.

See Also

ISAXErrorHandler/IVBSAXErrorHandler Interface

Applies to: MXValidator CoClass
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
SAX2 C++ Common Notices

Strings

In this implementation, all strings returned to handler callbacks consist of two parts: a pointer to the wchar buffer and the length of the string. This includes strings passed either directly, such as the element name for the startElement method, or indirectly, such as strings provided by ISAXAttributes and ISAXLocator classes. Strings returned from these components are owned by the components, not the calling process. Therefore, their memory should not be released, freed, or deleted. Strings, represented as a pointer to the character buffer and length, may be not zero-terminated.

When a string consists of both a pointer to the buffer and a length, the length always contains the correct length of the string: either it is zero-terminated or not. For input parameters, the representation of strings follows traditional zero-terminated Unicode string format, for example, `rdr.putBase(L"http://microsoft.com/")`. It's not guaranteed that the content of the string will still keep the value at the next call to a handler.

After any successful getProperty call, the caller should free the memory. The MXXML Simple API for XML (SAX2) property strings are the exception to the general COM rule that system-allocated objects are allocated by the data source and freed by the consumer.
Declaration Conflicts

The Microsoft® XML Core Services (MSXML) 5.0 for Microsoft Office implementation of SAX2 is nonvalidating. As a result, a validation error does not occur if the same element is declared twice, with different definitions, in an internal DTD. When element/entity declaration conflicts occur, the first declaration takes precedence.
Skipped Entities

If an entity is declared in an external DTD, MSXML SAX2 reports it as a skipped entity and the value of the nondeclared (skipped) entity is defaulted to empty. However, MSXML SAX2 does not report entities in attributes. They are quietly skipped.
Features

Features handled and recognized by SAXXMLReader are:

- "exhaustive-errors"
- "http://xml.org/sax/features/external-general-entities"
- "http://xml.org/sax/features/external-parameter-entities"
- "http://xml.org/sax/features/lexical-handler/parameter-entities"
- "http://xml.org/sax/features/namespaces"
- "http://xml.org/sax/features/namespace-prefixes"
- "preserve-system-identifiers"
- "schema-validation"
- "server-http-request"
Properties

Properties handled and recognized by SAXXMLReader are:

- "http://xml.org/sax/properties/lexical-handler"
- "http://xml.org/sax/properties/declaration-handler"
- "http://xml.org/sax/properties/dom-node"
- "http://xml.org/sax/properties/xml-string"
- "schemas"
- "schema-declaration-handler"
- "charset"
- "xmldecl-encoding"
- "xmldecl-version"
- "xmldecl-standalone"
Return Codes

All handlers may return either S_OK to continue or an error code other than S_OK. For error codes other than S_OK, parsing is aborted and the return code is returned by the parse(), parseURL(), or resume method.

Most parser methods may return general error codes like the following.

- E_OUTOFMEMORY
  - Out of memory.
- E_INVALIDARG
  - Input parameter is invalid.
- E_FAIL
  - General failure.

In addition to these return codes, each application is responsible for checking for and handling a return value of NULL, and should not pass NULL as a return value where it is not appropriate. Otherwise, the application might report an access violation at runtime.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - SAX2 Refer
SAX2 Visual Basic Common Notices

Visual Basic Wrappers

Microsoft® Visual Basic® wrappers are constructed from C++/COM interfaces by:

- Replacing the "I" prefix in front of the interface/class name with "IVB".
- Replacing all string representations from w_char_t*/int to BSTR. For handlers, strings should be passed by reference.
- Replacing IerrorInfo parameters with BSTRs containing error messages.
- Replacing HRESULT parameters with long.
- Making all parameter names follow simplified Hungarian notation for Visual Basic. For example, strName is used in place of pwchName. Prefixes are "srt", "n", "l", and "o".
- Wherever C++ uses NULL, Visual Basic uses VT_Empty.
- All interfaces are derived from IDispatch.

