About NDoc3

NDoc3 generates class libraries documentation from .NET assemblies and the XML documentation files generated by the .NET compiler.

NDoc3 uses add-on documenters to generate documentation in several different formats, including the MSDN-style HTML Help format (.chm) and MSDN-online style web pages.

The NDoc3 source code is available under the GNU General Public License. If you unfamiliar with this license or have questions about it, here is a FAQ.

We welcome everyone to use our software in the hopes that they can provide feedback, submit bug reports and fixes, or even join us as a developer!

Before You Begin

Make sure you have read Known Issues.

Make sure you have the necessary Help compilers.

What's New for This Release?

What's new in NDoc3 1.0 ?

NDoc3 Beta 4 incorporates new features related to better .NET 2.0, 3.0 and 3.5 support.

Highlights

- Support for generics
- Support for generic constraints
- Support for asymmetric property accessors

NDoc3 are build upon the NDoc project which has been dead for some time. Below are listed the most recent changes for the NDoc project.

Please be adviced that some of the features in NDoc have been temporarily disabled in NDoc3, fx. the "VS.NET 2003" Documenter.

What's new in NDoc 2.0 ?

NDoc 2.0 incorporates a very large number of new and enhanced features as well as bug fixes.

Highlights

NDoc 2.0 includes many new features:

- A complete re-implementation of the Html Help 2 documenter, now known as the "VS.NET 2003" documenter.
- Support for new code commenting tags like preliminary, threadsafety and exclude.
- Support for the **ObsoleteAttribute** and **FlagsAttribute** attributes.
- An extensibility feature that allows you to define your own code comment tags and control their formatting.
- User interface enhancements.
- Major performance enhancements in both reflection and document production.
- Better consistency between NDoc generated topics and MSDN topics.

"VS.NET 2003" Documenter

The new "VS.NET 2003" Documenter creates HTML

Help 2 documentation, with fully populated XML data islands, resulting in much better integration with the Visual Studio help system.

The new documenter produces documentation that more closely matches the latest MS documentation and includes features such as language filtering of syntax and outher text sections.

More information on this documenter can be found here

Performance

The performance of all documenters has been significantly improved.

- The XML Merge process has been speeded-up considerably, and is now only a small percentage of the total processing time regardless of the size of an NDoc project.
- Page production time has decreased by 20-50%.
- Memory usage has been significantly reduced.
- Namespace Hierarchies are produced without performance or stability problems, and are now always documented.

Assembly Loading

A number of improvements have been made to Assembly loading.

- The GUI no longer has to be restarted to reflect changes in an assembly.
- Most assemblies may now be loaded from network shares. However, due to .NET framework limitations, managed C++ assemblies must reside on a local drive or else security exceptions will be raised.
- Assembly resolution has been improved, it is now far less likely that an assembly will fail to be found.

Internationalization

NDoc can now process assemblies and comments with non-english characters.

Full Unicode (RTF-8) support is available in all documenters except MSDN HtmlHelp. There are some known problems with support for mixed character-sets in the HTML Help compiler, however these are out of our control...

Note that while we support other character-sets, NDoc generated text such as section headings cannot be localized in this release.

Concurrent NDoc runs

Multiple instances of NDoc can now be run concurrently. Previous problems with file locking have been resolved.

User Interface

- Drag and Drop. Assemblies may be added to a project by drag and drop from explorer to the assemblies list in the NDoc GUI.
- Error Handling. Error Handling has been considerably improved.
- Help Compiler Messages. Help compiler messages are now written to the log, and error messages are shown if an error occurs during compilation.
- Property Grid. The property grid has been enhanced in several respects.
- Assembly Loading Errors.
- Assemblies without XML docs can be documented
- Relative Assembly Paths

XML Documentation Tags

New Tags

Tag	Comments
<exclude/>	Directs NDoc to exclude the tagged type or member from the documentation. The tag overrides all visibility options.
	Marks the documentation of a type or member as preliminary.

<preliminary>	This tag can include text and block tags like <para> in order to put a custom warning into your help topics about preliminary items. If the tag is empty, preliminary topics will include the default message: [This is preliminary documentation and subject to change.] It is also possible to mark an entire help project as preliminary using the Preliminary project setting.
<devdoc>	

Enhanced Tags

Tag	Comments
<code>	• "lang" attribute • No more <include> to prevent indent • "Escaped" attribute
<see>	langword

Settings

New Settings

The following settings have been added to all documenters.

Setting	Comments
Main Settings	
	If true, intermediate files will be deleted after a successful build.

CleanIntermediates	For documenters that result in a compiled output, like the MSDN and VS.NET documenters, intermediate files include all of the HTML Help project files, as well as the generated HTML files.
FeedbackEmailAddress	If an email address is supplied, a mailto link will be placed at the bottom of each page using this address.
Preliminary	If true, NDoc will mark every topic as being preliminary documentation. The default notice is [This is preliminary documentation and subject to change.]
SdkDocVersion	Specifies to which version of the .NET Framework SDK documentation the links to system types will be pointing.
SdkDocLanguage	Specifies to which Language of the .NET Framework SDK documentation the links to system types will be pointing.
Show Attributes	
DocumentInheritedAttributes	If true, NDoc will document, in the syntax portion of topics, the attributes inherited from base types/members.
Visibility	
DocumentedInheritedMembers	Determines what types of inherited members are documented.
DocumentInheritedFrameworkMembers	If true, members inherited from .Net framework classes will be documented.

DocumentExplicitInterfaceImplementations	If true, members which explicitly implement interfaces will be included in the documentation. Normally, these members will not documented.
DocumentSealedProtected	If false, protected members of sealed classes will not be documented. Normally, these members will not documented.
SkipNamespacesWithoutSummaries	If true, NDoc will not document namespaces (nor any types within them) if an associated namespace summary does not exist.

Retired Settings

The following settings have been 'retired'.

Setting	Comments
Main Settings	
GetExternalSumeries	The performance of the Xml merge process has been considerably improved, and so summary information for inherited members is now always included (when available).
IncludeNamespaceHierarchy	Performance and stability problems with Namespace hierarchies have been resolved, and so namespace hierarchy pages are now always generated.

MSDN Documenter

New Settings

The following settings have been added.

Setting	Comments
Main Settings	
BinaryToc	If true, the documenter will produce a binary TOC. This can considerably improve the performance on opening a chm with a large table-of-contents. Normally this is set to true, but be aware that there are a few limitations imposed if using binary TOCs; see the Html Help Workshop help for more details...
Title	The documentation title shown in the banner at the top of every topic.
Extensibility	
ExtensibilityStylesheet	see here for further details.
HTML Help Options	
AdditionalContentResourceDirectory	Directory that contains resources (images etc.) used by the additional content pages. This directory will be recursively compiled into the help file.
LangID	The language ID of the locale used by the compiled helpfile

Retired Settings

The following settings have been 'retired'.

Setting	Comments

Main Settings	
SortTOCByNamespace	The Table-of-contents always shows Namespaces in sorted order.
SplitTOCs	This setting was used to overcome limitations in the old HtmlHelp2 documenter. With the introduction of the VS.NET documenter, this setting is no longer required.
DefaultTOC	The first namespace is now always the default selected item in the Table-of-contents.
LinkToSdkDocVersion	This setting has been superseded by SdkDocVersion and SdkDocLanguage, which are common to all documenters. NDoc will still read this setting, but the GUI will convert to the new settings on save of a project.

Improved hypertext linking logic

When a <see> reference to an item appears more than one time within a section on a page, only the first occurrence is linked, the following are just highlighted in bold. This reduces the visual 'clutter' on a page.

Table-of-Contents Page Icons

The icons used in the Table-of-Contents for documentation pages are now 'blank' rather than containing a question mark (?).

Operators and Type Conversions

Operators and Type Conversions are now correctly handled in the Table-of-Contents, and the page formatting more closely matches MSDN.

Attribute Handling

ObsoleteAttribute

The MSDN and VS.NET documenters will automatically add text to indicate that a type has the ObsoleteAttribute.

- On Namespace and Type Member list tables the summary will be start **Obsolete.**
- On Type Overview and Member Topics, the following text will be added before the summary description,
  NOTE: This *member_type* is now obsolete.
  If the ObsoleteAttribute description property is set, the description text will be displayed on the following line.

FlagsAttribute

If a type has FlagsAttribute applied, the MSDN and VS.NET documenters will automatically add the following text to the end of the summary on the type overview topic.

"This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values."

Known Issues and Limitations in NDoc3 1.0

Issue	Description
Very long type names	NDoc3 creates an HTML file your hard drive for each topic generates. Currently the nam this file is derived directly fro the full name of the type or member being documented. the total number of characte an item's full name (namesp + type + member name) exc the value of **_MAX_FNAME** characters), NDoc3 will fail because it attempts to create file with a name that is longe than the file system supports addition, the fully qualified p a file cannot exceed **_MAX_PATH** (260 character For fully qualified names und about 200 characters, the w around for this issue is to set **OutputDirectory** setting to a location as close to the root volume as possible. This will reduce the likelyhood of the path to the html files exceedi 260 characters. There is no work around for names that exceed these lengths. We will address this issue in future version of NDoc3.

Case-sensitivity	When the MSDN and JavaD documenters create files, problems can arise when the are Types or Members that d only by case. It is possible to work around problem by avoiding types ar members that differ only by c for example use public prope **Thing** and private field **_thin** instead of **Thing** and **thing**. We will address this issue in future version of NDoc3.
StrongNameIdentityPermissionAttribute	Assemblies decorated with th **StrongNameIdentityPermis** attribute can only be called b other assemblies that are ma with the specified key. NDoc3 fail when it attempts to docur an assembly compiled with th attribute. To work around this issue, conditionally compile a versic the assembly that does not include the StrongNameIdentityPermissi attribute.
Compact Framework incompatibilities	When an assembly compiled run on the .NET Compact Framework is added to an NI project, the GUI may throw a exception, especially if the assembly references Microsoft.WindowsCE.Forms SqlServerCe.

	There is no work around for t issue. We will address this issue in future version of NDoc3.
Localization	NDoc3 does not currently su Localization of headings and predefined text. We *may* address this issue future version of NDoc3.
Online SDK Links to generic classes	NDoc3 does not handle onlin SDK links to generic classes correctly, and therefor the generated links does not wor This issue will be addressed future version of NDoc3.

There are some additional tools you'll need to get in order to produce certain kinds of help titles. These tools are available online.

HTML Help Compilers

The necessary HTML Help compilers are freely available from Microsoft.

HTML Help Version 1

HTML Help 1 titles are the CHM files that most applications use for their help file format. This is the help technology used by MSDN prior to the release of Visual Studio.NET.

If you plan to create compiled Html Help 1 titles (CHM files), you will need to make sure you have Microsoft's HTML Help Workshop. This download includes the compiler capable of creating CHM help titles.

HTML Help Version 2 (Not tested in NDoc3)

HTML Help 2 is the help technology the Visual Studio.NET 2003 and higher, and the current versions of MSDN uses.

If you plan to create help titles that integrate with the VS.NET help system, you will need to download the Visual Studio Help Integration Kit (VSHIK). This download includes the compiler capable of creating HxS help titles.

Latex Compiler (Not included in this release)

If you intend to use the LaTeX documenter, you will need a compiler capable of generating dvi or pdf files. You can download a free one from http://www.miktex.org.

Other Tools

H2Reg

Deploying compiled HTML Help 2 titles and integrating them into your customers Visual Studio.NET help system can be tricky. The VSHIK download includes detailed instructions on how to do this using Windows Installer and a set of merge modules that are part of the integration kit download.

An alternative method is to use the H2Reg utility from helpware.net.

H2Reg is a shareware executable that can be included in your help title deployment projects. It can be used to register help titles, and plug them into the VS.NET help system.

