Analysis Services Programming
Programming Analysis Services Applications

Microsoft® SQL Server™ 2000 Analysis Services provides support for you to create and integrate custom applications that enhance your online analytical processing (OLAP) and data mining installation.

Analysis Services includes the Analysis server and PivotTable® Service. The Analysis server manages and stores multidimensional information and serves client application requests for OLAP data. PivotTable Service is an OLE DB for OLAP provider that connects client applications to the Analysis server and manages offline cubes. A repository of meta data contains definitions of OLAP data objects such as cubes and their elements.

An object model, Decision Support Objects (DSO), provides support for the Analysis Manager user interface and for custom applications that manage OLAP meta data and control the server. An interface, IOLapAddIn, enables your applications to extend and interact with the user interface. PivotTable Service provides access to OLAP data from the server and the ability to create local cubes.

You can create applications that:

- Manage the Analysis server and create and maintain OLAP and data mining objects such as cubes, dimensions, security roles, and data mining models.

- Extend the user interface by adding new objects to the object tree pane and by adding and responding to new menu choices.

- Connect to the Analysis server, query data in cubes, and create local cubes.

- Combine any or all of these functions.

This section contains the following topics.
<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analysis Services Architecture</strong></td>
<td>Information about the architecture of Analysis Services and its components</td>
</tr>
<tr>
<td><strong>Analysis Services Component Tools</strong></td>
<td>Information about the tools in Analysis Services that you can use to create administrative support applications and client data access applications</td>
</tr>
<tr>
<td><strong>SQL in Analysis Services</strong></td>
<td>Details of the implementation of SQL in Analysis Services</td>
</tr>
<tr>
<td><strong>Decision Support Objects</strong></td>
<td>Information about the Analysis Services server object model, the component tool for managing OLAP, and data mining objects</td>
</tr>
<tr>
<td><strong>Add-ins</strong></td>
<td>Information about the <strong>IOlapAddIn</strong> interface you can use in your applications to interact with the Analysis Manager user interface</td>
</tr>
<tr>
<td><strong>PivotTable Service</strong></td>
<td>Information about the client application service you can use with applications that query OLAP and data mining data and create local cubes and mining models</td>
</tr>
<tr>
<td><strong>Analysis Services Programming Samples</strong></td>
<td>Information about the samples that illustrate development of applications for Analysis Services</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Analysis Services Architecture

Microsoft® SQL Server™ 2000 Analysis Services includes the Analysis server and PivotTable® Service. The Analysis server creates and manages multidimensional data cubes for online analytical processing (OLAP) and provides multidimensional data to PivotTable Service, which in turn provides this data to clients through Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD) and OLE DB for OLAP provider services.

The server stores cube meta data (cube definition specifications) in a repository. Completed cubes can be stored in a variety of storage modes: as multidimensional database files (MOLAP), as tables in a relational database (ROLAP), or as a hybrid of multidimensional database files and relational tables (HOLAP).

Source data for multidimensional cubes resides in relational databases where the data has been transformed into a star or snowflake schema typically used in OLAP data warehouse systems. Analysis Services can work with many relational databases that support connections using ODBC or OLE DB. When used as part of SQL Server 2000, Analysis Services offers enhanced security and other capabilities. The Data Transformation Services (DTS) feature of SQL Server 2000 provides a means to manage the data warehouse from which Analysis Services creates cubes.

Control of the server is accomplished through the Analysis Manager user interface, or through custom applications developed using the Decision Support Objects (DSO) object model. DSO controls the creation and management of cubes by the server, and manages the cube meta data in the repository. The object model is used by the Analysis Manager program that provides the user interface through a snap-in to Microsoft Management Console (MMC). The DSO object model can be used by applications written in Microsoft Visual Basic® to provide custom programmatic control of the server. You can also develop custom applications to interact with the Analysis Manager user interface.

The following diagram illustrates the elements and functions of the Analysis server and its use of PivotTable Service to provide multidimensional data to
client consumer applications. The Analysis Manager user interface uses PivotTable Service to obtain multidimensional data from the server for browsing by the server administrator. For more information about client applications, see **PivotTable Service**.

**OLAP Services System Architecture (with PivotTable Service)**

- MMC
- OLAP Add-In Manager (MMC Snap-In)
- OLAP Manager
- Object Model (Decision Support Objects)
- OLAP server
- PivotTable Service
- OLE DB with OLAP Extensions
- ADO with OLAP Extensions
- Client Application

- Custom Add-In
- Custom Application
- Metadata

- ROLAP
- Source data
- ROLAP data
- HOLAP
- MOLAP

- Relational Database
- Cube
- Disk Storage
Analysis Services Programming
Analysis Services Component Tools

Microsoft® SQL Server™ 2000 Analysis Services provides three component tools you can use with your custom applications. You can use any combination of these tools to create applications that help you manage your Analysis Services installation and provide OLAP data to your end users.

DSO

Decision Support Objects (DSO) exposes the object model for the Analysis server. Your applications can use DSO to control and automate functionality on the server. You can also create and maintain OLAP objects such as cubes, dimensions, and roles.

The DSO object model consists of interfaces, objects, collections, methods, and properties. You create and manipulate DSO objects to manage the meta data for OLAP data. This meta data is stored in a repository in a relational database and is accessed by the Analysis server and DSO.

DSO uses hierarchically arranged groups of objects to define basic elements of OLAP data. These basic elements include databases, dimensions, cubes, partitions, aggregations, and virtual cubes. DSO addresses these basic elements in a hierarchical structure where elements contain other elements in a tree, with the server at the root of the tree. You can observe the hierarchy of this structure in the Analysis Manager tree pane.

Your application can use DSO in combination with other component tools of Analysis Services to enhance, augment, and automate your Analysis Services installation.

You can develop your DSO applications in Microsoft Visual Basic® and other languages that support the Component Object Model (COM). The DSO object model was developed in Visual Basic and is easiest to use with that language.

For more information, see Decision Support Objects.

Add-ins Interface and Objects

You can create applications that interact with and enhance the Analysis Services
user interface. Analysis Manager can call various routines in your application in response to user activity in the user interface. Your add-in can add nodes to the structure in the tree pane as the user selects or expands a node; it can also augment node menus with items that cause your program to be called when those items are selected. Multiple custom add-ins can be registered and operating at the same time.

The Analysis Manager user interface is implemented as an add-in and is called by the Analysis Services Add-in Manager in the same way that your custom add-in will be called.

Your add-in can use other Analysis Services component tools to enhance, augment, and automate your Analysis Services installation.

For more information about creating and registering your add-ins, see Add-ins.

**PivotTable Service**

PivotTable® Service, which is included with Analysis Services, is an OLE DB provider that supports the optional OLE DB for OLAP extensions. It functions as a connection interface with cache management functionality to Analysis Services to support client application access to OLAP data.

PivotTable Service is also an in-process desktop Analysis server designed to provide offline data analysis, cube building, and functionality to manipulate data. PivotTable Service stores data locally on the client for offline analysis and offers connectivity to the multidimensional data managed by Analysis Services, other OLE DB-compliant providers, and to non-OLAP relational data sources.

PivotTable Service supports OLE DB Multidimensional Expressions (MDX) as its native consumer interface, and a subset of SQL. PivotTable Service also extends the language defined in OLE DB by adding data definition language (DDL) and data manipulation language (DML) statements to define the structure of local multidimensional data cubes.

Your applications that use PivotTable Service to communicate with the Analysis server or to manage local cubes can use OLE DB interfaces for C++ or Microsoft ActiveX® Data Objects (ADO) and ADO (Multidimensional) (ADO MD) with any COM automation language, including Visual Basic.

You can use PivotTable Service with OLE DB-compliant data sources or ODBC-
compliant data sources. PivotTable Service supports the following relational database products:

- SQL Server version 7.0 and earlier and SQL Server 2000
- Microsoft Access 97 and later
- Oracle versions 7.3 and 8.0

See Also

Decision Support Objects
Add-ins
PivotTable Service
Analysis Services Programming
SQL in Analysis Services

Microsoft® SQL Server™ 2000 Analysis Services is both a multidimensional data provider and a tabular data provider. Therefore, executing a query returns either a multidimensional dataset or a flattened rowset, depending on the query language used. Analysis Services can interpret and process queries in both SQL and Multidimensional Expressions (MDX).