Currently, Visual Basic and C++ wrappers are completely disconnected for the user. However, the same coclasses implement both C++ and VB interfaces.
Features

Features handled and recognized by SAXXMLReader are:

- "exhaustive-errors"
- "http://xml.org/sax/features/external-general-entities"
- "http://xml.org/sax/features/external-parameter-entities"
- "http://xml.org/sax/features/lexical-handler/parameter-entities"
- "http://xml.org/sax/features/namespaces"
- "http://xml.org/sax/features/namespace-prefixes"
- "preserve-system-identifiers"
- "schema-validation"
- "server-http-request"
Properties

Properties handled and recognized by S AXXMLReader are:

- "http://xml.org/sax/properties/lexical-handler"
- "http://xml.org/sax/properties/declaration-handler"
- "http://xml.org/sax/properties/dom-node"
- "http://xml.org/sax/properties/xml-string"
- "schemas"
- "schema-declaration-handler"
- "charset"
- "xmldecl-encoding"
- "xmldecl-version"
- "xmldecl-standalone"
Base interface

All interfaces are derived from IDispatch.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Glossi
XML Glossary

ABCDEFGHIJKLMNOPQRSTUVWXYZ
A

attribute
Cascading Style Sheets (CSS)

CDF

Channel Definition Format (CDF)

character data

complex data type
Data island

Data Source Object

data types

definition

document element

document entity

Document Object Model (DOM)

document type declaration

Document Type Definition (DTD)

DOM

DTD
EDI

Electronic Data Interchange (EDI)

element

entity

entity reference

Extensible Markup Language (XML)

Extensible Stylesheet Language (XSL)
facet
invalid document

info set

invalid document
mixed content
N

namespace

NCName

notation

notation declaration
parsed entity

processing instruction
reference node
replacement text
SAX

schema

schema structures

SGML

Simple API for XML (SAX)

simple data type

Simple Object Access Protocol (SOAP)

SOAP

Standard Generalized Markup Language (SGML)
T

template
tokenized attribute type
Uniform Resource Identifier (URI)

Uniform Resource Locator (URL)

Uniform Resource Name (URN)

unparsed entity

URI

URL

URN
valid

vocabulary


W

W3C

well-formed XML

Worldwide Web Consortium (W3C)
X

XDR

XML

XML-Data Reduced (XDR)

XML declaration

XML document

XML engine

XML information set

XML Object Model

XML parser

XPath

XML Pointer Language (XPointer)

XML Query Language (XQL)

XML Schema Definition (XSD)

XML vocabulary

XPointer

XQL

XSD

XSL

XSL Formatting Objects
XSL Patterns

XSL Transformations (XSLT)
Terms and Definitions

attribute

An XML structural construct. A name-value pair, separated by an equals sign, included inside a tagged element that modifies certain features of the element. All attribute values, including things like size and width, are in fact text strings and not numbers. For XML, all values must be enclosed in quotation marks.

You can declare attributes for an XML element type using an attribute list declaration.

Cascading Style Sheets (CSS)

Formatting descriptions that provide augmented control over presentation and layout of HTML and XML elements. CSS can be used for describing the formatting behavior of simply structured XML documents, but does not provide a display structure that deviates from the structure of the source data. See also Extensible Stylesheet Language.

CDF

See Channel Definition Format (CDF).

Channel Definition Format (CDF)

An XML-based data format used in Microsoft® Internet Explorer 4.0 and
later to describe Active Channel™ content and desktop components.

CDF permits a Web publisher to offer frequently updated collections of information, or channels, enabling automatic delivery to compatible Web clients. The user only needs to choose the channel once, and scheduled deliveries of the channel information will be delivered to the client without further intervention.

character data

All the textual content of an element or attribute that is not markup. XML differentiates this plain text from binary data. In the XML OM, character data is stored in text nodes, which are implemented as DOM text objects.

complex data type

An element that can contain other elements or attributes. Also known as complex type. Appears as <complexType> in XML documents.

CSS

See Cascading Style Sheets (CSS).

data island

An XML document (<XML> or <SCRIPT language="XML">). that exists within an HTML page. It allows you to script against the XML document without having to load it through script or through the <OBJECT> tag. Almost anything that can be in a well-formed XML document can be inside a data island.

HTML is used as the primary document or display format, and XML is used
to embed data within the document.

**Data Source Object**

Provides a way to bind HTML controls directly to an XML data island. It assists developers in connecting to structured XML data and supplying it to an HTML page by using the data-binding facility of dynamic HTML.

XML Data Source Object allows you to work with data one node at a time, but you can also work with multiple nodes at a time, without having to walk the document tree. It binds the data to specific controls on the page and the controls are automatically populated with data from the Data Source Object.