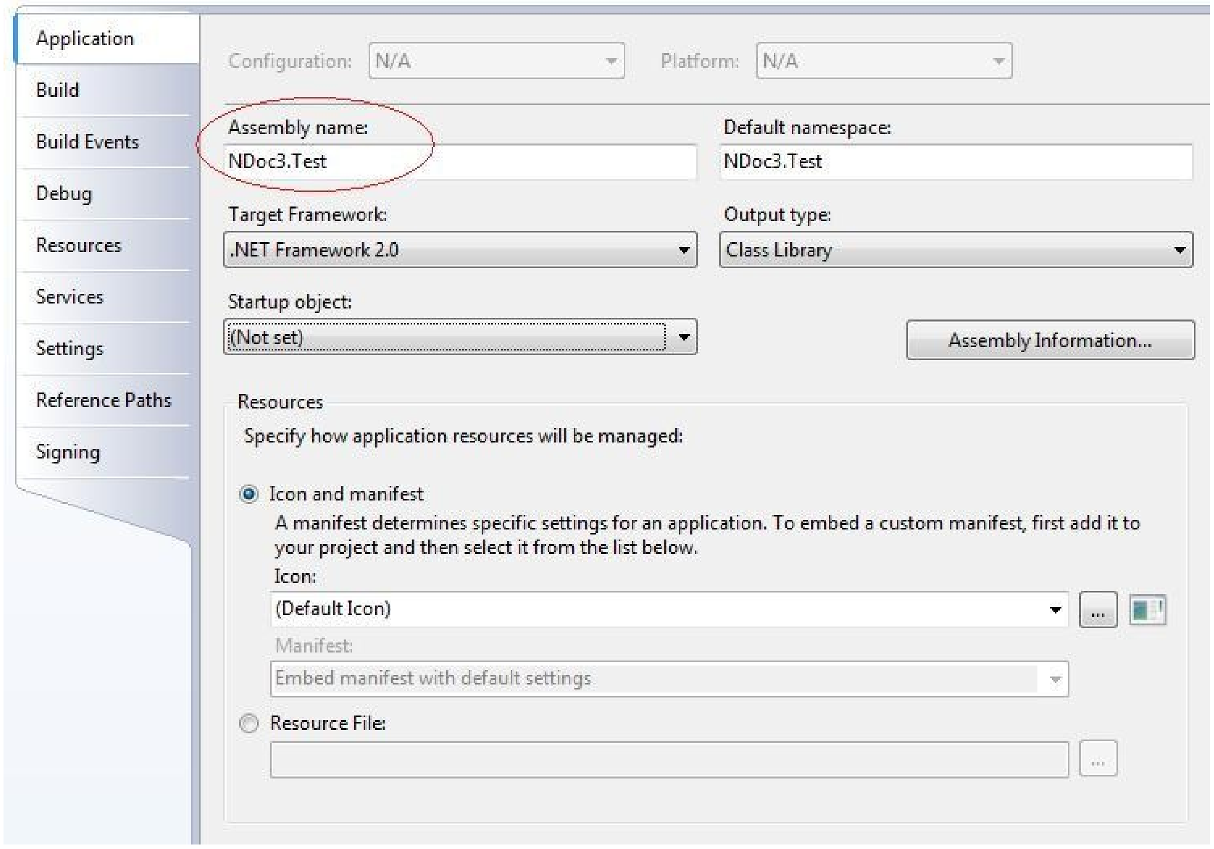
H2Reg uses an INI file to describe the help title and to control how it is registered and plugged into existing help collections. The VS.NET Documenter includes an option that will generate an H2Reg compatible INI file for your NDoc3 created HTML Help 2 titles.

The first thing you want to do is make sure that Visual Studio.NET is creating the XML documentation file each time it recompiles your assembly projects. You do this by setting the "XML Documentation File" property in the project setting dialog.

While not absolutely necessary, it is best to name the XML documentation file with the same base name as the name of your assembly

This is done differently in Visual Studio 2005 and Visual Studio 2008, therefore both methods will be described

Visual Studio 2008 and 2010

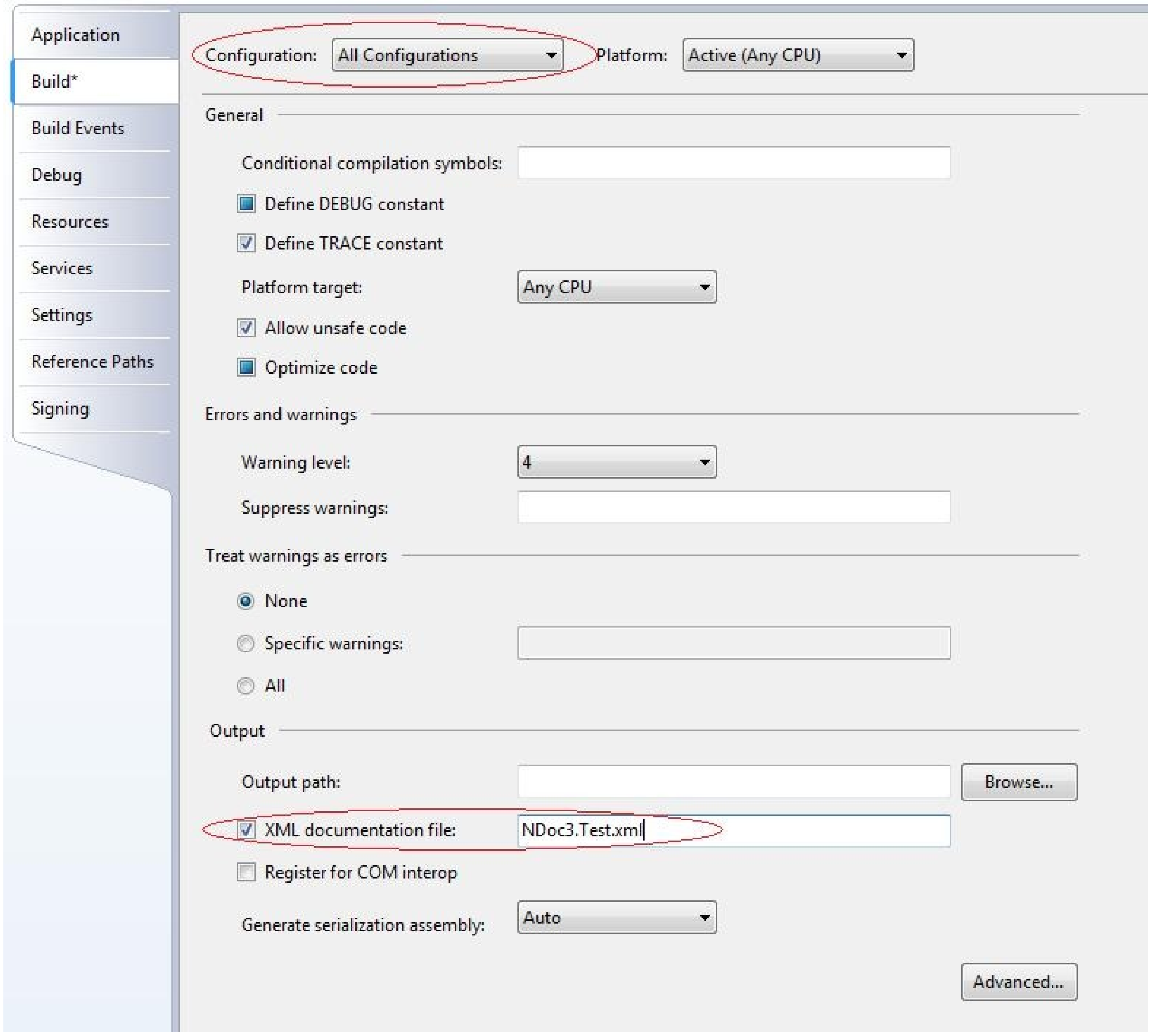


Configuration: N/A
Platform: N/A
Assembly name: NDoc3.Test
Default namespace: NDoc3.Test
Target Framework: .NET Framework 2.0
Output type: Class Library
Startup object: (Not set)

Resources
Specify how application resources will be managed:
A manifest determines specific settings for an application. To embed a custom manifest, first add it to your project and then select it from the list below.
Icon: (Default Icon)
Manifest: Embed manifest with default settings

Finding the assembly name

Set the XML Documentation File property to the assembly name, but with the .xml extension. do not forget to select the "All Configurations" option before you set this property. That way you can document both release and debug builds.



Configuration: All Configurations
Platform: Active (Any CPU)

General

Conditional compilation symbols:
Define DEBUG constant
[✓] Define TRACE constant
Platform target: Any CPU
[✓] Allow unsafe code
Optimize code

Errors and warnings

Warning level: 4
Suppress warnings:

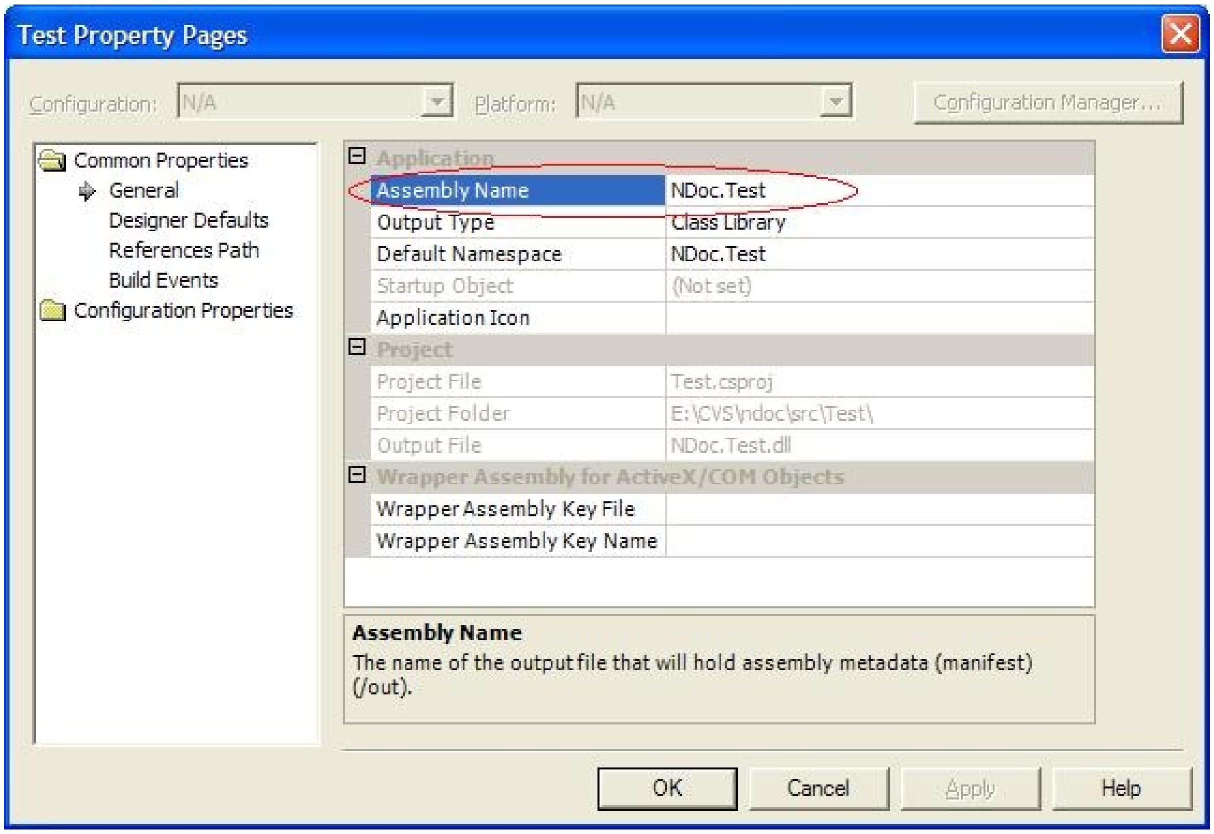
Treat warnings as errors

● None
○ Specific warnings:
○ All

Output

Output path:
[✓] XML documentation file: NDoc3.Test.xml
Register for COM interop
Generate serialization assembly: Auto

Visual Studio 2005

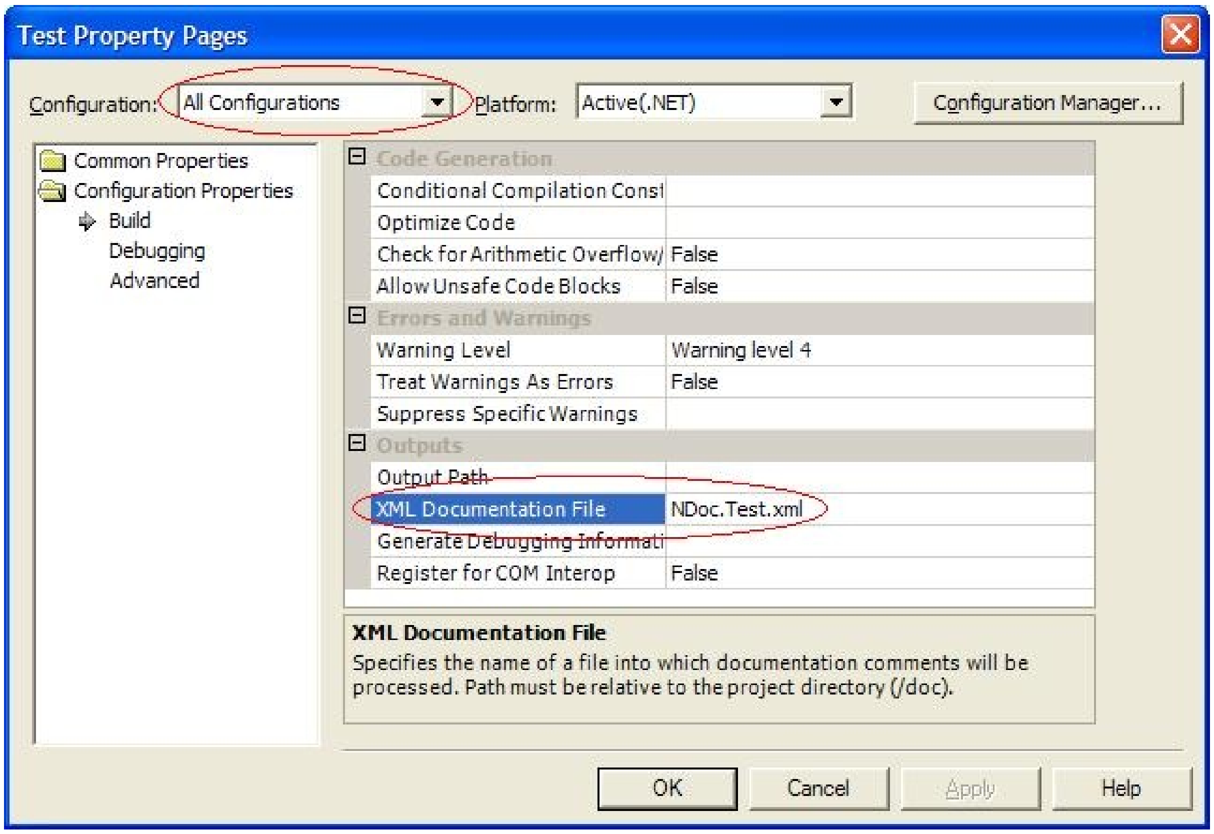


Application	
Assembly Name	NDoc.Test
Output Type	Class Library
Default Namespace	NDoc.Test
Startup Object	(Not set)
Application Icon	
Project	
Project File	Test.csproj
Project Folder	E:\CVS\ndoc\src\Test\
Output File	NDoc.Test.dll
Wrapper Assembly for ActiveX/COM Objects	
Wrapper Assembly Key File	
Wrapper Assembly Key Name	

Assembly Name
The name of the output file that will hold assembly metadata (manifest) (/out).