In addition to querying, you can use certain data definition language (DDL) statements to create local cubes, calculated members, user-defined sets, and cache. For more information, see PivotTable Service.

For more information about OLE DB, OLE DB for OLAP, and the MDX syntax as defined by OLE DB for OLAP, see the OLE DB documentation. For more information about Microsoft ActiveX® Data Objects (ADO) and ADO (Multidimensional) (ADO MD), see the ADO documentation.

See Also

MDX
SQL
Analysis Services Programming
Executing an SQL Query

SQL queries can be passed to Microsoft® SQL Server™ 2000 Analysis Services using either of the following data connectivity tools:

- Microsoft OLE DB, including OLE DB for Online Analytical Processing (OLE DB for OLAP)

- Microsoft ActiveX® Data Objects (ADO) and ADO (Multidimensional) (ADO MD).

OLE DB for OLAP extends OLE DB to include objects specific to multidimensional data. ADO MD extends ADO in the same way.

All three data connectivity tools provide objects that support SQL queries; OLE DB provides the Command object, ADO provides the Recordset object, and ADO MD provides the Cellset object. Each data connectivity tool, however, provides support in a different manner.

**OLE DB**

The OLE DB for OLAP Command object supports text commands in the Multidimensional Expressions (MDX) syntax by using the OLE DB ICommandText interface. The ICommandText::SetCommandText method specifies the command and the ICommand::Execute method processes the command. For commands that may be used several times, the ICommandPrepare::Prepare method prepares the command.

OLE DB for OLAP defines a language dialect identifier GUID (MDGUID_MDX) that denotes the MDX syntax. This GUID is used in ICommandText::SetCommandText to identify to the provider that the query language being used is the MDX syntax.

OLE DB also defines another language dialect identifier, DBGUID_DEFAULT, which denotes the default language dialect. The following rules apply to the use of this identifier (from the OLE DB for OLAP specification):

- If the provider is both a multidimensional data provider and a tabular
data provider, it must interpret the command string as SQL when DBGUID_DEFAULT is passed as the dialect identifier.

- If the provider is a multidimensional data provider, it should interpret the command string as MDX when DBGUID_DEFAULT is passed as the dialect identifier.

**IMPORTANT** Analysis Services does not simultaneously conform to both of these semantics. Instead, when DBGUID_DEFAULT is passed, Analysis Services analyzes the query string text and attempts to determine which dialect is being used.

If the SQL dialect is used, a flattened rowset is returned. If the MDX dialect is used, Analysis Services analyzes the rowset interface identifier (IID) to determine whether a multidimensional dataset or a flattened rowset is returned.

**ADO**

The **Open** method of the ADO **Recordset** object retrieves the results of an SQL query.

**Syntax**

```
object.Open [Source], [ActiveConnection], CursorType As CursorTypeEnum = adOpenUnspecified], [LockType As LockTypeEnum = adLockUnspecified], [Options As Long = -1]
```

**Parameters**

**object**

An instance of the ADO **Recordset** object.

**Source**

(Optional) A Variant that evaluates to a valid ADO **Command** object, valid SQL query, table name, stored procedure call, URL, or the name of a file containing a persistently stored ADO **Recordset** object. This argument corresponds to the **Source** property.
ActiveConnection

(Optional) A Variant that evaluates to a valid ADO Connection object variable name or a string that contains a connection definition. The ActiveConnection argument specifies the connection in which to open the Cellset object. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters. The ActiveConnection argument corresponds to the ActiveConnection property.

CursorType

(Optional) A CursorTypeEnum value that determines the type of cursor that the provider should use when opening the ADO Recordset object.

LockType

(Optional) A LockTypeEnum value that determines what type of locking (concurrency) the provider should use when opening the ADO Recordset object.

Options

(Optional) A value that indicates how the provider should evaluate the Source argument if it represents something other than an ADO Command object, or if the ADO Recordset object should be restored from a file where it was previously saved. This value may be set to a value supplied from either the ADO CommandTypeEnum or ExecuteOptionEnum enumerations.

Remarks

The Open method fails if either the Source or Active Connection parameters are missing or their corresponding properties are not set.

The default value for the CursorType property is adOpenUnspecified.

The default value for the LockType is adLockUnspecified.

ADO MD

The Open method of the ADO MD Cellset object retrieves the results of a multidimensional query.
Syntax

`object.Open [DataSource], [ActiveConnection]`

Parameters

`object`

An instance of the ADO MD `Cellset` object.

`DataSource`

(Optional) A Variant that evaluates to a valid multidimensional query, such as an MDX query. The `DataSource` argument corresponds to the `Source` property.

`ActiveConnection`

(Optional) A Variant that evaluates to a string specifying either a valid ADO `Connection` object variable name or a definition for a connection. The `ActiveConnection` argument specifies the connection in which to open the `Cellset` object. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters. The `ActiveConnection` argument corresponds to the `ActiveConnection` property.

Remarks

The `Open` method generates an error if either of its parameters is omitted and its corresponding property value has not been set prior to an attempt to open the `Cellset`. 
Analysis Services Programming
Exposed Schema

Some of the OLE DB schema rowsets that are common to tabular data providers are interpreted differently when they are used with Microsoft® SQL Server™ 2000 Analysis Services. The interpretation that Analysis Services ascribes to each one is listed in the following table.

<table>
<thead>
<tr>
<th>Rowset</th>
<th>Meaning in Analysis Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOGS</td>
<td>The list of catalogs (databases).</td>
</tr>
<tr>
<td>TABLES</td>
<td>The list of cubes. For SQL queries, each cube can function as a table.</td>
</tr>
<tr>
<td>COLUMNS</td>
<td>The list of levels and measures for each cube. Column names are of the form <code>dimension_name:level_name</code> and <code>dimension_name:measure_name</code>. For SQL queries, each level and each measure can function as an SQL column.</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Supported SQL SELECT Syntax

Microsoft® SQL Server™ 2000 Analysis Services supports the following subset of the SQL SELECT command syntax.

Syntax

```
SELECT [<options_clause>] <select_list> FROM <from_clause> [WHERE <where_clause>] [GROUP BY <groupby_clause>]
```

```
<options_clause> ::= <empty_clause> | DISTINCT
```

```
<select_list> ::= <scalar_exp_commalist> | ASTERISK
```

```
<scalar_exp_commalist> ::= <scalar_expression> [, <scalar_expression> [, <scalar_expression> [...]]]
```

```
<scalar_expression> ::= <column_ref> | <aggregate>
```

```
| ( <column_ref> ) AS IDENTIFIER
```

```
<aggregate> ::= <aggregate_func> (<column_ref>)
```

**Note** In the preceding line, <column_ref> must be a measure name. <aggregate_func> must agree with the Aggregate Function property of the measure.

```
<aggregatel_func> ::= COUNT | MIN | MAX | SUM
```

**Note** <select_list> can contain references only to levels or measures. If measures are specified, you must also specify <aggregate>.

```
<from_clause> ::= cube_name
```

```
<where_clause> ::= empty_clause | <search_condition>
```

```
<groupby_clause> ::= <column_ref_commalist>
```

```
<column_ref_commalist> ::= ( <column_ref> ) [, ( <column_ref> ) [, ( <column_ref> ) [...]]]
```

<search_condition> ::= <empty_clause>
| <search_condition> AND <search_condition>
| <search_condition> OR <search_condition>
| (<search_condition>)
| <comparison_predicate>

<comparison_predicate> ::= (<column_ref>) = VALUE | VALUE = (<column_ref>)

**Note** In the preceding line, <column_ref> must be a valid level name.

**Remarks**

There are some limitations of the SQL SELECT statement in Analysis Services.

- You cannot use DISTINCT or GROUP BY if <select_list> contains members.

- Using the DISTINCT option with levels in <select_list> can cause the following problems:

  - If a parent level has more than one member, and not all parents are listed, Analysis Services may return duplicate rows. Be sure to explicitly include all parents.