**data types**

The parts and subparts of an XML schema that are used as the basis of all the larger components in schema.

**definition**

A description used to create simple and complex data types.

**document element**

The element in an XML document that contains all other elements. It is the top-level element of an XML document and must be the first element in the document. There is exactly one document element, no part of which appears in the content of any other element. The document element represents the document as a whole; every other element represents a component of the document.
The terms root element and document element are interchangeable.

**document entity**

The starting point for an XML parser. Unlike other entities, the document entity has no name and cannot be referenced. It is the entity in which the XML declaration and document type declaration can occur.

**Document Object Model (DOM)**

A platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

**document type declaration**

An XML structural construct. Consists of markup code that indicates the grammar rules, or Document Type Definition (DTD), for the particular class of document. The document type declaration can also point to an external file that contains all or part of the DTD. It must appear following the XML declaration and preceding the document element. The syntax of the document type declaration is <!DOCTYPE content>.

**Document Type Definition (DTD)**

Can accompany a document, essentially defining the rules of the document,
such as which elements are present and the structural relationship between the elements. It defines what tags can go in your document, what tags can contain other tags, the number and sequence of the tags, the attributes your tags can have, and optionally, the values those attributes can have.

DTDs help to validate the data when the receiving application does not have a built-in description of the incoming data. The DTD is declared within the document type declaration production of the XML file. With XML, however, DTDs are optional.

See also schema.
**element**

An XML structural construct. An XML element consists of a start tag, an end tag, and the information between the tags, which is often referred to as the contents. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications. Each attribute specification has a name and a value. An instance of an element is declared using `<element>` tags.

Elements used in an XML file are described by a **DTD** or **schema**, either of which can provide a description of the structure of the data.

**entity**

An XML structural construct. A file, database record, or another item that contains data. The primary purpose of an entity is to hold content — not structure, rules, or grammar. Each entity is identified by a unique name and contains its own content, from a single character inside the document to a large file that exists outside the document. The function of an XML entity is similar to that of a macro definition.

The entity can be referred to by an **entity reference** to insert the entity's contents into the tree at that point. Entity declarations occur in the **DTD**.

**entity reference**

An XML structural construct. Acts as a placeholder for the content author, and the **XML parser** places the actual content at each reference site. To include an entity reference, you first insert an ampersand (&) and then enter the entity name followed by a semicolon (;), as follows: `&YourEntityName;`. Then, when the line is processed, the entity will be replaced with the entity's content.

It is used in much the same way as a macro.
Extensible Markup Language (XML)

A subset of SGML that is optimized for delivery over the Web, XML provides a uniform method for describing and exchanging structured data that is independent of applications or vendors.

The key is that with XML, the information is in the document, while the rendering instructions are elsewhere. In other words, content and presentation are separate. XML is the Web's language for data interchange and HTML is the Web's language for rendering.

At the time of this writing, XML 1.0 is a Worldwide Web Consortium Recommendation, which means that it is in the final stage of the approval process.

Extensible Stylesheet Language (XSL)

A language used to transform XML-based data into HTML or other presentation formats, for display in a Web browser. The transformation of XML into formats, such as HTML, is done in a declarative way, making it often easier and more accessible than through scripting. In addition, XSL uses XML as its syntax, freeing XML authors from having to learn another markup language.

In contrast to CSS, which "decorates" the XML tree with formatting properties, XSL transforms the XML tree into a new tree (the HTML), allowing extensive reordering, generated text, and calculations — all without modification to the XML source. The source can be maintained from the perspective of "pure content" and can simultaneously be delivered to different channels or target audiences by just switching style sheets.

XSL consists of two parts, a vocabulary for transformation and the XSL Formatting Objects.
A restriction on a data type. A single defining aspect of value space. There are two types of facets: fundamental and constraining.

**Return to top**

**infoset**

See [XML information set](https://www.w3.org/Inf.html).  

**Return to top**

**invalid document**

Documents that do not follow the XML tag rules. If a document has a **DTD** or **schema**, and it doesn't follow the rules defined in its DTD or schema, that document is invalid as well.

**Return to top**

**mixed content**

Element types with mixed content are allowed to hold either character data alone or character data interspersed with child elements. In this case, the types of the child elements can be constrained, but not their order or their number of occurrences.