Finding the assembly name

Set the XML Documentation File property to the assembly name, but with the .xml extension. do not forget to select the "All Configurations" option before you set this property. That way you can document both release and debug builds.



Code Generation	
Conditional Compilation Consi	
Optimize Code	
Check for Arithmetic Overflow/	False
Allow Unsafe Code Blocks	False
Errors and Warnings	
Warning Level	Warning level 4
Treat Warnings As Errors	False
Suppress Specific Warnings	
Outputs	
Output Path	
XML Documentation File	NDoc.Test.xml
Generate Debugging Informati	
Register for COM Interop	False

XML Documentation File
Specifies the name of a file into which documentation comments will be processed. Path must be relative to the project directory (/doc).

Setting the XML Documentation Property

Now, each time VS.NET compiles your assembly, it will aggregate all of the code comments that you include in the source files, in the XML Documentation File.

Non-Visual Studio

If you do not use Visual Studio.NET, make sure to compile your C# projects using the /doc compiler option.

The more, the better

The more code comments you add to your code, the more descriptive the generated help topics will be. That makes your help file more useful to the users of your assemblies.

At a minimum each public type, and the public and protected members of your public types, should have a **<summary>** item describing what the member does or represents.

The VS.NET C# code editor has a handy feature that makes it easy to create the basic code comments for each type and member:

For the following code snippet:

```
public class MyClass() {
     public MyClass( string s ) { }
}
```

if you place your cursor just above the **MyClass** constructor, and hit the '/' character three times in a row, VS.NET will create the skeleton of a code comment block for that member:

```
public class MyClass() {
     /// <summary>
     ///
     /// </summary>
     /// <param name="s"></param>
     public MyClass( string s ) { }
}
```

This applies to any type or member that can have code comment tags associated with it. Further more, once you have a code comment block, when you hit the '<' key to start a new tag, VS.NET will display an intellisense selector showing the appropriate list of code comment tags. Unfortunately this list won't include the additional tags that NDoc3 supports, but you can still add them by hand.

NDoc3 includes options that allow you to generate documentation for private and internal items. This is useful when you are using NDoc3 to generate in-house documentation. If you plan to use NDoc3 in this way you should also add code comments to types and members that would not normally be visible outside of the assembly.

The MSDN and VS.NET documenters that ship with NDoc3 support all the xml documentation tags defined in the C# Programmer's Reference.

They also support new tags, and extended syntax for some standard tags, as described below.

Some tags may only be used on certain types or members, see the Tag Usage Matrix or individual tag reference pages for further details.

Section Tags

Section tags are used to define the content of the different sections of the documentation of a type or member.

These tags are used as top-level tags.

Tag	Description
<event> [NDoc3 Only]	An event that may be raised by a member.
<inheritdoc /> [NDoc3 Only]	Indicates that the documentation should be inherited
<example>	An example of how to use a type or member.
<exception>	An exception that may be thrown by a member.
<exclude/> [NDoc3 Only]	Directs the documentation tool to exclude the taggec The tag overrides all visibility options.
<include>	References an xml node in an include file that contai
<overloads> [NDoc3 Only]	Documentation that applies to all the overloads of a first overload. The **<overloads>** tag can have two forms:

	• In the short form, it includes only one o summary.
	• In the long form, it can include one or r remarks and example).
	Example:
	`///<overloads>This method has two overl`
	`///<summary>This overload just says hel`
	`public void SayHello() { ... }`
	`///<summary>This one says hello to some`
	`public void SayHello(string toSomeone)`

<param>	A member parameter.
<permission>	Security permissions required to access a member.
<preliminary> [NDoc3 Only]	Marks the documentation of a type or member as pr **<para>** in order to put a custom warning into your he If the tag is empty, preliminary topics will include the [This is preliminary documentation and subject to ch It is also possible to mark an entire help project as p setting.
<remarks>	Additional information about a type or member, supp
<returns>	A member's return value.
<seealso>	Adds a link to the See Also section. Do **not** include this tag in the **<remarks>** section. Alternate syntax: • `<seealso href="url">[label]</` • `<seealso cref="member">[label`

<summary>	A short description of a type or member.

<threadsafety>
[NDoc3 Only]
	Denotes how a class or structure behaves in multi-th
	
	This tag has 2 attributes **static** and **instance** that ca is that static members are thread safe and instance r
	
	Textual content can be included as the inner text of t information about the thread safety of the type.
	
	`/// <summary>This overload just says he`
	`/// <threadsafety static="true" instanc`
	`///     <para>More information about us:`
	`/// </threadsafety>`
	`public class SafeClass() { ... }`

<value>	Defines a property value.

Block Tags

Block tags format text within the top-level tags. They are typically used to add structure to the text inside **<remarks>** and **<example>** sections .

Tag	Description
<code>	A block of code.
<list>	A definition list or table.

<note>
[NDoc3 Only]
	A formatted note block.
	
	Example:
	
	`/// <summary>`
	`/// <note>This is a note in the summary.</not`
	`/// </summary>`
	`public class SomeClass() { ... }`
	
	gives,

	Note This is a note in the summary.
<para>	Delimits a text paragraph.

Inline Tags

Inline tags are typically used inside **<para>** blocks.

Tag	Description
<c>	Marks inline code.
<paramref>	A reference to a parameter.
<see>	An Inline reference to another member or type. Alternate syntax: • `<see href="url">[label]</see>` • `<see cref="member">[label]</see>` • `<see langword="xxx"/>` where *xxx* is **null**, **sealed**, **static**, **abstract** or **virtual**

The MSDN and VS.NET documenters that ship with NDoc3 use a number of custom attributes to augment the code comment tags when generating documentation.

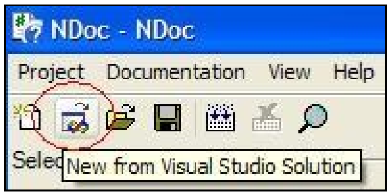
When a type or member is decorated with the attributes described below, NDoc3 will use this information in order to make the help file more descriptive.

Attribute	Comments
ObsoleteAttribute	Marks program elements that are no longer in use. NDoc3 will include warning text for each item that is marked as obsolete.
FlagsAttribute	Indicates that an enumeration can be treated as a bit field; that is, a set of flags. NDoc3 enhances the documentation as follows, • A standard sentence is added after the summary section. "This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values." • An additional column is added to the member table to document the

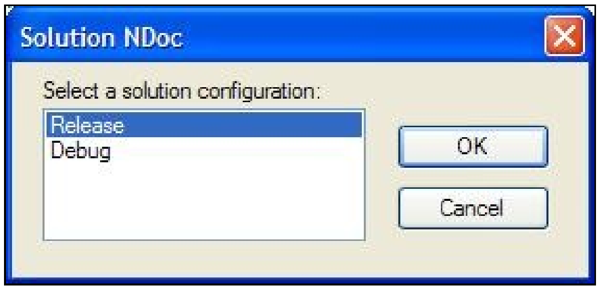
	underlying value of each enumeration member.
EditorBrowsableAttribute	This attribute prevents types of members from showing up in editor or type browsers. If the project setting **EditorBrowsableFilter** is set to **true**, then no documentation will be generated for items decorated with this attribute. The **<exclude/>** tag is the preferred mechanism for suppressing the documentation of types or members.
AssemblyVersionAttribute	If the **IncludeAssemblyVersion** project setting is **true**, then the assembly version will be included on each topic page.

Creating a New Project and Adding Assemblies

If you use Visual Studio.NET as your development environment, the easiest way to create a new NDoc3 project is to select the "New From Visual Studio" button on the toolbar.



After browsing to your solution file, you will be asked to select a configuration within the solution. This determines what assemblies (debug or release versions) will be used to generate your documentation.



The Configuration Selection Dialog

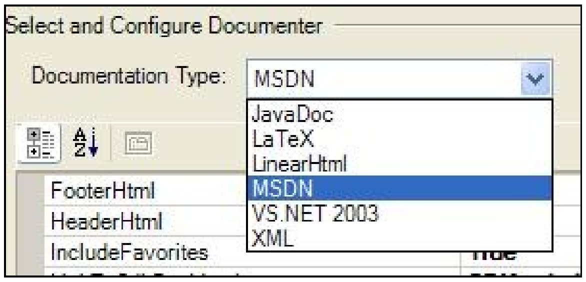
Once you select a configuration, the NDoc3 project designer will add all of the managed assembly projects from that solution to your new NDoc3 project.

If you don't use Visual Studio.NET, you can add assemblies to your project either by browsing for them with the **Add** button, or dragging them into the assembly list view directly from Window Explorer. Make sure that you are generating XML documentation during your compiles, otherwise NDoc3 will only be able to produce minimal documentation...

Choosing a Documenter

NDoc3 ships with a number of different options for generating documentation for your code. Each different format is referred to as a **Documenter**. Each Documenter determines it own layout, formatting, and in some cases content for the generated documentation.

The two most popular documenters are the MSDN documenter, which generate compiled CHM type documentation similar to MSDN online help, and the VS.NET documenter, which generates compiled html help compatible with Visual Studio.NET.



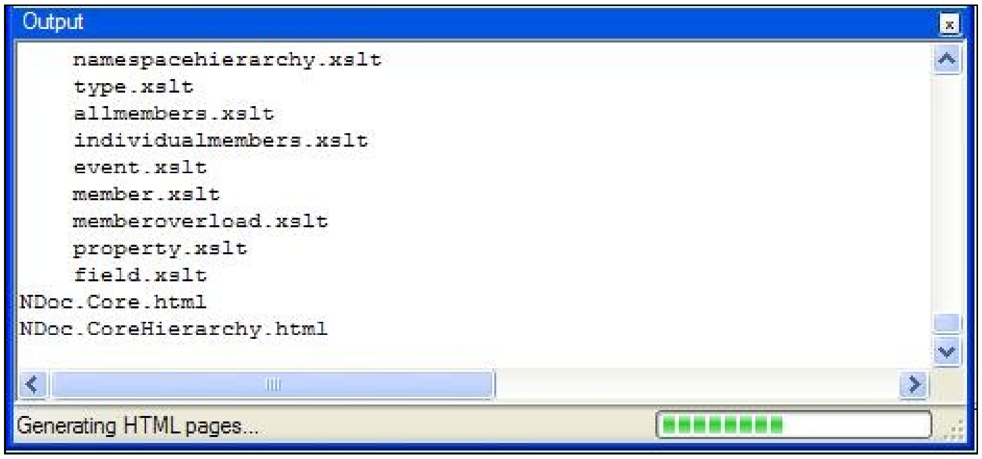
NDoc3 Documenters

All documenters share certain settings, such as what types of members to document. Each documenter also has its own set of custom settings which you can use to control the exact content of your documentation.

For more details about configuring documenters, refer to the Documenters section.