  - If the root level for a dimension contains more than one member, Analysis Services may return duplicate rows. Include all dimensions with root levels having more than one member as columns.

  - DISTINCT and GROUP BY may return multiple rows if the server contains more than one segment.

- You cannot use SQL syntax to query a virtual dimension that was created in SQL Server version 7.0 OLAP Services. You must use Multidimensional Expressions (MDX) to query this type of virtual
dimension.
Analysis Services Programming
Passing Queries from SQL Server to a Linked Analysis Server

The linked server feature of Microsoft® SQL Server™ 2000 allows you to execute queries against OLE DB data sources that are hosted on remote computers. There are no special requirements for using this feature with SQL Server 2000 Analysis Services, but there are some important points to note when configuring the two systems.

Security

Analysis Services uses security that is integrated with Microsoft Windows NT® 4.0 and Windows® 2000 to identify user accounts. It cannot be configured to recognize accounts created for use with SQL Server authentication. However, the MSSQLServer service on the linked server can be configured to log on using an account that has sufficient permissions to access Analysis Services.

Using the OPENQUERY Function

For best results with pass-through queries from SQL Server to Analysis Services, use the Transact-SQL function OPENQUERY to execute SQL commands between servers. OPENQUERY sends the commands of the query directly to the Analysis server, which then returns flattened rowsets (as described in the OLE DB documentation) that contain the requested data. The syntax of the passed query is not limited to the abbreviated SQL SELECT options supported by Analysis Services, but can also include Multidimensional Expressions (MDX) commands.

Executing SQL Queries using the OPENQUERY Function

The OPENQUERY function accepts two parameters: the name of the linked server and the text of the query to pass.

Examples
A. Returning Total Sales Grouped by Customer Gender

This query returns the total sales grouped by customer gender:

```
select * from openquery( LINKED_OLAP, 'select [Customer Gender:Gender], sum([measures:unit sales]) from sales group by [Customer Gender:Gender]' )
```

B. Returning Total Sales Grouped by Gender and Education

This query returns the total sales grouped by customer gender and education level:

```
select * from openquery( LINKED_OLAP, 'select [Customer Education Level:Education], [Customer Gender:Gender], sum([measures:unit sales]) from sales group by [Customer Education Level:Education], [Customer Gender:Gender]' )
```

Because there is a limitation in Analysis Services that causes GROUP BY and DISTINCT queries to produce multiple rows that satisfy the grouping and/or distinct functions (instead of just one), it may be necessary to copy the rows to a temporary table and reduce them further.

The following examples show how SQL Server can be used to merge the results of a query from Analysis Services.

C. Performing a DISTINCT Operation

In this query, SQL Server performs an additional DISTINCT operation on the data retrieved by Analysis Services:

```
select distinct * from openquery( LINKED_OLAP, 'select distinct [Customer Location:Country], [Customer Location:State Province], [Customer Location:City] from sales' )
```

D. Using the ORDER BY Command
In this query, the ORDER BY command sorts the values retrieved by Analysis Services:

```sql
select distinct * from openquery( LINKED OLAP,
'select distinct [Customer Location:Country!name],
[Customer Location:State Province!name],
[Customer Location:City!name]
from sales' )
order by
[Customer Location:Country!name],
[Customer Location:State Province!name],
[Customer Location:City!name]
```

### E. Guaranteeing the Correctness of the GROUP BY Command

In this query, SQL Server guarantees the correctness of the GROUP BY command (because Analysis Services might not coalesce all of the returned rows):

```sql
select [Customer Location:Country!name], [Customer Gender:Gender!name],
sum([measures:unit sales])
from openquery( LINKED OLAP,
'select [Customer Location:Country!name], [Customer Gender:Gender]
sum([measures:unit sales]) from sales
group by [Customer Location:Country!name],
[Customer Gender:Gender!name]' )
group by [Customer Location:Country!name], [Customer Gender:Gender]
```

### F. Using a WHERE Clause

This query combines all elements of the preceding examples and includes a WHERE clause:

```sql
select
[Customer Location:Country!name],
[Customer Gender:Gender!name],
```
sum([measures:unit sales])
from openquery( LINKED_OLAP,
'select
[Customer Location:Country!name],
[Customer Gender:Gender!name],
[Product:Product Family!name],
[Product:Product Department!name],
[Product:Product Category!name],
[Product:Product Subcategory!name],
sum([measures:unit sales])
from sales
where
[Product:Product Family!name] = "Food" and
[Product:Product Department!name] = "Baked Goods"
)
group by
[Customer Location:Country!name],
[Customer Gender:Gender!name],
[Product:Product Family!name],
[Product:Product Department!name],
[Product:Product Category!name],
[Product:Product Subcategory!name] ')
group by
[Customer Location:Country!name],
[Customer Gender:Gender!name],
[Product:Product Family!name],
[Product:Product Department!name],
[Product:Product Category!name],
[Product:Product Subcategory!name]
order by
[Customer Location:Country!name],
[Customer Gender:Gender!name],
[Product:Product Family!name],
[Product:Product Department!name],
[Product:Product Category!name],
[Product:Product Subcategory!name]

**Executing MDX Queries Using the OPENQUERY Function**

Because the OPENQUERY function causes SQL Server to pass the text of the query directly to Analysis Services, you can use MDX syntax for complex multidimensional queries. The result sets from MDX queries appear as flattened rowsets. For more information about how a multidimensional result set is mapped to a tabular rowset in OLE DB for OLAP, see the OLE DB documentation.

**Example**

The following example submits an MDX query using the OPENQUERY function.

```sql
select * from openquery
( LINKED_OLAP, 'select { measures.[unit sales] } on columns,
  non empty nest( nest( [customer location].[country].members,
  [gender].members ), [product category].[bread].children ) on rows
  from sales ')
```

**Avoiding Four-Part Naming**

It is possible to access the data of a cube directly from SQL Server using queries with four-part naming. (The four parts are linked-server-name, catalog, schema, and table.) However, this option is not recommended because SQL Server attempts to copy the contents of the entire fact table and then perform the calculations for aggregating the data itself, substantially increasing the query response time.
Analysis Services Programming
Adding a Linked Server

There are two ways to add a linked server. You can create one by using SQL Server Enterprise Manager interface or by issuing SQL commands. If you use SQL Server Enterprise Manager, you can configure a wider set of options. For more information about adding and using linked servers, see Configuring Linked Servers and Establishing Security for Linked Servers.
Analysis Services Programming
Decision Support Objects

The Decision Support Objects (DSO) library of Microsoft® SQL Server™ 2000 Analysis Services provides a robust set of Component Object Model (COM) objects and interfaces that you can use to create applications that can programmatically administer Analysis Services objects. With the DSO library, you can manage Analysis Services objects, such as servers, databases, data sources, dimensions, cubes, mining models, and roles. You can also administer security, process cubes and mining models, and so on. For more information about the features of Analysis Services, see Analysis Services Features.

The following table lists topics in this section, and describes their contents.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Decision Support Objects</td>
<td>Gives a brief overview of DSO.</td>
</tr>
<tr>
<td>Redistributing Decision Support Objects</td>
<td>Describes the files used to support DSO, including prerequisites and redistribution instructions.</td>
</tr>
<tr>
<td>Decision Support Objects Architecture</td>
<td>Provides information about the implementation of the DSO object model, including discussion of the MDStore interface and a brief description of each object supported by DSO.</td>
</tr>
<tr>
<td>Using Decision Support Objects</td>
<td>Explains how you can use DSO to perform common and advanced tasks in Analysis Services. Sample programs are provided to get you started.</td>
</tr>
<tr>
<td>Decision Support Objects Programmer's Reference</td>
<td>Details the interfaces, objects, collections, methods, and properties in DSO.</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Introducing Decision Support Objects

Microsoft® SQL Server™ 2000 Analysis Services provides a wide range of online analytical processing (OLAP) and data mining functionality. As it is also designed to be very flexible and extensible, you can also add third-party services and packages, such as data mining algorithm providers, to extend the capabilities of Analysis Services even further. However, with a wide range of functionality comes a wide range of complexity, and Analysis Services provides a complex and robust set of OLAP and data mining features.