**Return to top**

**namespace**

A mechanism that allows developers to uniquely qualify the element names and relationships and to make these names recognizable. By doing so, they can avoid name collisions on elements that have the same name but are defined in different vocabularies. They allow tags from multiple namespaces to be mixed, which is essential if data is coming from multiple sources. Namespaces ensure that element names do not conflict, and clarify who defined which term.

A namespace identifies an [XML vocabulary](https://www.w3.org/TR/xml-namespaces/) defined within a [URN](https://www.w3.org/inf/urn.html). An attribute on an element, attribute, or entity reference associates a short name
with the URN that defines the namespace; that short name is then used as a
prefix to the element, attribute, or entity reference name to uniquely
identify the namespace. Namespace references have scope. All child nodes
beneath the node that specifies the namespace inherit that namespace. This
allows nonqualified names to use the default namespace. See also RDF
namespace.

NCName

An XML name that does not contain a colon (:). An NCName begins with
either a letter or an underscore (_) character, followed by any combination
of letters, digits, accents, diacritical marks, periods (.), hyphens (-), and
underscores (_) permitted in the XML specification. The following list
shows some example NCNames:

x

_aaabb.ccc

catalog

part-number

_-.--...

ÅÆÐÓÖ

Return to top

notation

tells the parser what type of object is being referenced. Usually refers to a
data format of non-XML data, such as BMP. A notation identifies by name
the format of unparsed entities, the format of elements that bear a notation
attribute, or the application to which a processing instruction is addressed.

Return to top

notation declaration
Tells the parser how to deal with a specific binary file type, as well as provides a name and an external identifier for a notation.

The notation declaration gives an internal name to an existing notation so that it can be referred to in attribute list declarations, unparsed entity declarations, and processing instructions.

The external identifier is used for the notation, which can allow an XML parser or its client application to locate a helper application capable of processing data in the given notation.

parsed entity

An entity that has content that is parsed and replaced with actual literal values. The result is called the replacement text. Parsed entities can only contain character data or XML markup.

processing instruction

An XML structural construct. A mechanism for embedding information in a file intended for proprietary applications rather than the XML parser or browser. The XML parser passes the instructions to the application.

A processing instruction is a string of text included almost anywhere in an XML document's character data between <? and ?> marks. It begins with the name of the application for which the PI is intended, followed by the data for the instruction.

An example is the XML declaration that begins every valid XML file:

`<?xml version="1.0" standalone="yes" ?>`

reference node

The reference node for a search context is the node that is the immediate
parent of all nodes in the search context. Every search context has an associated reference node.

replacement text

The content of parsed entities, after replacement of character references and parameter-entity references.

SAX

See Simple API for XML.

schema

A formal specification of element names that indicates which elements are allowed in an XML document, and in what combinations. It also defines the structure of the document: which elements are child elements of others, the sequence in which the child elements can appear, and the number of child elements. It defines whether an element is empty or can include text. The schema can also define default values for attributes.

A schema is functionally equivalent to a DTD, but is written in XML. A schema also provides for extended functionality such as data typing, inheritance, and presentation rules. Consequently, the new schema languages are far more powerful than DTDs.

schema structures

The compounds that can be constructed from data types and are used to describe the element, attribute, and validation structure of a document type.
SGML

See **Standard Generalized Markup Language**.

Return to top

Simple API for XML (SAX)

An XML API that allows developers to take advantage of event-driven XML parsing. Unlike the DOM specification, SAX doesn't require the entire XML file to be loaded into memory. SAX notifies you when certain events happen as it parses your document. When you respond to an event, any data you don't specifically store is discarded. If your document is very large, using SAX will save significant amounts of memory when compared to using DOM. This is especially true if you only need a few elements in a large document.

Return to top

**simple data type**

An element that contain only text. Also known as simple type. Appears as `<simpleType>` in XML documents. Attributes are considered simple types because they contain only text.

Return to top

Simple Object Access Protocol (SOAP)

An open, extensible way for applications to communicate using XML-based messages over the Web, regardless of what operating system, object model, or language they use. SOAP provides a way to use the existing Internet infrastructure to enable applications to communicate directly with each other without being unintentionally blocked by firewalls.