Building the Documentation

Once you've chosen a Documenter, and configured it to your liking, hit the Build Button to create your documentation. As the project builds and compiles, you will see build status in the Output Window at the bottom of the Designer Screen, as well as in a progress indicator on the status bar.



```
    namespacehierarchy.xslt
    type.xslt
    allmembers.xslt
    individualmembers.xslt
    event.xslt
    member.xslt
    memberoverload.xslt
    property.xslt
    field.xslt
NDoc.Core.html
NDoc.CoreHierarchy.html
```

Generating HTML pages...

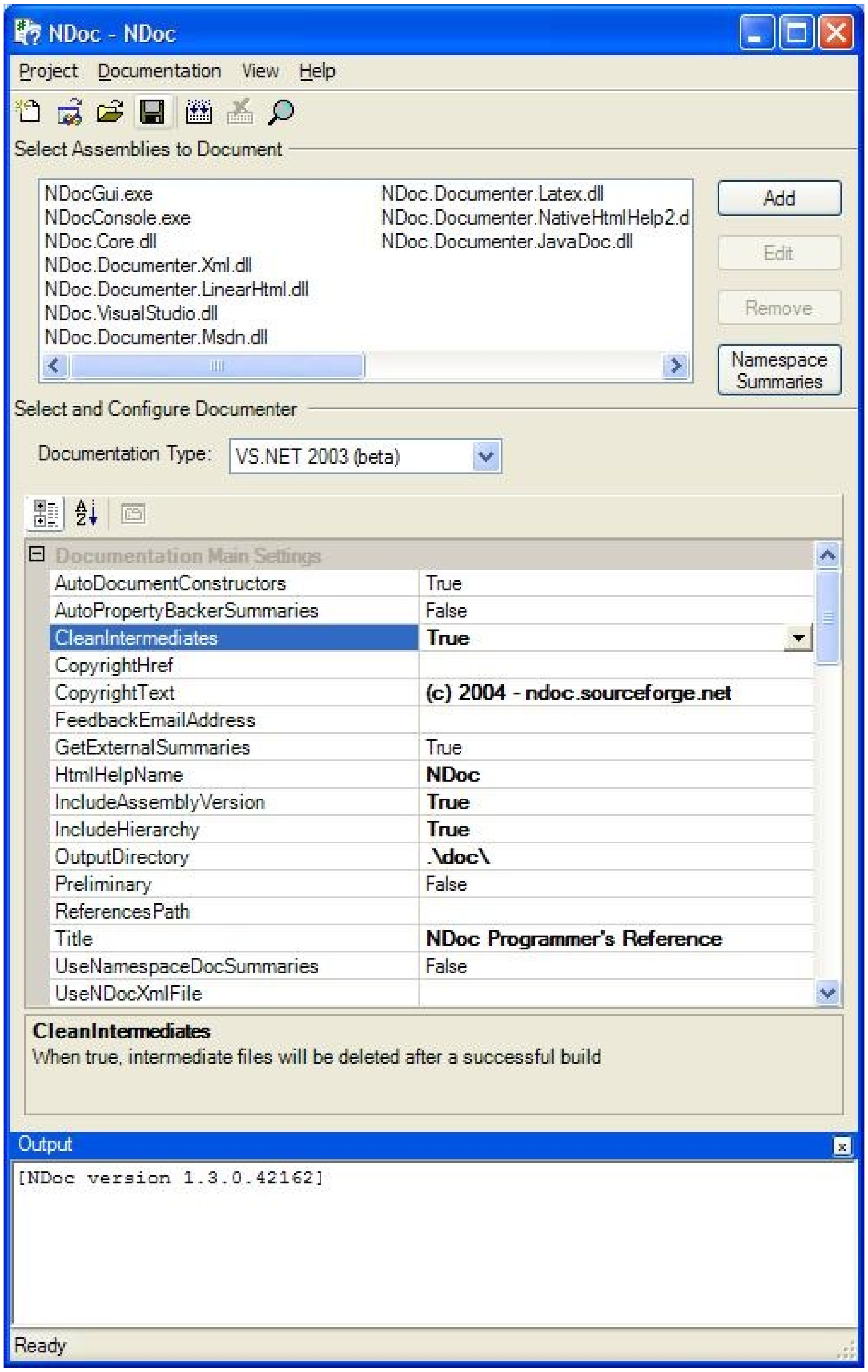
NDoc3 Build Progress

Viewing the Documentation

Once the build completes, you can view the resulting documentation by clicking the View button.

Overview

The assemblies that you want to document are organized in a project file. The project file keeps the list of assemblies as well as the settings you set that determine how your documentation will appear.


NDoc - NDoc
Project Documentation View Help
Select Assemblies to Document
Add
Edit
Remove
Namespace Summaries
Select and Configure Documenter
Output
Ready

NDocGui.exe
NDocConsole.exe
NDoc.Core.dll
NDoc.Documenter.Xml.dll
NDoc.Documenter.LinearHtml.dll
NDoc.VisualStudio.dll
NDoc.Documenter.Msdn.dll
NDoc.Documenter.Latex.dll
NDoc.Documenter.NativeHtmlHelp2.d
NDoc.Documenter.JavaDoc.dll

Documentation Type: VS.NET 2003 (beta)

Documentation Main Settings

AutoDocumentConstructors	True
AutoPropertyBackerSummaries	False
CleanIntermediates	**True**
CopyrightHref	
CopyrightText	**(c) 2004 - ndoc.sourceforge.net**
FeedbackEmailAddress	
GetExternalSummaries	True
HtmlHelpName	**NDoc**
IncludeAssemblyVersion	**True**
IncludeHierarchy	**True**
OutputDirectory	**.\doc**
Preliminary	False
ReferencesPath	
Title	**NDoc Programmer's Reference**
UseNamespaceDocSummaries	False
UseNDocXmlFile	

CleanIntermediates
When true, intermediate files will be deleted after a successful build

```
[NDoc version 1.3.0.42162]
```

The NDoc3 Project Designer

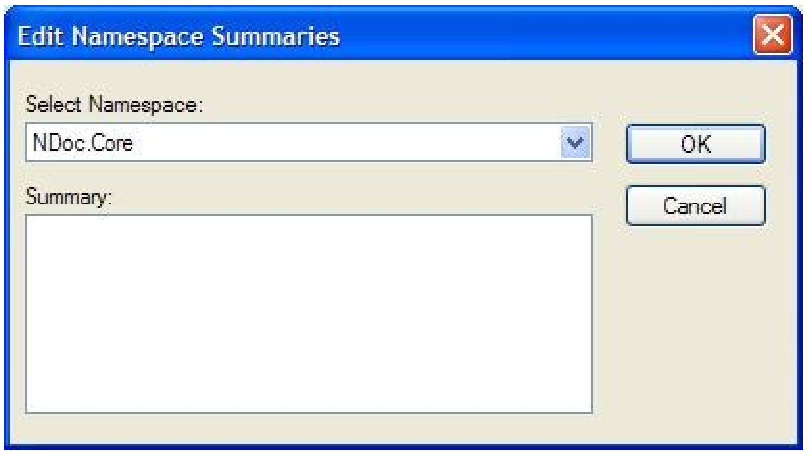
Creating a new NDoc3 project is merely a matter of selecting the assemblies that will be included, selecting a documentation style, and setting the properties of your documentation.

Namespace Summaries

The standard C# documentation tags don't include any way to provide a summary for a namespace. NDoc3 supports two mechanisms for specifying namespace summaries. One is by supplying them in a special code file within your project. The other is via the project designer.

In the project designer, click the "Namespace Summaries" button and you will be presented with a dialog that allows you to set a namespace summary for each namespace in your project.

These summaries will be included in the compiled documentation



The Namespace Summary Dialog

Documenter Settings

Each Documenter shares some basic settings, like what types of members to document and where to create the final output. Each Documenter also has its own specific settings that govern its operation. These settings can be modified in the Project Designer and saved as part of your project.

Settings are organized by category. When you select a setting in the property control, you will see a short explanation of what the setting does. This is a good way to familiarize yourself with how each Documenter works and what it is capable of.

For a complete list of the documenters and their settings, refer to the Documenters Topic.

Option Descriptions

Option	Description
ShowProgressOnBuild	If **true**, the build progress window will automatically be shown whenever a build starts.
LoadLastProjectOnStartup	If **true**, NDoc will load the last project when it starts. This is a per user setting.
MRUSize	The number of items to display in the most recently used project list.
HtmlHelpWorkshopLocation	This option only applies to using the MSDN documenter. By Default NDoc will look in a couple of well known locations for the Html Help v1 compiler (HHC.EXE). If you have the help compiler installed in a non-standard location, and NDoc is telling you that it cannot find the Html Help Workshop, set this property to the path where you have the compiler located. This setting is a machine specific setting and will apply to any user logged in.

Overview

The NDoc3 console application (**NDoc3Console.exe**) exposes the full building capability of the GUI application and designed to be integrated into automated build process.

Syntax

The console application uses a simple name-value-pair syntax for specifying options. Named options are prefixed with a dash in the following fashion: **-name=value**, with no white-space after the dash or around the equals sign. All parameters described below that are enclosed in square brackets are optional. Paths containing spaces need to be quoted.

Usage 1

```
NDoc3Console  assembly[,xmldoc] [assembly[,xmldoc]]
                   [[-referencepath=dir] [-referenc
                   [-namespacesummaries=filename]
                   [-documenter=documenter_name]
                   [[-property=value] [-property=va
                   [-verbose]
```

where:

- **assembly** is the full path to an assembly to be documented
- **xmldoc** is the full path to the compiler generated /doc file

  *if not specified, NDoc3 will look (in the same directory as **assembly**) for a file with the name of the assembly but with an extension of .xml instead of .dll or .exe*
- **referencepath** is the full path to a directory where referenced assemblies can be located
- **namespacesummaries** is the full path to a namespace summary XML document
- **documenter** is the name of the documenter to use

if not specified, the MSDN documenter will be used

- **property** is the case sensitive name of a property to set on **documenter**
- **value** is the value to set **property** to
- **verbose** causes full progress information to be displayed during the build

Usage 2

```
NDoc3Console -recurse=dir[,maxDepth]
                [[-referencepath=dir]
                [-referencepath=dir]...]
                [-namespacesummaries=filename]
                [-documenter=documenter_name]
                [[-property=value] [-property=value]
                [-verbose]
```

where:

- **recurse** will document all assemblies in the specified directory if they have a /doc XML file named with the name of the assembly but with an extension of .xml instead of .dll or .exe
- **maxDepth** is the maximum depth of sub-directories below **dir** to search
- **referencepath** is the full path to a directory where referenced assemblies can be located
- **namespacesummaries** is the full path to a namespace summary XML document
- **documenter** is the name of the documenter to use

  if not specified, the MSDN documenter will be used
- **property** is the case sensitive name of a property to set on **documenter**
- **value** is the value to set **property** to
- **verbose** causes full progress information to be

displayed during the build

Usage 3

```
NDoc3Console [-documenter=documenter_name]
               -project=ndocfile
               [-verbose]
```

where:

- **documenter** is the name of the documenter to use

  if not specified, the MSDN documenter will be used
- **project** is the full path to an NDoc3 project file
- **verbose** causes full progress information to be displayed during the build

Usage 4

```
NDoc3Console [-help]
               [documenter_name [property_name]]
```

where:

- **help** displays help information

  *if **documenter_name** is not specified, basic usage syntax will be displayed*
- **documenter_name** displays help about a specific documenter

  *if **property_name** is not specified a list of all the settable properties on **documenter_name** will be displayed*
- **property_name** is the name of the property on **documenter_name** to display help about

Available Documenters

The currently supported documenter names are: **JavaDoc**, **LaTeX**, **LinearHtml**, **MSDN**, **VS.NET_2003**, and **XML**.

Namespace Summaries File Syntax

```
<namespaces>
        <namespace name="My.NameSpace">My summary.</r
        <namespace name="My.OtherNameSpace">My other
        ...
</namespaces>
```

Using NAnt

No tutorial available yet. You can contribute a tutorial by sending a mail.

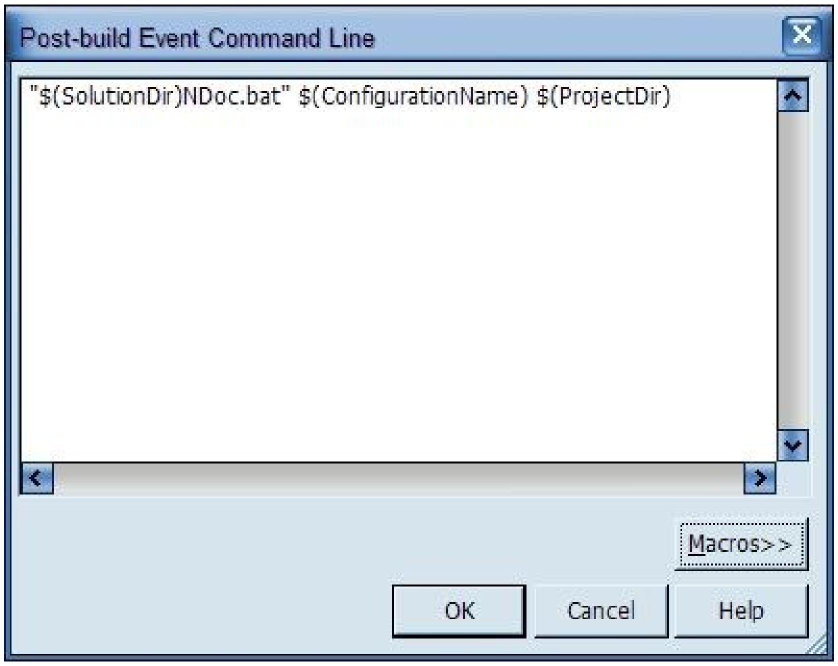
Using VS.NET Build Events

Build events can be used to invoke NDoc3 when build a VS.NET solution. Because NDoc3 builds can be time consuming it is recommended that they only be included in Release builds or in a special documentation build configuration.