To access such a rich, extensible, wide-ranging set of features in a simple, straightforward fashion, the Decision Support Objects (DSO) library supplies a hierarchical object model for use with any development environment that can support Component Object Model (COM) objects and interfaces, such as Microsoft Visual C++®, Microsoft Visual Basic®, and Microsoft Visual Basic Scripting Edition.

One of the features of the DSO object model, discussed in another topic, is the use of the MDStore interface. This shared interface allows development environments that use late binding, such as Visual Basic Scripting Edition, to easily support such a complex hierarchical model. The MDStore interface is used in DSO by objects that supply functionality for databases, cubes, partitions, and aggregations in Analysis Services. For more information about the MDStore interface, see MDStore Interface.

See Also

Decision Support Objects Architecture
Analysis Services Programming
Redistributing Decision Support Objects

Decision Support Objects (DSO) is an object library that enables applications to manipulate objects on the Analysis server directly. DSO can be redistributed with custom applications as needed.

Before installing DSO, ensure that Microsoft® Data Access Components (MDAC) and PivotTable® Service have been installed. Then, install the following files.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msmddo80.dll</td>
<td>The DSO library, version 8.0.</td>
</tr>
<tr>
<td>Msmdso.rll</td>
<td>The DSO resource file, version 8.0.</td>
</tr>
<tr>
<td>Msmdnet.dll</td>
<td>The Analysis Services network interface.</td>
</tr>
<tr>
<td>Msmdlock.dll</td>
<td>The Analysis Services lock manager.</td>
</tr>
<tr>
<td>Msmddo.dll</td>
<td>The Microsoft SQL Server™ version 7.0 OLAP Services compatibility file.</td>
</tr>
<tr>
<td></td>
<td>This file is not required if the application will only use features available in SQL Server 2000 Analysis Services.</td>
</tr>
</tbody>
</table>

The DLLs are installed in the following location:

C:\Program Files\Common Files\Microsoft Shared\DSO

The resource file, Msmdso.rll, does not need to be registered and is installed by default in the following location:

C:\Program Files\Common Files\Microsoft Shared\DSO\Resources\1033

To register the DLL files, you should use Regsvr32.exe or use the DLLSelfRegister() functions of the DLL files. Additionally, registry entries for each file should be made under the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs

If this registry value (known as the reference counter) already exists, it should be incremented by one during installation of the DSO files. When uninstalling the DSO files, the reference counter should be decremented by one. The DSO files
should not be deleted if the corresponding reference counter is greater than zero.

**Maintaining Backward Compatibility**

The backward compatibility files that are included with Analysis Services (that is, Msmddo.dll and Msmdsgn.dll) are not compatible with the files of the same name that were included with SQL Server 7.0 OLAP Services (that is, the 7.0 versions of these files). Use the following rules to assist you in determining which files to install:

- If the application will only be used with SQL Server 2000 Analysis Services, Msmddo80.dll, Msmdnet.dll, Msmdlock.dll, and Msmdso.rll must be installed on the target computer.

- If the application will also be used with SQL Server 7.0 OLAP Services, Msmddo.dll must be installed. If the 7.0 version of this file already exists on the target computer, it must be replaced with the newer version.

**CAUTION** Installing the DSO library included with SQL Server 2000 Analysis Services on an Analysis server using SQL Server 7.0 OLAP Services, without upgrading OLAP Manager to Analysis Manager, will cause OLAP Manager to stop functioning.
Analysis Services Programming
**Decision Support Objects Architecture**

Decision Support Objects (DSO) is a library of Component Object Model (COM) classes and interfaces that provide access to the Analysis server. These classes and interfaces, when used together, form an object model that corresponds to the internal structure of the objects managed by Microsoft® SQL Server™ 2000 Analysis Services and can be used to manage them programmatically.

Conceptually, DSO uses hierarchically arranged groups of objects to define basic elements of Analysis Services data storage, as implemented by the Analysis server. These basic elements are databases, data sources, dimensions, cubes, data mining models, and roles. DSO maintains these basic elements in a hierarchical structure where elements contain other elements in a tree, with the server object at the root of the tree. Other objects support this basic structure. For example, databases, cubes, partitions, and aggregations support dimensions. The following diagram shows an overview of the DSO object model hierarchy.

The **DSO Server** object contains a collection that defines databases accessed by the server. Each database can contain groups of objects that define cubes, linked cubes, or virtual cubes. A cube contains one or more partitions, which contain one or more aggregations. Linked cubes serve to provide local server access to a cube on another server; the remote server publishes the cube, and the local server subscribes to it by creating a linked cube. A virtual cube is a special case of a cube, combining portions of the cubes it contains, similar to the way a relational database view combines portions of tables. A database can also contain one or more relational or OLAP data mining models, represented in DSO by the **MiningModel** object. Data mining models can contain one or more data mining columns. Databases also can contain roles, used to manage security on the database and its associated cubes and data mining models.

**MDStore Interface**

You will notice in the diagram that two of the most important collections of objects in DSO, databases and cubes, are supplied with a collection named **MDStores**, unlike other objects in DSO, such as data mining models or
Databases, cubes, partitions, and aggregations expose a common interface, called **MDStore**, that provides the methods and properties you use to manipulate the objects. These objects must be referenced from its parent object, and they cannot be created independently; the only way to create a database, cube, partition, or aggregation is through the **MDStores** collection of the parent object. For example, the only way to obtain an object reference to a database in DSO, or to create a new database using DSO, is through the **MDStores** collection of the DSO **Server** object.

This special collection maintains references to the objects that make up the elements of this hierarchy beneath the server. The **MDStores** collection provides special implementations of **Add**, **Find**, and **Remove** methods, and a convenient **AddNew** method, which maintain parent-child relationships among the various DSO objects. The **ContainedClassType** property determines what type of objects the **MDStores** collection can contain; the value of this property directly corresponds to the **ClassType** property of the **MDStore** objects contained by the collection.

The **MDStore** interface also uses the **SubClassType** property to further differentiate DSO objects. For example, this property is used to tell the difference between a cube, a linked cube, and a virtual cube.

This figure shows an expanded view of the DSO structure as viewed by its collections, including the value of the **ClassType** property of the objects contained in each collection.

**Objects and Interfaces**

There are two major object classifications in DSO: objects that can be accessed and managed directly using their default interface, and objects that implement other DSO interfaces in addition to their default interface.

Objects that can be accessed and managed directly have their own collections, methods, and properties, and they fully implement their default interface; there is no change in the behavior of the interface based on its usage or parent-child association. For example, the DSO **DataSource** object behaves the same way whether the parent is the DSO **Server** object or the DSO **Cube** object.
The following table lists the eight directly accessible DSO interfaces and the **ClassType** property values associated with them.

<table>
<thead>
<tr>
<th>Interface</th>
<th>ClassType property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>clsColumn</td>
</tr>
<tr>
<td>CubeAnalyzer</td>
<td>clsCubeAnalyzer</td>
</tr>
<tr>
<td>DataSource</td>
<td>clsDataSource</td>
</tr>
<tr>
<td>MemberProperty</td>
<td>clsMemberProperty</td>
</tr>
<tr>
<td>MiningModel</td>
<td>clsMiningModel</td>
</tr>
<tr>
<td>PartitionAnalyzer</td>
<td>clsPartitionAnalyzer</td>
</tr>
<tr>
<td>Server</td>
<td>clsServer</td>
</tr>
</tbody>
</table>

Objects that are implemented with more than one interface use a subset of the collections, methods, and properties associated with the interface for the implemented variation of a DSO object; for example, the **Command** interface is implemented differently for database commands, cube commands, and role commands. Each collection, method, and property description contains the names of the objects in which it appears. Conversely, each object description contains the names of the collections, methods, and properties that it implements.

The following table lists the six DSO interfaces and the **ClassType** property values associated with them.