Return to top

**SOAP**

See **Simple Object Access Protocol**.
**Standard Generalized Markup Language (SGML)**

The international standard for defining descriptions of structure and content of electronic documents. Despite its name, SGML is not a language in itself, but a way of defining languages that are developed along its general principles. SGML defines the way that a markup language is built by specifying the syntax and definitions for the elements and attributes that compose it.

XML is a subset of SGML designed to deliver SGML-type information over the Web, while HTML is an application of SGML.

**template**

The basis of the XML style sheet is the template rule, which makes a template that allows a user agent to construct a styled Result node from a Source node. The template has two parts:

The matching part identifies the source (XML) node to which the processing action is to be applied. The matching information is contained in the **match** attribute.

The processing part defines how the children are to be processed and what styling is to be applied to them. The processing information is contained in the template's child elements.

**tokenized attribute type**

In a tokenized type, the parser will normalize all white space to a single space character and will eliminate leading and trailing white space altogether. It will also validate the contents based on the declared type.

Seven attribute types are characterized as tokenized types because each value represents either a single token (**ID**, **IDREF**, **ENTITY**, **NM_TOKEN**)
or a list of tokens (IDREFS, ENTITIES, and NM_TOKENS).

**Uniform Resource Identifier (URI)**

A superclass that includes both **URNs** and **URLs**. Presently, URI means URL in nearly all cases when discussing XML, although it is expected that URNs will become more numerous in the future. The URI supplies a universally unique number or name that can identify an element or attribute in a universally unique way.

URIs are a slightly more general scheme for locating resources on the Internet that focuses more on the resource and less on the location. In theory, a URI could find the closest copy of a mirrored document or locate a document moved from one site to another.

**Uniform Resource Locator (URL)**

The set of **URI** schemes that have explicit instructions on how to access the resource on the Internet.

URLs are uniform in that they have the same basic syntax no matter what specific type of resource (Web page, newsgroup) is being addressed or what mechanism is described to fetch it.

**Uniform Resource Name (URN)**

Identifies a persistent Internet resource. A URN can provide a mechanism for locating and retrieving a schema file that defines a particular namespace. While an ordinary **URL** could provide similar functionality, a URN is more robust and easier to manage for this purpose because a URN can refer to more than one URL.

Unlike URLs, URNs are not location-dependent.
unparsed entity

Any block of non-XML data, sometimes referred to as a binary entity because its content is often a binary file (such as an image) that is not directly interpreted by the XML parser. An unparsed entity could contain plain text, so the term binary is a bit misleading.

Unlike a parsed entity, an unparsed entity requires a notation, which identifies the format or type of resource to which the entity is declared. Beyond a requirement that an XML parser make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.

URI

See Uniform Resource Identifier.

URL

See Uniform Resource Locator.

URN

See Uniform Resource Name.

valid XML

XML that conforms to the rules defined in the XML specification, as well as the rules defined in the DTD or schema.

The parser must understand the validity constraints of the XML
specification and check the document for possible violations. If the parser finds any errors, it must report them to the XML application. The parser must also read the **DTD**, validate the document against it, and again report any violations to the XML application.

Because all of this parsing and checking can take time and because validation might not always be necessary, XML supports the notion of the **well-formed** document.

**vocabulary**

See XML **vocabulary**.

**W3C**

See **Worldwide Web Consortium**.

**well-formed XML**

XML that follows the XML tag rules listed in the W3C Recommendation for XML 1.0, but doesn't have a DTD or schema. A well-formed XML document contains one or more elements; it has a single document element, with any other elements properly nested under it; and each of the parsed entities referenced directly or indirectly within the document is well formed.

Well-formed XML documents are easy to create because they don't require the additional work of creating a DTD. Well-formed XML can save download time because the client does not need to download the DTD, and it can save processing time because the **XML parser** doesn't need to process the DTD.
Worldwide Web Consortium (W3C)

A standards body located at MIT that sets standards for XML, HTML, XSL, and many other Web technologies.

Return to top

XDR

See XML-Data Reduced.

XML

See Extensible Markup Language.

Return to top

XML-Data Reduced (XDR)

An early language used to create a schema, which identifies the structure and constraints of a particular XML document. XML-Data Reduced refers to the subset of the XML-Data schema specification that was made available in MSXML 3.0 and later. It carries out the same basic tasks as DTD, but with more power and flexibility. Unlike DTD, which requires its own language and syntax, XML-Data Reduced uses XML syntax for its language. Unlike XSD, which has only recently been recommended as a standard, XML-Data Reduced was implemented and made available by Microsoft well ahead of the existence of XSD as a recommended standard by the W3C XML Schema Working Group.