Because build events for C# projects are not configuration based, it is necessary to check the configuration name if you do not want NDoc3 to run every time a build occurs. This can be accomplished using VS.NET's built in build variables:

```
if $(ConfigurationName) == Release
```

Because specifying all of NDoc3's command line paramters directly in the build event editing window can be difficult and error prone, calling a batch file can be easier to manage and maintain. Use the Post-Build event for the last project in the solution that will be built or for each project for which you want seperate documentation.



```
"$(SolutionDir)NDoc.bat" $(ConfigurationName) $(ProjectDir)
```

Calling a batch file from the post-build event

For simple solutions the following batch file will invoke NDoc3 for each Release build.

```
IF NOT %1 == Release GOTO end

"%ProgramFiles%\NDoc3\bin\NDoc3Console.exe" -recurs

:end
```

More complex solutions may require additional parameters such as reference directories etc.

Using Other Build Tools

NDoc3 can be included in builds orchestrated by any build tool that supports executing command line applications. The exact mechanism will be specific to the tool, but most (if not all) support arbitrary execution of executables with paramters.

Overview

A Documenter represents a particular method of creating, organizing, formatting and packaging your documentation. NDoc3 ships with six Documenters:

- VS.NET
- MSDN
- XML
- JavaDoc
- LinearHtml
- LaTeX

NDoc3 uses a pluggable architecture, so it is also possible to generate your own, custom, documenters.

Generating the documentation consists of two basic steps:

1. Merging the /doc XML summary with reflected meta-data from the assemblies.
2. Transforming that merged XML into the documentation (HTML for the MSDN and VS.NET documenters).

The settings below govern how exactly the XML summary data is merged with the reflected meta-data and therefore govern what items will and will not appear in the final documentation.

These settings are shared by all of the current NDoc3 documenters. Particular documenters might include additional steps like compiling the transformed HTML into compiled help files. They will have additional settings that govern the details specific to the documenter's format.

Settings

Setting	Description
(Global)	
ReferencePaths	The directories used t

	references.
Documentation Main Settings	
AssemblyVersionInfo	Determines what type information is docume Valid values are, **None** **Assembly Version** **File Version**
AutoDocumentConstructors	If **true**, automatic sum default constructors w summary for a parame present, the default co **Initializes a new inst class** is inserted.
AutoPropertyBackerSummaries	If **true**, the documente summary for fields wh

	the value for) a proper added if there is no ex gives you a way to op particular cases. Curr conventions supporte **_Length** and **length** v **Length**.
CleanIntermediates	If **true**, intermediate fil successful build. For documenters that like the MSDN and VS intermediate files inclu project files, as well as
CopyrightHref	The URI of a copyrigh will be included with e
CopyrightText	A textual copyright no with each topic.
FeedbackEmailAddress	If an email address is be placed at the botto this address.
Preliminary	If **true**, NDoc3 will ma preliminary document include a notice that th preliminary The default notice is: | documentation and su
SdkDocLanguge	Specifies to which lan Framework SDK docu system types will be p
SdkDocVersion	Specifies to which ver Framework SDK docu system types will be p
UseNamespaceDocSummaries	If **true**, the documente

	the name **Namespace** The summary from tha the namespace summ show up in the resultir You may want to use # conditional compilatior **NamespaceDoc** class assemblies.
UseNDocXmlFile	When set, NDoc3 will as input instead of refl specified on the projec Very useful for debugg *empty for normal usag*
Show Attributes	
DocumentAttributes	If **true**, custom attribut members are docume topics.
DocumentedAttributes	When **DocumentAttri** specifies which attribu Empty to show all. Format: '<attribute-nar to-show>,<property-to starts-with>,<property show>|(etc...)'.
DocumentInheritedAttributes	When **DocumentAttri** specifies whether attri classes are visible
ShowTypeIdInAttributes	Set this to **true** in orde property in the attribut
Show Missing Documentation	
ShowMissingParams	If **true**, all parameters comments will contain

	Documentation in the
ShowMissingRemarks	If **true**, all members w comments will contain Documentation in the
ShowMissingReturns	If **true**, all members w comments will contain Documentation in the
ShowMissingSummaries	If **true**, all members w comments will contain Documentation in the
ShowMissingValues	If **true**, all properties v comments will contain Documentation in the
Visibility	
DocumentEmptyNamespaces	If **true**, empty namesp documentation. Norma not documented.
DocumentExplicitInterfaceImplementations	If **true**, members that interfaces will be inclu Normally, these memb
DocumentInheritedFrameworkMembers	If **true**, members inhe classes will be include convention, these mei documented. Note: if **DocumentInh** this setting will be ign
DocumentInheritedMembers	If **true**, inherited mem documentation. By co are normally documer
DocumentInternals	If **true**, types and mer will be included in the Normally, internal item

DocumentPrivates	If **true**, types and men be included in the doc when using NDoc3 to intended for internal u Normally, private item
DocumentProtected	If **true**, protected men documentation. Since protected memb can be accessed outs true by default.
DocumentProtectedInternalAsProtected	If **true**, NDoc3 will trea members as "protecte
DocumentSealedProtected	If **true**, protected insta classes will be docum *Note: For this setting t* ***DocumentProtected***
EditorBrowsableFilter	Sets the level of filteri types/members marke **EditorBrowsable** attr **Warning: enabling th invalid links in the d**
SkipNamespacesWithoutSummaries	If **true**, NDoc3 will not that do not have an as summary.

Overview

The VS.NET documenter creates compiled Html Help version 2 titles similar in format to the .NET Framework SDK collection. Html Help 2 is the help technology used by the Visual Studio.NET help system as well as by newer version of MSDN and the SDK documentation.

Documentation generated with this documenter can be integrated into Visual Studio and MSDN. This entails inclusion in the combined table of contents, index and search functions, as well as context sensitive and dynamic help from with the Visual Studio IDE.

Settings

All documenters share a common set of documenter settings.

Setting	Description
AboutPageIconPage	HTML file that displays the Help About image.
AboutPageInfo	Displays product information Help About.
AdditionalContentResourceDirectory	Directory that contains resources (images etc.) usec by the additional content pages. This directory will be recursively compiled into the help file.
BuildSeparateIndexFile	If **true** a separate index file is generated, otherwise it is compiled into the HxS (recommended).
CharacterSet	The character set that will be used when compiling the hel file
CollectionNamespace	The Html Help 2 registry namespace (avoid spaces).

	Used in conjunction with **GenerateCollectionFiles** ar **RegisterTitleWithNamespa**
CollectionTOCStyle	Determines how the collectic table of contents will appear the help browser.
CreateFullTextIndex	If **true**, creates a full text inde for the help file.
DocSetList	A comma-separated list of **DocSet** filter identifiers in wh topics in this title will be included.
EmptyIndexTermPage	Displays when a user choose a keyword index term that ha sub keywords but is not direc associated with a topic itself.
ExtensibilityStylesheet	Path to an XSLT stylesheet t contains templates for documenting extensibility tag
FooterHtml	Raw HTML that is used as a page footer instead of the default footer. %FILE_NAME is dynamically replaced by th name of the file for the currer html page. %ASSEMBLY_NAME% is dynamically replaced by the name of the assembly for the current page. %ASSEMBLY_VERSION% i dynamically replaced by the version of the assembly for tl current page. %TOPIC_TITLE% is dynamically replaced by the title of the current page.

GenerateCollectionFiles	If **true**, creates collection file to contain the help title. Thes all the title to be plugged into the Visual Studio help namespace during deployme
HeaderHtml	Raw HTML that is used as a page header instead of the default blue banner. %FILE_NAME%\" is dynamically replaced by the name of the file for the curre html page. %TOPIC_TITLE% is dynamically replaced by th title of the current page.
HtmlHelpName	The HTML Help project file a the compiled HTML Help file use this property plus the appropriate extension as names.
IncludeDefaultStopWordList	If **true**, the default stop-word list is compiled into the help (A stop-word list is a list of words that will be ignored during a full text search)
IntroductionPage	An HTML page that will be displayed when the root TOC node is selected.
LangID	The language ID of the local used by the compiled help fil
LinkToSdkDocVersion	Specifies to which version of the .NET Framework SDK documentation the links to system types will be pointing
NavFailPage	Page that opens if a link to a topic or URL is broken.

OmitSyntaxSection	If **true**, the syntax section on member topics will not be generated (improves performance)
OutputDirectory	The directory in which .html files and the .Hx* files will be generated.
PlugInNamespace	If GenerateCollectionFiles i true, the resulting collection v be plugged into this namespace during deployme
RegisterTitleAsCollection	If **true** , the HxS title will be registered as a collection afte a successful compilation of t documentation. (ignored if RegisterTitleWithNamespa is true) *Note: This is a development and only applies to the mach where the build is run.*
RegisterTitleWithNamespace	Should the compiled Html 2 t be registered on this machin after it is compiled. Good for testing. (If true CollectionNamespace is required) *Note: This is a development and only applies to the mach where the build is run.*
Title	This is the title displayed at t top of every page.
UseHelpNamespaceMappingFile	If the documentation include references to types registere in a separate html help 2

	namespace, supplying a mapping file allows **XLinks** t be created to topics within th namespace.
Version	The version number for the help file (#.#.#.#)

XLinks

Unlike Html Help 1, which uses HTML <A> links, Html Help 2 uses an implementation of XLinks to reference topics in external help titles. An XLink is much like an HTML <A> link but allows for additional linking meta-data and more complex types of links. Html Help 2 uses XLinks to look-up topic by keyword, rather than by file name. This de-couples the link from the physical layout of the target help title.

NDoc generates XLinks to topics in the .NET framework. The following is an NDoc generated link to the Framework SDK topic on **System.Void**:

```
<MSHelp:link
        keywords="frlrfSystemVoidClassTopic"
        indexMoniker="!DefaultAssociativeIndex"
        namespace="ms-
help://MS.NETFrameworkSDKv1.1">void</MSHelp:link>
```

Example XLink

The three attributes on the above link tell the Html Help 2 system exactly how to resolve the desired topic. The first item required is a **namespace**. Each help collection registered on a machine has a unique namespace. The namespace above identifies the .NET Framework 1.1 SDK documentation. The **indexMoniker** identifies a specific type of index within that namespace. There can be numerous types of indices within a help collection, but the **DefaultAssociativeIndex** is used to create associations between topics. Finally, the **keyword** attribute identifies the specific topic desired within the index.

Keywords are defined within a help topic as part of the embedded XML data that is generated in the HTML header. The keyword for **System.Void** is declared as follows within the Html Help for that topic:

```
    <MSHelp:Keyword Index="A"
Term="frlrfSystemVoidMembersTopic"/>
```

Links to Framework Topics

Framework keywords are generated from the full name of the type or member. NDoc generates links to framework topics, either through inheritance or via the <see> tag, by determining the keyword identifier from the item being linked to and within the framework SDK help namespace.

Links to Non-Framework Topics

It is also possible to create links to your own Html Help 2 titles using the same XLinking mechanism. For NDoc to properly generate an XLink to an external title, it needs to know both the help namespace and the keyword value within the **DefaultAssociativeIndex**.

When NDoc generates an Html Help 2 topic, it includes an XML data island that includes an associative index term similar to those that appear within framework topics. Since NDoc can document **private** and **internal** members, the index generated is slightly more complicated than framework keywords, but the keyword value is generated deterministically.

The thing that cannot be determined is the value of the help namespace. The mapping between managed type names and help namespaces is provided by specifying a mapping file via the UseHelpNamespaceMappingFile setting. This setting points to an XML file that conforms to the namespace map XSD schema.

```
<namespaceMap xmlns="urn:ndoc-sourceforge-
net:documenters.NativeHtmlHelp2.schemas.namespaceMap">

    <helpNamespace ns="ms-help://companyX.sharedhelpcollection">

        <managedNamespace ns="CompanyX"/>

    </helpNamespace>

    <helpNamespace ns="ms-help://companyX.producthelpcollection">

        <managedNamespace ns="CompanyX.Product1"/>

        <managedNamespace ns="CompanyX.Product2"/>

     </helpNamespace>

</namespaceMap>
```

This file specifies the managed namespace to help namespace mapping that will be used when creating XLinks to managed types that are not part of the documentation set being processed. Each **helpNamespace** entry can contain **1..n managedNamespace** entries. When resolving the help namespace of a managed type, the most specific **managedNamespace** entry will be used to select the appropriate help namespace.