<table>
<thead>
<tr>
<th>Interface</th>
<th>ClassType property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>clsDatabaseCommand</td>
</tr>
<tr>
<td></td>
<td>clsCubeCommand</td>
</tr>
<tr>
<td></td>
<td>clsRoleCommand</td>
</tr>
<tr>
<td>Dimension</td>
<td>clsDatabaseDimension</td>
</tr>
<tr>
<td></td>
<td>clsCubeDimension</td>
</tr>
<tr>
<td></td>
<td>clsPartitionDimension</td>
</tr>
<tr>
<td></td>
<td>clsAggregationDimension</td>
</tr>
<tr>
<td>Level</td>
<td>clsDatabaseLevel</td>
</tr>
</tbody>
</table>
For more information, see [Objects](#) and [Interfaces](#).

**See Also**

[MDStores](#)

[MDStore Interface](#)
Analysis Services Programming
Server (Decision Support Objects)

The **Server** object in the Decision Support Objects (DSO) hierarchy represents the root object in the model tree and handles the functionality related to the Analysis server. The **Server** object uses the **Server** interface, with a **ClassType** property of **clsServer**.

The **Server** object is used to:

- Connect to and disconnect from an Analysis server.

- Start, pause, and stop the Analysis server service (MSSQLServerOLAPService) provider.

- Provide detailed information, such as the version and edition, of an Analysis server.

- Create other Microsoft® SQL Server™ 2000 Analysis Services objects, such as databases, data sources, commands, dimensions, cubes, data mining models, and roles.

- Manage object locking in Analysis Services, controlling read/write access in a multiple user situation.

- Provide access to DSO **Database** objects using the **MDStores** collection.

**See Also**

[Analysis Server](#)

[clsServer](#)

[Database (Decision Support Objects)](#)
MDStore Interface

Working with Servers
Analysis Services Programming
Database (Decision Support Objects)

The **Database** object in Decision Support Objects (DSO) represents a database in Microsoft® SQL Server™ 2000 Analysis Services. Database objects are accessed in DSO through the **MDStores** collection of the DSO Server object. As such, the **Database** object is supported by the **MDStore** interface, with the **ClassType** property set to `clsDatabase`, and the **Database** interface.

The **Database** object is used to:

- Create, edit, and delete commands, data sources, cubes, dimensions, data mining models, and roles applicable to a database in Analysis Services.

- Manage transactions involving objects that belong to the database, such as cubes, dimensions, and mining models.

- Provide access to events, using the **Database** interface, which is used to supply client applications with progress information on currently executing database tasks.

**See Also**

- `clsDatabase`
- **Command (Decision Support Objects)**
- **Cube (Decision Support Objects)**
- **Data Mining Model (Decision Support Objects)**
- **Databases**
- **DataSource (Decision Support Objects)**
- **Dimension (Decision Support Objects)**

**MDStore Interface**
Role (Decision Support Objects)
Server (Decision Support Objects)
Working with Databases
Analysis Services Programming
**DataSource (Decision Support Objects)**

The **DataSource** object in Decision Support Objects (DSO) provides access to the data sources associated with a database, cube, partition, or aggregation in Microsoft® SQL Server™ 2000 Analysis Services, under the **DataSources** collection for each DSO object. The **DataSource** object uses the **DataSource** interface, with a **ClassType** property of **clsDataSource**.

The **DataSource** object is used to:

- Retrieve data source-specific information such as connection strings and quote characters for use by client applications.
- Determine various states, such as connection state, of the data source in Analysis Services.

**See Also**

- [Aggregation (Decision Support Objects)](#)
- [clsDataSource](#)
- [Cube (Decision Support Objects)](#)
- [Data Sources](#)
- [Database (Decision Support Objects)](#)
- [Partition (Decision Support Objects)](#)
- [Working with Data Sources](#)
Analysis Services Programming
Cube (Decision Support Objects)

The **Cube** object in Decision Support Objects (DSO) provides access to cubes, virtual cubes, and linked cubes associated with a database in Microsoft® SQL Server™ 2000 Analysis Services, supplied by the **MDStores** collection of the DSO **Database** object using the **MDStore** interface. The **Cube** object uses the **MDStore** interface, with a **ClassType** property of **clsCube**.

The **Cube** object is used to:

- Provide access to the commands, data sources, dimensions, measures, partitions, and roles associated with a cube, virtual cube, or linked cube in Analysis Services.

- Edit the structure of a cube, virtual cube, or linked cube.

- Process a cube, virtual cube, or linked cube.

- Manage object locks for a cube, virtual cube, or linked cube.

**See Also**

- **clsCube**
- **Command (Decision Support Objects)**
- **Cubes**
- **Database (Decision Support Objects)**
- **DataSource (Decision Support Objects)**
- **Dimension (Decision Support Objects)**
- **MDStore Interface**
- **Measure (Decision Support Objects)**
- **Partition (Decision Support Objects)
Role (Decision Support Objects)

Working with Cubes and Measures
Analysis Services Programming
Dimension (Decision Support Objects)

The Dimension object in Decision Support Objects (DSO) provides access to shared dimensions, virtual dimensions, and private dimensions in Microsoft® SQL Server™ 2000 Analysis Services for several objects, including databases, cubes, partitions, and aggregations. Shared dimensions and virtual dimensions are accessed using the Dimensions collection of the DSO Database object, while private dimensions are accessed using the Dimensions collection of the DSO Cube object. Partitions and aggregations, associated with a source cube, also support access to the dimensions related to them. Partition and aggregation dimensions are also accessed by the Dimensions collection of the Partition and Aggregation DSO objects, respectively.

All four types of dimensions are supported with DSO objects. Each dimension object detailed in the following table uses the Dimension interface, with the ClassType property set to the appropriate value for the dimension type as shown.

<table>
<thead>
<tr>
<th>Dimension type</th>
<th>Dimension object</th>
<th>Dimension class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database dimensions</td>
<td>DbDimension</td>
<td>clsDatabaseDimension</td>
</tr>
<tr>
<td>Cube dimensions</td>
<td>CubeDimension</td>
<td>clsCubeDimension</td>
</tr>
<tr>
<td>Partition dimension</td>
<td>PartitionDimension</td>
<td>clsPartitionDimension</td>
</tr>
<tr>
<td>Aggregation dimension</td>
<td>AggregationDimension</td>
<td>clsAggregationDimension</td>
</tr>
</tbody>
</table>

The Dimension object is used to:

- Provide access to shared dimensions, virtual dimensions, and private dimensions in Analysis Services.

- Create and edit levels for a shared dimension or private dimension.

- Determine the various states of a shared dimension, virtual dimension, or private dimension, such as the temporary status and validity.
Process a shared dimension, virtual dimension, or private dimension.

See Also

Aggregation (Decision Support Objects)

clsDatabaseDimension

clsCubeDimension

clsPartitionDimension

clsAggregationDimension

Cube (Decision Support Objects)

Database (Decision Support Objects)

Dimension Interface

Dimensions

Partition (Decision Support Objects)

Working with Dimensions and Levels
Analysis Services Programming
Data Mining Model (Decision Support Objects)

The **MiningModel** object in Decision Support Objects (DSO) provides support for data mining models in Microsoft® SQL Server™ 2000 Analysis Services. To access data mining models, the **MiningModels** collection of the DSO **Database** object is used. The **MiningModel** object uses the **MiningModel** interface, with a **ClassType** property of **clsMiningModel**.

The **MiningModel** object is used to:

- Provide access to data mining columns for a relational or OLAP data mining model.

- Construct and modify relational or OLAP data mining models.

- Process a relational or OLAP data mining model.

- Provide access to mining model roles.

**See Also**

**clsMiningModel**

**Column (Decision Support Objects)**

**Data Mining Models**

**Data Mining Examples**

**Database (Decision Support Objects)**

**Role (Decision Support Objects)**
Analysis Services Programming
Role (Decision Support Objects)

The **Role** object in Decision Support Objects (DSO) provides access to role-based security for databases, cubes, and data mining models. The DSO **Database**, **Cube**, and **MiningModel** objects all support the **Roles** collection, making security functions available for each DSO object.

All three types of roles are supported with DSO objects. Each role object detailed in the following table uses the **Role** interface, with the **ClassType** property set to the appropriate value for the role type as shown.