Return to top

XML declaration

The first line of an XML file can optionally contain the "xml" processing instruction, which is known as the XML declaration. The XML declaration can contain pseudo-attributes to indicate the XML language version, the character set, and whether the document can be used as a standalone entity.
An example is the XML declaration that begins every valid XML file:

```xml
<?xml version="1.0" standalone="yes" ?>
```

**XML document**

A document object that is well formed, according to the XML recommendation, and that might (or might not) be valid. The XML document has a logical structure (composed of declarations, elements, comments, character references, and processing instructions) and a physical structure (composed of entities, starting with the root, or document entity).

**XML engine**

Software that supports XML functionality on the client; Internet Explorer 4.0 and later include XML engines. Its components include the XML parser, the XSL processor, and schema support.

**XML information set**

A description of the information available in a well-formed XML document.

**XML Object Model**

An API that defines a standard way in which developers can interact with the elements of the XML structured tree. The XML object model exposes properties, methods, and the actual content (data) contained in an object. It controls how users communicate with trees, and exposes all tree elements as objects, which can be accessed without any return trips to the server. The XML OM uses the W3C standard **Document Object Model**.
**XML parser**

A software module used to read XML documents and provide access to their content and structure. The XML parser generates a hierarchically structured tree, then hands off data to viewers and other applications for processing, and finally returns the results to the browser. A validating XML parser also checks the XML syntax and reports errors.

**XPath**

The result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer. The primary purpose of XPath is to address parts of an XML document. It also provides basic facilities for manipulation of strings, numbers, and Booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath gets its name from its use of a path notation as used in URLs for navigating through the hierarchical structure of an XML document.

**XML Pointer Language (XPointer)**

A W3C initiative that specifies constructs for addressing the internal structures of XML documents. In particular, it provides for specific reference to elements, character strings, and other parts of XML documents, whether or not they bear an explicit ID attribute.

An XPointer consists of a series of location terms, each of which specifies a location, usually relative to the location specified by the prior location term. Each location term has a keyword (such as id, child, ancestor, and so on) and can have arguments, such as an instance number, element type, or attribute. For example, the XPointer:

```
child(2,precocious)
```

refers to the second child element whose type is *precocious*. 
XML Query Language (XQL)

A set of extensions to XSL Patterns proposed to the W3C.

XQL is an extension to the capabilities of XSL that will provide for searching into, and data retrieval from, XML documents. It provides ways to manipulate XML in order to create new documents, to control the content of existing documents, and to manage the ordering and presentation of these documents along with XSL.

XML Schema Definition (XSD)

A language proposed by the W3C XML Schema Working Group for use in defining schemas. Schemas are useful for enforcing structure and/or constraining the types of data that can be used validly within other XML documents. XML Schema Definition refers to the fully specified and currently recommended standard for use in authoring XML schemas. Because the XSD specification was only recently finalized, support for it was only made available with the release of MSXML 4.0. It carries out the same basic tasks as DTD, but with more power and flexibility. Unlike DTD, which requires its own language and syntax, XML Schema Definition uses XML syntax for its language. XSD closely resembles and extends the capabilities of XDR. Unlike XDR, which was implemented and made available by Microsoft in MSXML 2.0 and later releases, the W3C now recommends the use of XSD as a standard for defining XML schemas.

See also schema.

XML vocabulary

A set of actual elements and the structure for a specific document type used in particular data formats. Vocabularies, along with the structural relationships between the elements, are defined in a DTD that serves as the rulebook for that vocabulary.
One of the first and probably most well-know vocabularies is the **Channel Definition Format** used to define Web pages that are designed to be sent automatically, or "pushed" to client users.

**XPointer**

See **XML Pointer Language**.

**XQL**

See **XML Query Language**.

**XSD**

See **XML Schema Definition**.

**XSL**

See **Extensible Stylesheet Language**.

**XSL formatting objects**

A set of formatting semantics expressed as an **XML vocabulary**.