The managed type **CompanyX.Product2.Class1** would map to **ms-help://companyX.producthelpcollection**, while **CompanyX.Core.Class1** would map to **ms-help://companyX.sharedhelpcollection**.

See Also

The VS.NET Documenter, Namespace Map Schema

Visual Studio Integration

Each Html Help 2 topic includes an XML data island, that is used by the help system for creating indices, linking topics, looking up keywords, filtering topics and a number of other features. A typical XML data island looks like:

```
<xml>
        <MSHelp:TOCTitle Title="Object Class"/>
        <MSHelp:RLTitle Title="Object Class"/>
        <MSHelp:Keyword Index="K" Term="Object class, about Object
        class"/>
        <MSHelp:Keyword Index="A"
        Term="frlrfSystemObjectClassTopic"/>
        <MSHelp:Keyword Index="F" Term="System.Object"/>
        <MSHelp:Attr Name="DocSet" Value="NETFramework"/>
        <MSHelp:Attr Name="TopicType" Value="kbSyntax"/>
        <MSHelp:Attr Name="DevLang" Value="CSharp"/>
        <MSHelp:Attr Name="DevLang" Value="VB"/>
        <MSHelp:Attr Name="DevLang" Value="C++"/>
        <MSHelp:Attr Name="DevLang" Value="JScript"/>
        <MSHelp:Attr Name="DevLang" Value="VJ#"/>
        <MSHelp:Attr Name="Technology" Value="WFC"/>
        <MSHelp:Attr Name="Technology" Value="ManagedC"/>
        <MSHelp:Attr Name="TechnologyVers" Value="kbWFC"/>
        <MSHelp:Attr Name="TechnologyVers" Value="kbManagedC"/>
        <MSHelp:Attr Name="Locale" Value="kbEnglish"/>
        <MSHelp:Attr Name="DocSet" Value="NETCompactFramework"/>
        <MSHelp:Attr Name="TechnologyVers" Value="kbProfile2NETCF"/>
        <MSHelp:Attr Name="HelpPriority" Value="2"/>
```

```
</xml>
```

When NDoc generates an Html Help 2 topic, it creates an XML data island similar. This allows the help title to be integrated into Visual Studio.NET in such a way that your topics will show up in the help index, the help contents, the search pane, the dynamic help pane and even in context sensitive help from the code editor. This level of integration makes it very easy for consumers of your code to get the information they need right from the development environment and is one of the most valuable features of Html Help 2.

Requirements

Even though NDoc generates the necessary XML data island for each topic, the nature of the Html Help 2 system requires some additional information in order to integrate generated help with VS.NET.

1. The help title must be part of a collection.

   Html Help 2 defines two levels of containment for help sets: collections and titles. A title is a single set of related Html Help topics, compiled into an HxS file. A collection is one or more related sets of help title. (Confusing the issue: help titles can also be treated as collections, but not vice-a-versa). Each help collection must be assigned a unique namespace.

   Only help collections can be integrated into the VS.NET help system.

2. The collection must be registered on each machine.

   The Html Help 2 system maintains a registry (not to be confused with the windows registry) of all of the help topics currently installed on the machine. This registry is used when traversing links, doing searches, displaying indices etc. It is an essential part of the implementation of Html Help 2. For this reason the help collection needs to be registered when it is deployed, otherwise it will not be accessible from VS.NET.

3. The registered help collection must be "plugged-in" to the VS.NET help namespace.

Any registered help collection can be referenced from any other registered help collection. This allows one collection to include the title in another collection, even though those titles are not directly part of the parent collection. It is via this "plug-in" mechanism that third party help collections can be added to the VS.NET help system.

The namespace for the VS.NET 2003 help collection is: **MS.VSCC.2003**

Confused yet?

Html Help 2 is fairly complex technology. Further information can be found in the VSHIK documentation. The website helpware.net also has a lot of useful information and tutorials on Html Help 2. HelpWare also has a shareware tool by the name of FAR that will prove invaluable for exploring the capabilities of the Html Help 2 system.

The VS.NET documenter has a number of settings meant to simplify this entire process.

- First, make sure that GenerateCollectionFiles is set to true. This will generate the necessary Html Help 2 collection meta data files that will allow you to plug into VS.NET.
- Second, supply a value for CollectionNamespace. Don't use spaces or any URI special characters. It is also a good idea to make an effort to assure that this namespace will be globally unique. The same rules you use to generate unique managed namespaces also work well here.
- Third, make sure you supply the correct value for PlugInNamespace. This makes sure that when you deploy your help files with h2reg.exe, they will be plugged into VS.NET. (The default value of **ms.vscc** allows h2reg to decide at install time whether VS.NET 2002 or 2003 should be used.)

Filters

Each help topic's XML data island can have one or more DocSet values. The DocSet is what VS.NET uses when it filters the help index and search panes. DocSets are defined at the collection level, and then individual topics can be included in a DocSet via its XML data island.

MSDN defines a number of set filters as follows:

```
Windows Client SDK,             Query: "DocSet"="WCS
NET Framework,                  Query: ("DocSet"="NE
Visual Studio Macros,           Query: "DocSet" = "V
Visual Basic,                   Query: "DocSet" = "V
Visual C++,                     Query: "DevLang" = "
Platform SDK,                   Query: "DocSet" = "P
(no filter),                    Query:
Enterprise Servers,             Query: "DocSet" = "N
Internet Development,           Query: "DocSet" = "D
Visual Studio,                  Query: "DocSet" = "V
Visual C#,                      Query: "DocSet" = "C
Samples,                        Query: "TopicType"="
Visual FoxPro,                  Query: "DocSet" = "V
Knowledge Base,                 Query: "DocSet" = "k
Visual J#,                      Query: "DocSet" = "V
.NET Compact Framework,         Query: "DocSet"="Sma
```

Adding your collection to any of the above sets is as easy as supplying a comma separated list via the DocSetList property. An entry will be made in each topic's data island for each item in this list. When you deploy your help, your topics will show up when the user is filtering by sets which you have defined.

NDoc will however create and register a single DocSet filter corresponding to the value of HtmlHelpName, and each topic will be included in that set. Defining additional custom filters is possible, but is outside of the scope of NDoc. To do so you will need to delve more deeply into the deployment and registration process as described in VSHIK.

See Also

The VS.NET Documenter, VSHIK documentation, Deploying Html Help 2

Deploying Html Help 2

The Html Help 2 system maintains a registry of all help collections and titles currently installed on a machine. This registry determines what help titles are included in each help collection as well as maintains references between help collections.

In order to view an Html Help 2 title or collection it is required that it be registered. This makes deploying Html Help 2 documentation more complex than simply delivering a single file as with CHM based help.

Windows Installer

It is possible to create Windows Installer packages to deploy and properly register help collections and titles. VSHIK includes detailed instructions on how to go about this as well as a set of merge modules that contain the registration actions. Unfortunately, it is a rather involved process that includes a number of manual steps to create the proper records in the installer database. There is currently no way to automate the generation of Windows Installer packages for Html Help 2 files.

H2Reg

Another option is to use the shareware tool H2Reg.exe from helpware.net. H2Reg is a command line utility that will register help collections and titles during installation. It can be included in scripted installer as well as Windows Installer packages as a custom action.

If the setting GenerateCollectionFiles is **true**, NDoc will create an H2Reg compatible INI file that contains the proper values to register the generated help title, as well as plug it into the VS.NET help collection.

In order to deploy the NDoc generated Html Help 2 files with H2Reg, follow these steps:

1. Set **GenerateCollectionFiles** to **true**
2. Include all of the generated help files as well as the generated INI file in your installer.

3. Include the H2Reg executable as well as the file **H2Reg.ini** (located in the directory where you installed H2Reg) in your installer.
4. During installation copy the help files as well as the generated INI file to there final location.
5. Execute H2Reg with the following syntax:

   ```
   H2reg -r "CmdFile=<full path to the
   generated INI file>"
   ```

6. During uninstallation, before the help files are removed, execute H2Reg with the following syntax:

   ```
   H2reg -u "CmdFile=<full path to the
   generated INI file>"
   ```

See Also

The VS.NET Documenter, GenerateCollectionFiles, VSHIK Deployment Instructions, H2Reg online

Overview

Settings

All documenters share a common set of documenter settings.

Setting	Description
BinaryTOC	Create a binary table-of-contents file. This can significantly reduce the amount of time required to load a very large help document.
ExtensibilityStylesheet	Path to an xslt stylesheet that contains templates for documenting extensibility tags.
FilesToInclude	Specifies external files that must be included in the compiled CHM file. Multiple files must be separated by a pipe ('|').
FooterHtml	Raw HTML that is used as a page footer instead of the default footer. "%FILE_NAME%\" is dynamically replaced by the name of the file for the current html page. "%ASSEMBLY_NAME%\" is dynamically replaced by the name of the assembly for the current page. "%ASSEMBLY_VERSION%\" is dynamically replaced by the version of the assembly

	for the current page. "%TOPIC_TITLE%\" is dynamically replaced by the title of the current page.
HeaderHtml	Raw HTML that is used as a page header instead of the default blue banner. "%FILE_NAME%\" is dynamically replaced by the name of the file for the current html page. "%TOPIC_TITLE%\" is dynamically replaced by the title of the current page.
HtmlHelpName	The HTML Help project file and the compiled HTML Help file use this property plus the appropriate extension as names.
IncludeFavorites	Turning this flag on will include a Favorites tab in the HTML Help file.
OutputDirectory	The directory in which .html files and the .chm file will be generated.
OutputTarget	Sets the output type to HTML Help (.chm) or Web or both.
RootPageContainsNamespaces	If **true**, the Root Page will be made the container of the namespaces in the table-of-contents. If false, the Root Page will be made a peer of the namespaces in the table-of-contents.
RootPageFileName	The name of an html file to

	be included as the root home page. **SplitTOCs** is disabled when this property is set.
RootPageTOCName	The name for the table-of-contents entry corresponding to the root page. If this is not specified and **RootPageFileName** is, then the TOC entry will be 'Overview'.
SdkLinksOnWeb	If **true**, links to system types and members will point to the online MSDN library.
ShowVisualBasic	This is a temporary property until we get a working language filter in the output like MSDN.
Title	This is the title displayed at the top of every page.

Overview

The XML Documenter is the simplest of the NDoc3 Documenters. It is primarily a development tool.

As part of the documentation compile process, NDoc3 merges the type information in the assemblies being documented with the code comment summary XML document that the /doc compiler option emits. The XML Documenter allows you to save this merged set of data for curiosity's sake or debugging purposes.

Used in conjunction with the **UseNDocXmlFile** setting, this is especially useful when you are working on your own documenters.

IMPORTANT: The XML produced by this documenter is an internal implementation detail and as such is **not** guaranteed to remain constant, or even remain back-backward compatible between released versions.

Settings

All documenters share a common set of documenter settings.

Setting	Description
OutputFile	This is the path and filename of the file where the merged documentation will be written. This can be absolute or relative from the .ndoc project file.

Overview

The JavaDoc documenter is used to make a set of HTML documentation similar in format and layout to the documentation created by Java's JavaDoc technology.

Due to lack of interest, this documenter is not under active development. If you are interested in updating this documenter, please contact one of the NDoc3 Admins.

Settings

All documenters share a common set of documenter settings.

Setting	Description
Title	The name of the JavaDoc project.
OutputDirectory	The folder where the root of the HTML set will be located. This can be absolute or relative from the .ndoc project file.

Overview

Settings

All documenters share a common set of documenter settings.

Setting	Description
OutputFile	This is the path and filename of the file where the merged documentation will be written. This can be absolute or relative from the .ndoc project file.

Overview

The LaTeX documenter can be used to create dvi or postscript documents.

This documenter requires that a LaTeX compiler be installed. You can download a free one from www.MiKTeX.org.

Settings

All documenters share a common set of documenter settings.

Setting	Description
OutputDirectory	The folder documentation will be created. This can be absolute or relative from the .ndoc project file.
LatexCompiler	Path to the LaTeX compiler executable (Set to empty if you do not have LaTeX installed).
TexFileBaseName	Name of the LaTeX document, excluding the file extension.

The following table indicates where top-level tags can be used.