<table>
<thead>
<tr>
<th>Role type</th>
<th>Role object</th>
<th>Role class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database role</td>
<td>DbGroup</td>
<td>clsDatabaseRole</td>
</tr>
<tr>
<td>Cube role</td>
<td>CubeGroup</td>
<td>clsCubeRole</td>
</tr>
<tr>
<td>Mining model role</td>
<td>MiningModelGroup</td>
<td>clsMiningModelRole</td>
</tr>
</tbody>
</table>

The **Role** object is used to:

- Provide access to the list of users for a role associated with a database, cube, or data mining model.

- Create or modify permissions on a role associated with a database, cube, or data mining model.

- Provide access to commands for a role associated with a database, cube, or data mining model.

**See Also**

- [clsDatabaseRole](#)
- [clsCubeRole](#)
- [clsMiningModelRole](#)
- [Cube (Decision Support Objects)](#)
Data Mining Model (Decision Support Objects)
Database (Decision Support Objects)
Command (Decision Support Objects)
Roles
Analysis Services Programming
Aggregation (Decision Support Objects)

The Aggregation object in Decision Support Objects (DSO) provides access to aggregations associated with a partition in Microsoft® SQL Server™ 2000 Analysis Services. The MDStores collection of the DSO Partition object allows access to aggregations. The Aggregation object uses the MDStore interface, with a ClassType property of clsAggregation.

The Aggregation object is used to:

- Provide access to the dimensions, data sources, and measures associated with an aggregation.

- Edit the properties and objects associated with an aggregation.

- Determine the various states of an aggregation, such as validity and processing status.

See Also

Aggregations
clsAggregation
DataSource (Decision Support Objects)
Dimension (Decision Support Objects)
MDStore Interface
Measure (Decision Support Objects)
Analysis Services Programming
**Command (Decision Support Objects)**

The **Command** object in Decision Support Objects (DSO) provides access to commands associated with databases, cubes, and roles in Microsoft® SQL Server™ 2000 Analysis Services. The DSO **Database**, **Cube**, and **Role** objects support the **Commands** collection for access to associated commands.

All three types of commands are supported with DSO objects. Each command object detailed in the following table uses the **Command** interface, with the **ClassType** property set to the appropriate value for the command type as shown.

<table>
<thead>
<tr>
<th>Command type</th>
<th>Command object</th>
<th>Command class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database command</td>
<td>DbCommand</td>
<td>clsDatabaseCommand</td>
</tr>
<tr>
<td>Cube command</td>
<td>CubeCommand</td>
<td>clsCubeCommand</td>
</tr>
<tr>
<td>Role command</td>
<td>RoleCommand</td>
<td>clsRoleCommand</td>
</tr>
</tbody>
</table>

The **Command** object is used to:

- Provide access to the properties, such as the statement and ordinal position, of a command associated with a database, cube, or role.

- Manage locks for a command associated with a database, cube, or role.

**See Also**

[Commands](#)

[clsDatabaseCommand](#)

[clsCubeCommand](#)

[clsRoleCommand](#)

**Cube (Decision Support Objects)**

**Database (Decision Support Objects)**

**Role (Decision Support Objects)**
Analysis Services Programming
Level (Decision Support Objects)

The **Level** object in Decision Support Objects (DSO) provides access to levels associated with dimensions in Microsoft® SQL Server™ 2000 Analysis Services. The **Levels** collection of the DSO **Dimension** object provides access to DSO **Level** objects. In addition, the **Levels** collection is supported by the **Cube**, **Aggregation**, and **Partition** DSO objects.

All four types of levels are supported with DSO objects. Each level object detailed in the following table uses the **Level** interface, with the **ClassType** property set to the appropriate value for the level type as shown.

<table>
<thead>
<tr>
<th>Level type</th>
<th>Level object</th>
<th>Level class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database level</td>
<td>DbLevel</td>
<td>clsDatabaseLevel</td>
</tr>
<tr>
<td>Cube level</td>
<td>CubeLevel</td>
<td>clsCubeLevel</td>
</tr>
<tr>
<td>Partition level</td>
<td>PartitionLevel</td>
<td>clsPartitionLevel</td>
</tr>
<tr>
<td>Aggregation level</td>
<td>AggregationLevel</td>
<td>clsAggregationLevel</td>
</tr>
</tbody>
</table>

The **Level** object is used to:

- Provide access to the member properties associated with a level.

- Edit the properties, such as level type and custom rollup expression, associated with a level.

**See Also**

- clsDatabaseLevel
- clsCubeLevel
- clsPartitionLevel
- clsAggregationLevel

**Dimension (Decision Support Objects)**

**Levels**
Working with Dimensions and Levels
Analysis Services Programming
Measure (Decision Support Objects)

The Measure object in Decision Support Objects (DSO) provides support for measures. The Measures collection of the DSO Cube, Partition, and Aggregation objects provides access to cube, partition, and aggregation measures, respectively.

All three types of measures are supported with DSO objects. Each measure object detailed in the following table uses the Measure interface, with the ClassType property set to the appropriate value for the level type as shown.

<table>
<thead>
<tr>
<th>Measure type</th>
<th>Measure object</th>
<th>Measure class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube measure</td>
<td>CubeMeasure</td>
<td>clsCubeMeasure</td>
</tr>
<tr>
<td>Partition measure</td>
<td>PartitionMeasure</td>
<td>clsPartitionMeasure</td>
</tr>
<tr>
<td>Aggregation measure</td>
<td>AggregationMeasure</td>
<td>clsAggregationMeasure</td>
</tr>
</tbody>
</table>

The Measure object is used to edit the properties, such as the aggregation function and source column, associated with a measure in Microsoft® SQL Server™ 2000 Analysis Services.

See Also

Aggregation (Decision Support Objects)
clsCubeMeasure
clsPartitionMeasure
clsAggregationMeasure
Cube (Decision Support Objects)
Measures
Partition (Decision Support Objects)
Working with Cubes and Measures
Analysis Services Programming
Member Property (Decision Support Objects)

The MemberProperty object in Decision Support Objects (DSO) provides access to member properties associated with levels in Microsoft® SQL Server™ 2000 Analysis Services. The MemberProperties collection of the DSO Level object supports access to the member properties associated with a level. The MemberProperty object uses the MemberProperty interface, with a ClassType property of clsMemberProperty.

The MemberProperty object is used to edit the properties, such as column type and language, of a member property in Analysis Services.

See Also

clsMemberProperty

Dimension (Decision Support Objects)

Level (Decision Support Objects)

Member Properties

Working with Dimensions and Levels
Analysis Services Programming
Partition (Decision Support Objects)

The **Partition** object in Decision Support Objects (DSO) provides access to partitions in Microsoft® SQL Server™ 2000 Analysis Services. Access to partitions associated with cubes in Analysis Services is supplied by the **MDStores** collection of the DSO **Cube** object. As such, the **Partition** object uses the **MDStore** interface, with a **ClassType** property of **clsPartition**.

The **Partition** object is used to:

- Provide access to data sources, dimensions, aggregations, and measures associated with a partition in Analysis Services.
  
- Update partitions programmatically.
  
- Manage locks on partitions.

See Also

- [Aggregation (Decision Support Objects)](#)
- [clsPartition](#)
- [Cube (Decision Support Objects)](#)
- [DataSource (Decision Support Objects)](#)
- [Dimension (Decision Support Objects)](#)
- [MDStore Interface](#)
- [Measure (Decision Support Objects)](#)
- [Partitions](#)
Analysis Services Programming
Column (Decision Support Objects)

The Column object in Decision Support Objects (DSO) provides access to data mining columns in Microsoft® SQL Server™ 2000 Analysis Services. The Columns collection of the MiningModel and Column objects are used to access data mining columns and nested data mining columns, respectively, in Analysis Services. The Column object uses the Column interface, with a ClassType property of clsColumn.

The Column object is used to:

- Provide access to nested data mining columns associated with a column in a data mining model.

- Edit properties, such as content type and modeling flags, for a column in Analysis Services.