Conceptually, these objects form a tree. The formatting objects denote typographic elements such as page, paragraph, rule, and so forth. Finer control over the presentation of these elements is provided by a set of formatting properties, such as indents; word- and letter-spacing; and widow, orphan, and hyphenation control. The formatting objects and formatting properties provide the vocabulary for expressing presentation intent.
XSL Patterns

A declarative, non-procedural selection language implemented in MSXML versions 3.0 and earlier. For MSXML 4.0 and later, XSL Patterns is not supported. For more information about XSL Patterns, download the MSXML 2.5 SDK from MSDN®.

XSL Transformations (XSLT)

Makes use of the expression language defined by XPath for selecting elements for conditional processing and for generating text.

XSLT provides two "hooks" for extending the language, one hook for extending the set of instruction elements used in templates and one hook for extending the set of functions used in XPath expressions. These hooks are both based on XML namespaces.
Microsoft XML Core Services (MSXML) 5.0 for Microsoft Office - XML Developers' Guide

Use Data Types Within XML Documents
What is a data type within an XML document?

A data type within an XML document is a type that has been assigned to an element on the instance using the `dt:dt` attribute, or through an XML Schema, a formal definition of an XML document. In addition, data types can be declared as elements. The XML parser uses the data type information to validate the document.

Previously, XML element values were limited to a single type, string, so developers who wanted to process XML documents had to spend time converting all values to a type within their own applications. Typing your XML data will let the parser take care of data type conversions. In addition, because values are assigned a particular data type, changes made to that value must conform to that data type. This provides you with a method for validating user input.

In addition to having a string value, each XML element can also have a typed value. For example, the following XML element can have both a value of "1970-09-30" and a typed value of "Wed Sep 30 00:00:00 PDT 1970."

```xml
<date>1970-09-30</date>
```
How do I type XML element values?

Consider the following XML document

```xml
<?xml version="1.0"?>
<weather xmlns="x-schema:weatherSchema.xml">
  <date>1970-09-30</date>
  <degrees>67.5</degrees>
</weather>
```

where `weatherSchema.xml` is the following file:

```xml
<Schema xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="date" content="textOnly" dt:type="date"/>
  <ElementType name="degrees" content="textOnly" dt:type="float"/>
  <ElementType name="weather" content="eltOnly">
    <element type="date"/>
    <element type="degrees"/>
  </ElementType>
</Schema>
```

The preceding schema is a formal definition assigning data types to the elements of the preceding XML document. The attribute `xmlns` is the XML keyword for a namespace declaration. The declaration `xmlns:dt="urn:schemas-microsoft-com:datatypes"` assigns `dt` to the namespace `urn:schemas-microsoft-com:datatypes`. Any type qualified with `dt` will be from the namespace `urn:schemas-microsoft-com:datatypes`.

The schema and data type namespaces are declared at the beginning of the XML Schema so that the `dt` prefix can then be used to denote which `type` attributes hold data type designations.

```xml
<Schema xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="NUMBER" content="textOnly" dt:type="number"/>
</Schema>
```
Typing an element within the XML document

An element can also be typed through the dt attribute on the instance of the element. To do this, declare the data type namespace at the beginning of the XML document.

```xml
<NUMBERS xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <NUMBER dt:dt="number">44533</NUMBER>
</NUMBERS>
```

The dt prefix can now be used on the dt attribute to assign a data type to an instance of an element.
Data types as elements within the XML document

In addition to declaring data types on the instance of an element and in a schema, they can also be used as elements. Compare the following examples. The first declares data types on the instance of an element.

```xml
<shoes xmlns:dt="urn:schemas-microsoft-com:datatypess" id="f1121" siz
<sizes id="mens">
  <size dt:dt="int">8</size>
  <size dt:dt="int">10</size>
  <size dt:dt="int">12</size>
</sizes>
```

The second example declares the types as elements.

```xml
<shoes xmlns:dt="urn:schemas-microsoft-com:datatypes" id="f1121" siz
<array id="array1">
  <dt:int>8</dt:int>
  <dt:int>10</dt:int>
  <dt:int>12</dt:int>
</array>
```

It is not possible to type such elements through schema.

For a list of data types that you can use in your XML documents, see XDR Schema Data Types Reference.
Try it!

In the following text box, write an <ElementType> declaration for a <PRICE> element that has a data type of "fixed.14.4". Note that XML is case sensitive.

Check Declaration