Tag	Namespace	Class	Structure	Interface	Enum
<inheritdoc />					
<event>					
<example>		l	l	l	l
<exception>					
<exclude/>		l	l	l	l
<include>		l	l	l	l
<overloads>					
<param>					
<permission>					
<preliminary>		l	l	l	l
<remarks>		l	l	l	l
<returns>					
<seealso>		l	l	l	l
<summary>	l	l	l	l	l
<threadsafety>		l	l		
<value>					

The <c> tag is used to indicate that text within a description should be marked as code.

```
<c>text</c>
```

where:

text
    The text you would like to indicate as code.

Applies To

Can be used inline within any other markup.

Remarks

Use <code> to indicate multiple lines as code.

Example

[C#]
```
public class MyClass
{
   /// <summary>
   /// <c>MyMethod</c> is a method in the <c>MyClass
   /// </summary>
   public static void MyMethod(int Int1)
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags | <code> | Microsoft's definition

The <code> is used to indicate multiple lines as code.

```
<code [lang="language"][escaped="true"]>cont
```

where:

lang="*language*" [NDoc3 extension]
  Applies a filter for this language. (Optional)

escaped="true" [NDoc3 extension]
  Escapes all reserved characters within *content*. (Optional)

content
  The text you want marked as code.

Applies To

Can be used inline within any other markup.

Remarks

A language filter can be attached using the optional *lang* attribute. Standard languages are **Visual Basic**, **C#**, **C++** and **JScript**. Multiple languages can be specified as a comma separated list such as "Visual Basic, C#, C++".

The *escaped* attribute can be used to escape all reserved characters within the text.

Note: All content within xml comments must be well-formed!!!

Example

Note how, in the following comments, the xml text can be entered verbatim because the escaped="true" attribute has been applied.

[C#]
```
/// <summary>
/// Loads the XML.
/// </summary>
/// <example> The XML should have the following for
/// <code escaped="true">
///   <root>
///     <member name="name"/>
///   </root>
/// </code>
/// </example>
public void LoadXml(string xml)
{
   //do something here...
}
```

See Also

Tag Usage | NDoc3 Tags | <c> | Microsoft's definition

The <event> tag describes which events can be raised by the current method.

```
<event cref="member">description</event>
```

where:

cref = "*member*"
: A reference to an event that is available from the current compilation environment. The compiler checks that the given event exists and translates *member* to the canonical element name in the output XML. *member* must appear within double quotation marks (" ").

description
: A description.

Applies To

Method.

Remarks

Example

See Also

Tag Usage | NDoc3 Tags

The <example> tag describes an example of how to use a type or member.

```
<example>description</example>
```

where:

description
A description of the code sample.

Applies To

All types and members.

Remarks

Commonly, this tag is used with the <code> tag.

Example

[C#]

```
public class MyClass
{
   /// <summary>
   /// The GetZero method.
   /// </summary>
   /// <example> This sample shows how to call the Ge
   /// <code>
   ///   class MyClass
   ///   {
   ///      public static int Main()
   ///      {
   ///         return GetZero();
   ///      }
   ///   }
   /// </code>
   /// </example>
   public static int GetZero()
   {
      return 0;
```

```
        }
    }
```

See Also

Tag Usage | NDoc3 Tags | <code> | Microsoft's definition

The <exception> tag describes which exceptions can be thrown by a member.

```
<exception cref="member">description</except
```

where:

cref = "*member*"
: A reference to an exception that is available from the current compilation environment. The compiler checks that the given exception exists and translates *member* to the canonical element name in the output XML. *member* must appear within double quotation marks (" ").

description
: A description.

Applies To

Property, Method, Event, Operator, Type Conversion

Remarks

This tag is applied to a method definition.

Example

[C#]

```
using System;

/// comment for class
public class EClass : Exception
{
   // class definition ...
}

/// <exception cref="System.Exception">Thrown when..
class TestClass
{
   public static void Main()
```

```
    {
        try
        {
        }
        catch(EClass)
        {
        }
    }
}
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <exclude/> tag directs NDoc3 to exclude the current item from documentation.

```
<exclude/>
```

Applies To

All Types and Members.

Remarks

This tag takes precedence over any visibility options.

Examples

See Also

Tag Usage | NDoc3 Tags

The <include> tag lets you refer to comments in another file that describe the types and members in your source code.

```
<include file='filename' path='tagpath[@name
```

where:

filename
: The name of the file containing the documentation. The file name can be qualified with a path. Enclose *filename* in single quotation marks (' ').

tagpath
: The path of the tags in *filename* that leads to the tag *name*. Enclose the path in single quotation marks (' ').

name
: The name specifier in the tag that precedes the comments; *name* will have an *id*.

id
: The ID for the tag that precedes the comments. Enclose the ID in double quotation marks (" ").

Applies To

All Types and Members.

Remarks

This is an alternative to placing documentation comments directly in your source code file.

The <include> tag uses the XML XPath syntax. Refer to XPath documentation for ways to customize your <include> use.

Example

This is a multi-file example. The first file, which uses <include>, is listed below:

[C#]

```
/// <include file='xml_include_tag.doc' path='MyDocs
```

```
class Test
{
   public static void Main()
   {
   }
}

/// <include file='xml_include_tag.doc' path='MyDocs/
class Test2
{
   public void Test()
   {
   }
}
```

The second file, xml_include_tag.doc, contains the following documentation comments:

```
<MyDocs>

<MyMembers name="test">
<summary>
The summary for this type.
</summary>
</MyMembers>

<MyMembers name="test2">
<summary>
The summary for this other type.
</summary>
</MyMembers>

</MyDocs>
```

Compiler XML Output

```
<?xml version="1.0"?>
<doc>
    <assembly>
```

```
        <name>t2</name>
    </assembly>
    <members>
        <member name="T:Test">
            <summary>
The summary for this type.
</summary>
        </member>
        <member name="T:Test2">
            <summary>
The summary for this other type.
</summary>
        </member>
    </members>
</doc>
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <inheritdoc /> are used to indicate that documentaion should be inherited from the base type.

```
<inheritdoc />
```

Applies To

Any inherited member.

See Also

Tag Usage | NDoc3 Tags

The <list> tag describes a numbered or bulleted list, a definition list or a table.

```
<list type="bullet" | "number" | "table" | "
   <listheader>
      <term>term</term>
      <description>description</description>
   </listheader>
   <item>
      <term>term</term>
      <description>description</description>
   </item>
</list>
```

where:

term
: A term to define, which will be defined in *description*.

description
: Either an item in a bullet or numbered list or the definition of a *term*.

Remarks

The `<listheader>` block is used to define the heading row of a table. When defining a table, you only need to supply an entry for *term* in the heading.

Each item in the list is specified with an `<item>` block. When creating a definition list, you will need to specify both *term* and *description*. However, for a table, bulleted list, or numbered list, you only need to supply an entry for *description*.

A list or table can have as many `<item>` blocks as needed.

Examples

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <note> tag produces a formatted note block.

```
<note type="caution" | "inheritinfo" | "impl
   description
</note>
```

where:

description
Either an item in a bullet or numbered list or the definition of a *term*.

Applies To

Can be used inline within any other markup.

Remarks

Examples

See Also

Tag Usage | NDoc3 Tags

The <overloads> tag provides documentation that applies to all the overloads of a member.

```
Short Form
<overloads>
   summary_description
</overloads>
```

```
Expanded Form
<overloads>
   <summary>summary_description</summary>
   [<remarks>remarks_description</remarks>]
   [<example>examples_description</example>]
</overloads>
```

where:

term
: A term to define, which will be defined in *description*.

description
: Either an item in a bullet or numbered list or the definition of a *term*.

Applies To

Property, Method, Event, Operator.

Remarks

This tag only needs to be specified on the first overload.

The tag can have two forms:

- In the short form, it includes only one or more text blocks that are treated as the summary.
- In the expanded form, it can include one or more applicable section tags (summary, remarks and example).

Examples

See Also

Tag Usage | NDoc3 Tags | <summary> | <remarks> | <example>

The <para> tag is used to add structure to text.

```
<para [lang="language"]>content</para>
```

where:

lang="*language*" [NDoc3 extension]
Applies a filter for this language. (Optional)

content
The text of the paragraph.

Applies To

Can be used inline within any other markup.

Remarks

This tag is for use inside a tag, such as <summary>, <remarks>, or <returns>, and lets you add structure to the text.

A language filter can be attached using the optional *lang* attribute. Standard languages are **Visual Basic**, **C#**, **C++** and **JScript**. Multiple languages can be specified as a comma separated list such as "Visual Basic, C#, C++".

ExampleV

See <summary> for an example of using this tag.

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <param> tag describes one of the parameters for the method.

```
<param name='name'>description</param>
```

where:

name
    The name of a method parameter. Enclose the name in single quotation marks (' ').

description
    A description for the parameter.

Applies To

Property, Method, Event, Operator.

Remarks

Example

[C#]

```
/// text for class MyClass
public class MyClass
{
   /// <param name="Int1">Used to indicate status.</
   public static void MyMethod(int Int1)
   {
   }
   /// text for Main
   public static void Main ()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <paramref> tag indicates that a word is a parameter.

```
<paramref name="name"/>
```

where:

name
: The name of the parameter to refer to. Enclose the name in double quotation marks (" ").

Applies To

Can be used inline within any other markup.

Remarks

Example

[C#]

```
/// text for class MyClass
public class MyClass
{
   /// <remarks>MyMethod is a method in the MyClass c
   /// The <paramref name="Int1"/> parameter takes a
   /// </remarks>
   public static void MyMethod(int Int1)
   {
   }

   /// text for Main
   public static void Main ()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

}

The <permission> tag lets you document the security access of a member.

```
<permission cref="member">description</permi
```

where:

cref = "*member*"

A reference to a member or field that is available to be called from the current compilation environment. The compiler checks that the given code element exists and translates *member* to the canonical element name in the output XML. *member* must appear within double quotation marks (" ").

description

A description of the access to the member.

Applies To

All Members.

Remarks

The **System.Security.PermissionSet** lets you specify access to a member.

Example

[C#]

```
using System;
class TestClass
{
   /// <permission cref="System.Security.PermissionS
   public static void Test()
   {
   }
   public static void Main()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <preliminary> tags marks the documentation for the current item as preliminary.

```
<preliminary>[description]</preliminary>
```

where:

description
An optional textual message or warning that replaces the default preliminary warning.

Applies To

All Types and Members.

Remarks

If the empty form of this tag is used (<preliminary/>) the default message of "[This is preliminary documentation and subject to change.]" will be included in the generated help topic.

If a value is supplied for the content of this tag, that value will appear in the help topic, replacing the default message. You can format the message text using the para and list tags, but this is not required.

Entire help titles can marked preliminary using the Preliminary documenter setting.

Examples

[C#]

```
// The class summary will get the default message
/// <preliminary/>
public class MyClass
{
   /// <preliminary>
   /// <para>This method is just for testing right no
   /// </preliminary>
   public void Dump ()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags

The <remarks> tag describes additional information about a type or member..

```
<remarks>description</remarks>
```

where:

description
Additional information regarding the type or member.

Applies To

All Types and Members.

Remarks

The <remarks> tag is used to add additional information about a type or member, supplementing the information specified with <summary>.

Example

[C#]

```
/// <summary>
/// You may have some primary information about this
/// </summary>
/// <remarks>
/// You may have some additional information about tl
/// </remarks>
public class MyClass
{
   /// text for Main
   public static void Main ()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <returns> tag describes the return value of a method.

```
<returns>description</returns>
```

where:

description
: A description of the return value.

Applies To

Method.

Remarks

Example

[C#]
```
/// text for class MyClass
public class MyClass
{
   /// <returns>Returns zero.</returns>
   public static int GetZero()
   {
      return 0;
   }

   /// text for Main
   public static void Main ()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags | Microsoft's definition

The <see> tag specifies a link to other documentation from within text.

```
<see cref="member">[label]</see>
OR
<see href="URL">[label]</see>
OR
<see langword="null | sealed
| static | abstract | virtual | true | false
```

where:

label
: text to display as the link

cref = "*member*"
: A reference to a member or field that is available to be called from the current compilation environment. The compiler checks that the given code element exists and passes *member* to the element name in the output XML. *member* must appear within double quotation marks (" ").

href = "*URL*" [NDoc3 extension]
: A reference to an external resource at the address given by the URL.

langword [NDoc3 extension]
: A common .NET language keyword. These keywords are highlighted, and, in some cases, expanded into descriptive phrases (see remarks for further details). Note that although the syntax above only shows the common keywords, any word specified will be highlighted.

Applies To

Can be used inline within any other markup.

Remarks

Use <seealso> to indicate text that you might want to appear in a **See Also** section.

Note: As of release 1.3, the MSDN and VS.NET documenters will only create a link on the first occurrence of each unique cref specified within a documentation section; further <see> tags will just be highlighted. This improves the readability of the documentation.

langword expansions

keyword	expansion
null	null reference (**Nothing** in Visual Basic)
sealed	sealed (**NotInheritable** in Visual Basic)
static	static (**Shared** in Visual Basic)
abstract	abstract (**MustInherit** in Visual Basic)
virtual	virtual (**CanOverride** in Visual Basic)

Example

See <summary> for an example of using <see>.

See Also

Tag Usage | NDoc3 Tags | <seealso> | Microsoft's definition

The <seealso> tag is used to specify links appearing in a **See Also** section.