See Also

clsColumn
Data Mining Columns
Data Mining Examples
Data Mining Model (Decision Support Objects)
Analysis Services Programming
Cube Analyzer (Decision Support Objects)

The **CubeAnalyzer** object in Decision Support Objects (DSO) provides access to analysis functions for cubes in Microsoft® SQL Server™ 2000 Analysis Services. The **Analyzer** property of the DSO **Cube** object provides access to the **CubeAnalyzer** object. The **CubeAnalyzer** object uses the **CubeAnalyzer** interface, with a **ClassType** property of **clsCubeAnalyzer**.

The **CubeAnalyzer** object is used to provide access to the query log of a cube in Analysis Services.

**See Also**

[clsCubeAnalyzer](#)

[Cube (Decision Support Objects)](#)
Analysis Services Programming
Partition Analyzer (Decision Support Objects)

The **PartitionAnalyzer** object in Decision Support Objects (DSO) provides access to the partition analysis tools for a partition in Microsoft® SQL Server™ 2000 Analysis Services. It is accessed through the **Analyzer** property of the DSO **Partition** object. The **PartitionAnalyzer** object uses the **PartitionAnalyzer** interface, with a **ClassType** property of **clsPartitionAnalyzer**.

The **PartitionAnalyzer** object is used to:

- Provide access to the aggregation analysis capabilities for a partition in Analysis Services.

- Provide additional information, such as existing designed aggregations and goal queries, to the aggregation analysis process for a partition.

**See Also**

- **clsPartitionAnalyzer**
- **Partition (Decision Support Objects)**
Analysis Services Programming
Using Decision Support Objects

You can use Decision Support Objects (DSO) in your applications to administer the server and to create and maintain OLAP and data mining objects. DSO objects can also be created and used to incorporate the server functions of Microsoft® SQL Server™ 2000 Analysis Services into client applications. For information about the programming environments in which you can use DSO to create such client applications, see Development Environments.

A common sequence of operations for an application using DSO is:

1. Connect to an Analysis server.

2. Create a database object to contain dimensions and cubes.

3. Add a data source that contains the data.

4. Create dimensions and their levels.

5. Create a cube and specify dimensions and measures.

6. Process a cube to load its structure and data.

These operations are described and illustrated using Microsoft Visual Basic®, Microsoft Visual Basic Scripting Edition, and Microsoft Visual C++® code in Common Operations and Examples.
Analysis Services Programming
Development Environments

You can use Decision Support Objects (DSO) with any language that supports COM Automation, such as Microsoft® Visual Basic®, Microsoft Visual Basic Scripting Edition, or Microsoft Visual C++®.

Visual Basic

You can use the DSO library in Visual Basic simply by setting the Decision Support Objects library as a project reference.

Example

Dim dsoServer As New DSO.Server
dsoServer.Connect "LocalHost"

Setting Program References

To use the DSO library in a Visual Basic application, add it as a reference to the project. On the Project menu, click References, and then select Microsoft Decision Support Objects.

Visual Basic Scripting Edition

Visual Basic Scripting Edition programmers can use DSO by using the CreateObject method. This method can be used to create any needed object in the DSO libraries.

Example

The following example creates a DSO.Server object:

Dim dsoServer
Set dsoServer = CreateObject("DSO.Server")
dsoServer.Connect "LocalHost"

In order to support scripting, the DSO Database, Cube, Partition, and
**Aggregation** objects implement the **MDStore** class interface as well as their own class interface; this is the default interface used for **MDStores** collections of DSO objects. The following code example illustrates the retrieval of a database from a server using the **MDStores** collection:

```vbs
Dim dsoDatabase
Set dsoDatabase = dsoServer.MDStores(1)
```

The **ClassType** and **SubClassType** properties of the **MDStore** interface allow for class determination. If the class-specific interface for a particular object is needed, first use the **CreateObject** command to create an instance of the object, then set it to the required object in the **MDStores** collection. This is demonstrated in the following code example:

```vbs
Dim dsoServer
Dim dsoDatabase
Set dsoServer = CreateObject("DSO.Server")
dsoServer.Connect "LocalHost"
Set dsoDatabase = CreateObject("DSO.Database")
Set dsoDatabase = dsoServer.MDStores(1)
```

**CAUTION** It is recommended that, for any DSO object supporting the **MDStore** interface, the class-specific interface not be used.
Analysis Services Programming
Common Operations and Examples

This topic provides basic and advanced examples, in a step-by-step format, that show you how to perform common operations using Decision Support Objects (DSO). These examples assume that you have some experience in creating Microsoft® Visual Basic® or Visual C++® database applications. The examples provide complete routines in Visual Basic that perform the following functions:

- Connects to your Analysis server
- Accesses the **FoodMart 2000** sample database
- Adds a new database and data source
- Adds dimensions and levels
- Adds a cube and measures
- Processes the cube
- Creates a virtual cube
- Creates a linked cube
- Creates virtual dimensions
- Performs incremental updates on a cube

**Note** You can also find sample applications on the Microsoft SQL Server™ 2000 CD-ROM in the `\MSOLAP\Samples` directory. These illustrate some of the techniques you can use for developing your own applications that use DSO or
PivotTable® Service. The Readme.txt file in the folder provides descriptions of the individual samples and instructions for installing them on your computer. Examples include displaying meta data and data over the Web, creating and processing cubes with DSO, and creating a write-enabled cube.

The examples use DSO objects to create a cube derived from the **FoodMart 2000** sample database, which is provided with Microsoft SQL Server 2000 Analysis Services. It uses the **Sales_Fact_1997**, **Product**, **Store**, and **Time_By_Day** tables in a star schema, as shown here.

The following dimensions and measures are also used, as shown here.

**Remarks**

You may also find it helpful to review the structure of the **FoodMart 2000** database itself by examining it either in Microsoft Access or in a third-party computer-aided software engineering (CASE) tool. This will help you to understand how the preceding steps are applied to the specific instance of the **FoodMart 2000** database.

Some of these exercises can modify the **FoodMart 2000** sample database. We recommend that you make backup copies of the following directories before you proceed:

- C:\Program Files\Microsoft Analysis Services\Bin

- C:\Program Files\Microsoft Analysis Services\Data
Analysis Services Programming

**OLAP Examples**

The basic examples provided in this topic cover the most commonly employed functions of the Decision Support Objects (DSO) library. All of the examples are written in Microsoft® Visual Basic®, with additional code examples in Microsoft Visual C++® given on selected topics. The following table lists the topics covered.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Working with Servers</strong></td>
<td>Describes how to connect to an Analysis server</td>
</tr>
<tr>
<td><strong>Working with Databases</strong></td>
<td>Explains how to list, add, and delete databases for an Analysis server</td>
</tr>
<tr>
<td><strong>Working with Data Sources</strong></td>
<td>Shows how to list data sources on, add new data sources to, or delete existing data sources for a database</td>
</tr>
<tr>
<td><strong>Working with Dimensions and Levels</strong></td>
<td>Demonstrates how to list, add, and delete shared dimensions for a database</td>
</tr>
<tr>
<td><strong>Working with Cubes and Measures</strong></td>
<td>Describes how to list, add, delete, and process cubes for a database, including examples on how to list and add measures to a cube</td>
</tr>
</tbody>
</table>

**Note** The examples presented in this topic are for educational purposes. The code is intended to illustrate the functionality of the DSO library, and it does not contain error-handling routines.
Analysis Services Programming

**Working with Servers**

The example code in this topic shows you how to connect to an Analysis server using Decision Support Objects (DSO) and list some of its property values.

If the Analysis server is not installed on the computer on which you are running this example, change `LocalHost` to the name of the Microsoft® Windows NT® Server 4.0 or Windows® 2000 computer where the Analysis server is installed and running.

**List Servers**

The following code example illustrates the use of the `DSO.Server` object in connecting to and retrieving attributes from an Analysis server.