```
<seealso cref="member">[label]</seealso>
OR
<seealso href="URL">[label]</seealso>
```

where:

label
: text to display as the link

cref = "*member*"
: A reference to a member or field that is available to be called from the current compilation environment. The compiler checks that the given code element exists and passes *member* to the element name in the output XML. *member* must appear within double quotation marks (" ").

href = "*URL*" [NDoc3 extension]
: A reference to a member or field that is available to be called from the current compilation environment. The compiler checks that the given code element exists and passes *member* to the element name in the output XML. *member* must appear within double quotation marks (" ").

Applies To

All Types and Members.

Remarks

Use <see> to specify an in-line link within text.

This is a 'top-level' tag. Do **not** nest this within other tags.

> **Note:** Do not apply this tag to enumeration members. In the MSDN-style documenters, enumeration members are listed in a table on the enumeration type topic rather than individual topics, and any <seealso> tags will be ignored.

Example

See <summary> for an example of using <seealso>.

See Also

Tag Usage | NDoc3 Tags | <see> | Microsoft's definition

The <summary> tag is used to provide a short description of a type or member.

```
<summary>description</summary>
```

where:

description
A summary of the object.

Applies To

All Types and Members.

Remarks

This tag should be treated as mandatory for all publicly accesible types and members. It is the primary description used by IntelliSense and the Object Browser in VisualStudio, and most other development tools.

Use <remarks> to add supplemental information to a type or member description.

Example

[C#]

```
/// text for class MyClass
public class MyClass
{
   /// <summary>MyMethod is a method in the MyClass 
   /// <para>Here's how you could make a second para
   /// <seealso cref="MyClass.Main"/>
   /// </summary>
   public static void MyMethod(int Int1)
   {
   }

   /// text for Main
   public static void Main ()
```

```
        {
        }
    }
```

See Also

Tag Usage | NDoc3 Tags | <remarks> | Microsoft's definition

The <threadsafety> tag is used to describe how a class or structure behaves in multi-threaded scenarios.

```
<threadsafety static="true|false" instance="
```

where:

static="true|false"
: indicates whether static member of this class are safe for multi-threaded operations.

instance="true|false"
: indicates whether members of instances of this type are safe for multi-threaded operations.

Applies To

Class, Structure.

Remarks

Examples

[C#]

```
/// <threadsafety static="true" instance="false"/>
public class MyClass
{
   /// not safe across threads
   public void InstanceMethod()
   {
   }

   /// safe across threads
   public static void StaticMethod()
   {
   }
}
```

See Also

Tag Usage | NDoc3 Tags

The <value> tag is used to describe the value of a property.

```
<value>property-description</value>
```

where:

property-description
  A description for the property.

Applies To

Property.

Remarks

By convention, properties should always have a <value> tag.

For read-only properties, the text in the <value> tag will often be substantially the same as that in the <summary> tag.

Example

[C#]
```
/// text for class MyClass
public class MyClass
{
   /// <summary>MyProperty is a property in the MyCl
   /// <value>A string containing the text "MyProper
   public string MyProperty()
   {
       get
       {
           return "MyProperty String";
       }
   }

}
```

See Also

Tag Usage | NDoc3 Tags | <summary> | Microsoft's definition

NDoc3 Extensibility

Both the MSDN and VS.NET documenters support an extensibility model that allows you to define your own tags and control where and how they appear in the documentation. NDoc3 relies heavily on XSLT to generate documentation and the extensibility model is based on XSLT as well.

1. The first step is to add your custom tag to the code comments in your C# files:

   [C#]

   ```
   /// <myTag>This is a custom tag</myTag>
   /// <summary>
   ///   When processed by the VS.NET or MSDN
   ///   using the stylesheet "extend-ndoc.xs
   ///   property will result in end-user def
   ///   final help output topics
   /// </summary>
   /// <remarks>This is a test of an inline <
   public class ABunchOfCustomTags
   {
   }
   ```

   When the compiler generates the /doc summary file, it will include your custom tag in the XML.

2. Next create an XSLT file with templates that control where and how your tag will be displayed:

   ```
   <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:MSHelp="http://msdn.microsoft.com/mshelp">

       <xsl:template match="node()" mode="xml-data-island"
   priority="1">

           <MSHelp:Attr Name="TargetOS"
   Value="Windows"/>

       </xsl:template>
   ```

```
    <xsl:template match="ndoc" mode="header-section">
        <style type="text/css">
            .green
            {
                color:green;
            }
        </style>
    </xsl:template>        <xsl:template match="myTag"
mode="seealso-section">
        <h1 class="green">
            <xsl:apply-templates select="./node()"
mode="slashdoc"/>
        </h1>
    </xsl:template>
    <xsl:template match="null" mode="slashdoc">
        <xsl:text> null reference (Nothing in Visual Basic)
</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

This stylesheet must be valid XSLT markup, and must also include the XSLT namespace.

The templates in this stylesheet need to be written to match the names of your tags. The **match** attribute is an XPath query that defines the context at which your template will execute. The most common usage is to simply match the name of your custom tag, but it is also possible to change the behavior of your tag based on what sort of code artifact it is associated with.

> **Note:** If you want NDoc3 to perform standard processing of tags *within* your tags (for example,

expanding in-line tags such as <see>) you must include the following within your processing

```
<xsl:apply-templates select="." mode="slashdoc"/>
```

The **header-section** template allows you to specify additional content in the **<head>** of the generated html. This allows stylesheet authors to define custom css styles or override NDoc3's default styles.

The **mode** attribute is used to specify where in the documentation your tag will be displayed. There are two types of extensibility tags that can be defined:

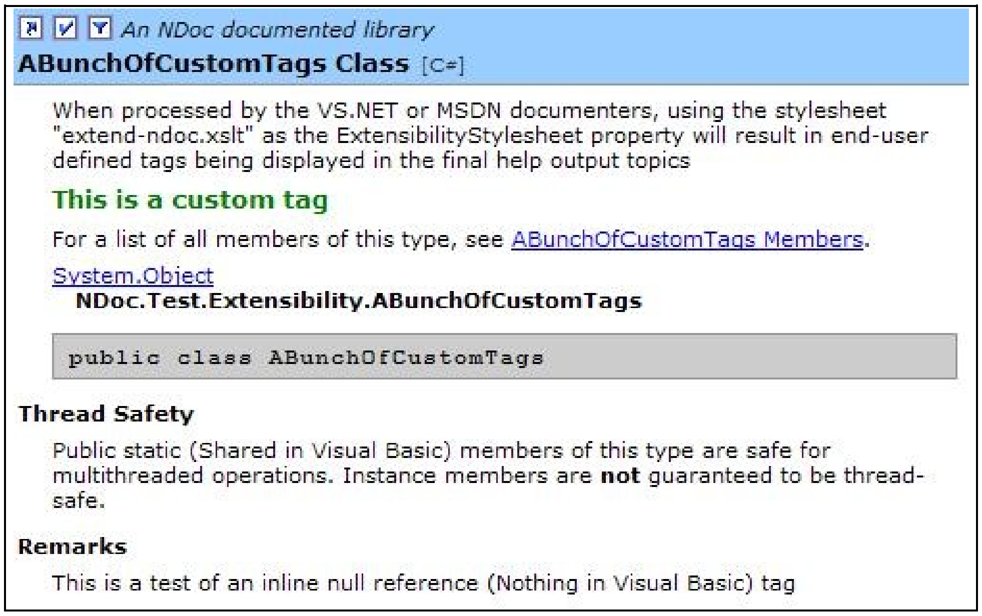
1. **Section Tags -** These are block tags that will be rendered in a particular section of the documentation. For a section tag, the mode of the template needs to correspond to a predefined list of section names. Refer to the section topic for a complete list of sections and their descriptions.
2. **Inline Tags -** These are tags that appear inline with text and other tags within a comment. To define an inline tag, the mode of its template should be set to **"slashdoc"**. These templates will then be executed whenever the tag is encountered within a block of text.

If a template matches any of the generic XPath selections: **node()**, **text()**, ***, or **@*** you will need to supply a **priority** attribute on the template with a value greater than 0.5. This forces XSLT to use your templates rather than the default templates that NDoc3 includes for the various extensibility sections.

Remember: XSLT is case sensitive; so both the **match** pattern and the **mode** names must be the correct case or your templates will be ignored.

3. Next, set the ExtensibilityStylesheet property of your MSDN or VS.NET project to the path to your stylesheet. When NDoc3 builds the documentation, it will include this stylesheet and display your tags according to the rules you specify in the XSLT templates.

The resulting overview page for this topic will look like:



An NDoc documented library

ABunchOfCustomTags Class [C#]

When processed by the VS.NET or MSDN documenters, using the stylesheet "extend-ndoc.xslt" as the ExtensibilityStylesheet property will result in end-user defined tags being displayed in the final help output topics

This is a custom tag

For a list of all members of this type, see ABunchOfCustomTags Members.

System.Object
   NDoc.Test.Extensibility.ABunchOfCustomTags

```
public class ABunchOfCustomTags
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

Remarks

This is a test of an inline null reference (Nothing in Visual Basic) tag

Example Extensibility Output

See Also

Extensibility Sections

Extensibility Sections

Below are the names of the documentation sections where custom tags can be displayed. These names are used as the **mode** attribute in extensibility stylesheet templates.

Section Name	Description
header-section	Templates in the header-section will be output within the html **<head>** tag.
preliminary-section	This is the notification that it output for items marked with the <preliminary> tag or projects compiled with the Preliminary setting equal to **true**.
summary-section	The section corresponding to the <summary> tag.
thread-safety-section	The section where thread safety information is displayed.
syntax-section	The section where item syntax is displayed.
value-section	The section corresponding to the <value> tag.
parameter-section	The section corresponding to the <param> tag.
returnvalue-section	The section corresponding to the <returns> tag.
implements-section	
remarks-section	The section corresponding to the <remarks> tag.
after-remarks-section	A section ***after*** the section corresponding to the <remarks> tag. **Note:** This extension section will be

	processed regardless of whether <remarks> exist or not...
obsolete-section	The section denoting that an item is decorated with the **ObsoleteAttribute**.
events-section	The section corresponding to the <event> tag.
exceptions-section	The section corresponding to the <exception> tag.
example-section	The section corresponding to the <example> tag.
member-requirements-section	
type-requirements-section	
seealso-section	The list of See Also links at the bottom of a topic.
enumeration-members-section	The section where an enumeration's members are listed.
footer-row	The footer row at the bottom of each topic.
title-row	The title row at the top of each topic.
overloads-remarks-section	
overloads-example-section	
overloads-summary-section	
xml-data-island	The embedded XML data island used by Html Help 2 to include meta-data about a topic. *Applies only to the VS.NET 2003*

	documenter.

See Also

NDoc3 Extensibility

Contribute to the Success of NDoc3

There are many ways you can become involved in the NDoc3 development effort.

- Participating in the ndoc3-users email list is the easiest way to share your tips and tricks for using NDoc3 with other users.
- If you find bugs please use the NDoc3 bug tracker database, and if you have suggestions for new features please use the NDoc3 feature request tracker database so that we can keep current with our user's needs.
- If you have some time to spare, become part of the NDoc3 developer team. There are plenty of new features we are working on implementing, and there are always bugs to fix, and the more people we have working on NDoc3, the better it becomes. To join the effort, contact one of the project admins via the NDoc3 SourceForge foundry, and we will get you set up.

Thanks for using NDoc3.

There are numerous place online to get NDoc support.

The best place to start is often the support archive at the NDoc SourceForge foundry where you will often turn up the answer you are looking for.

The NDoc3 Users's Mailing List

You can also search the the archives or send a message to the ndoc3-users mailing list.

The NDoc3 Tracker Database

If you still can not find the answer you are looking for, NDoc3 maintains a Tracker database at SourceForge.net.

- To submit a support request, visit the NDoc support page at SourceForge.net.
- To submit a bug report, visit the NDoc bug tracker page at SourceForge.net.
- If there is a feature you'd like to suggest, let us know at the Request For Enhancement page at SourceForge.net.

Articles About NDoc

There are also a number of good articles online, although they may be somewhat out-of-date as most were written some time ago...

- [Documenting C# with XML comments], by Ollie Cornes
- [Using NDoc: Adding World-Class Documentation to Your .NET Components], by Shawn Van Ness
- [Fixing NDoc to emit links for Everett's MSDN docs], by Shawn Van Ness
- [Integrate NDoc HTML Help 2 in Visual Studio.NET], by Fons Sonnemans
- [Creating class documentation with NDoc], by Rick Harris

- [Integrating Help into Visual Studio.NET], by Sune Trudslev