**Example**

The following code example connects to the local Analysis server using the `Connect` method of the `DSO.Server` object, and then it displays some of the basic server properties in the Immediate window:

```vbscript
Private Sub ListServerProps()
    Dim dsoServer As DSO.Server
    Dim enuClassType As DSO.ClassTypes

    ' Create instance of server and connect.
    ' "LocalHost" will default to the
    ' local Windows NT Server 4.0 where the
    ' Analysis server is installed.
    Set dsoServer = New DSO.Server
    dsoServer.Connect "LocalHost"

    ' Show the server's information to the user.
    If dsoServer.ClassType = clsServer Then
```

```vbscript
```
Debug.Print "Server Properties:"
Debug.Print " Name: " & dsoServer.Name
Debug.Print "Description: " & dsoServer.Description
' The Machine property is a hidden property of the
' DSO.Server object.
End If

End Sub
Analysis Services Programming

**Working with Databases**

Each Analysis server contains an **MDStores** collection of database objects (that is, objects of **ClassType clsDatabase**). A database in Decision Support Objects (DSO) contains dimensions and their subordinate levels, data sources, roles, and commands. Each database object also contains an **MDStores** collection of cube objects (that is, objects of **ClassType clsCube**.)

The following examples discuss the methods used to list, add, and delete databases on an Analysis server.

**List Databases**

The following code example connects to the specified DSO server and iterates through all of the databases on that server, using the **MDStores** collection for the Analysis server object.

**Example**

When executed, the following code example prints the name and description of every database defined for the specified Analysis server:

```vbnet
Private Sub ListDatabases()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore

    ' Connect to the server.
    dsoServer.Connect "LocalHost"

    ' For each MDStore database object on the server,
    ' print its name.
    For Each dsoDB In dsoServer.MDStores
        Debug.Print "Database: " & dsoDB.Name & _
        " - " & dsoDB.Description
    Next dsoDB
End Sub
```
Create a New Database

To create an **MDStore** object of **ClassType clsDatabase** on the server, the **AddNew** method of the **Server.MDStores** database object collection is used in the following code example.

**Example**

The following code example creates a new database, **TestDB**, on the local Analysis server:

```vbnet
Private Sub CreateDatabase()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore

    Dim strDBName As String
    Dim strDBDesc As String

    ' Create a connection to the Analysis server.
    dsoServer.Connect "LocalHost"

    ' Initialize the string variables for the new database name and description.
    strDBName = "TestDB"
    strDBDesc = "Test Database"

    ' Is there already a database by this name?
    If dsoServer.MDStores.Find(strDBName) Then
        MsgBox strDBName & " already exists."
        Exit Sub
    End If
End Sub
```
'Add new database to server object collection.
' Using the AddNew method from MDStores.
Set dsoDB = dsoServer.MDStores.AddNew(strDBName)

'Assign the description to the MDStore's
'Description property, and then call the Update method.
dsoDB.Description = strDBDesc
dsoDB.Update

'Inform the user that the database was added to the server.
MsgBox (strDBName & " added to server " & dsoServer.Name)

End Sub

**Remove a Database**

To remove a database, use the **Remove** method of the Server.MDStores
collection as demonstrated in the following code example.

**Example**

The following code example removes the **TestDB** database from the local
Analysis server:

Private Sub RemoveDatabase()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore

    Dim strDBName As String
    Dim blnResult As Boolean

    ' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"
' Set the database name variable to TestDB.
strDBName = "TestDB"

' Check to make sure that the TestDB database is on
' the Analysis server.
If dsoServer.MDStores.Find(strDBName) Then
   ' The database was on the server.

   ' Delete the TestDB database.
   dsoServer.MDStores.Remove strDBName

   ' Inform the user.
   MsgBox strDBName & " removed from server " & dsoServer.Name
Else
   ' The database was not on the server.

   ' Inform the user.
   MsgBox strDBName & " not found on server " & dsoServer.Name
End If

End Sub

**Note** The **TestDB** database is referred to in other examples. Use the code example provided earlier in this topic to re-create the database for other examples.

**See Also**

[Databases](#)
Analysis Services Programming

**Working with Data Sources**

Collections of data sources (that is, **MDStore** objects of **ClassType clsDataSource**) are contained in **MDStore** objects of **ClassType clsDatabase**, **clsCube**, and **clsPartition**. Each object's data source specifies an external database that will be used as the source of data.

A database can contain multiple data sources in its **DataSources** collection. However, each cube and partition contains only a single data source.

The two examples in this topic demonstrate how to list and add a data source to the database's **DataSources** collection.

**List Data Sources**

The easiest way to list data sources is to iterate through the **DataSources** collection of an **MDStore** database object, as shown in the following code example which lists the **Name** and **ConnectionString** properties of each data source for every database on a given Analysis server.

**Example**

The following code example loops through the **DataSources** collection of each database on the local Analysis server, printing some of the basic properties for each data source in the Immediate window:

```vba
Private Sub ListDataSources()
    Dim dsoServer As New dso.Server
    Dim dsoDB As dso.MDStore
    Dim dsoDS As dso.DataSource

    ' Create a connection to the Analysis server.
    dsoServer.Connect "LocalHost"

    ' Step through the databases in the
' MDStores collection of the server.
For Each dsoDB In dsoServer.MDStores

' Print the name & description of the database.
Debug.Print "DATABASE: " & dsoDB.Name & " - " & _
    dsoDB.Description

' Determine whether the database has data sources.
If dsoDB.DataSources.Count = 0 Then
    Debug.Print "    Data source: None"
Else
    ' Iterate through and print the data source
    ' information.
    For Each dsoDS In dsoDB.DataSources
        Debug.Print "    Data source: " & dsoDS.Name
        Debug.Print "    Connect String: " & dsoDS.ConnectionString
    Next
End If
Next

End Sub

Add a Data Source

The process to add a new data source is similar to the process for adding a new database. The AddNew method of the DataSources collection for a given database creates a new data source for the database.

Example

The following code example adds a data source named **FoodMart** to the **TestDB** database object on the local Analysis server:

Private Sub AddDataSource()
    Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDS As DSO.DataSource
Dim strDBName As String
Dim strDSName As String
Dim strDSConnect As String

' Initialize variables for the database name, data source name, and the ConnectionString property for the data source.
strDBName = "TestDB"
strDSName = "FoodMart"
strDSConnect = "Provider=MSDASQL.1;User ID=sa;" & _
    "Data Source=FoodMart;Connect Timeout=15"

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Locate the database first.
If dsoServer.MDStores.Find(strDBName) Then
    Set dsoDB = dsoServer.MDStores(strDBName)
    ' Check to see whether the data source already exists.
    If dsoDB.DataSources.Find(strDSName) Then
        MsgBox "Data source " & strDSName & _
            " already exists for database " & strDBName
    Else
        ' Create a new data source.
        Set dsoDS = dsoDB.DataSources.AddNew(strDSName)
        ' Add the ConnectionString properties
        dsoDS.ConnectionString = strDSConnect
        ' Update the data source.
        dsoDS.Update
'Inform the user
MsgBox "Data source " & strDSName & 
" has been added to database " & strDBName
End If
Else
MsgBox strDBName & " is missing."
End If

End Sub

See Also

Data Sources
Working with Dimensions and Levels

The dimensions of a cube store data derived from relational database tables and contain the categorical data you want to analyze.

The dimensions you build should be distinct categories that you want to add to cubes in your database (such as Time, Customer Education, and Customer Age). A dimension can be created from a single dimension table (star schema) or from multiple dimension tables (snowflake schema). Dimensions are classified as either standard or time dimensions, depending upon the data type of the corresponding column in the dimension table.

Collections of dimensions are contained within objects of `ClassType clsDatabase, clsCube, clsPartition, and clsAggregation`. The dimension objects contained within each of these collections are of respective `ClassTypes clsDatabaseDimension, clsCubeDimension, clsPartitionDimension, and clsAggregationDimension`.

The List Dimensions example lists existing dimensions and their related levels. The Add Dimensions example creates new dimensions and levels.

List Dimensions

The `Dimensions` collection of the `DSO.Server` object contains all shared dimensions on an Analysis server, as illustrated by the following code example.

Example

The following code example illustrates the hierarchical nature of dimensions and levels by listing the levels in order of precedence for every dimension contained in every database on a given Analysis server, printing basic properties of each dimension and level in the Immediate window:

Private Sub ListDimensions()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoDim As DSO.Dimension
Dim dsoLev As DSO.Level

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Enumerate databases on a server.
For Each dsoDB In dsoServer.MDStores
    Debug.Print "DATABASE: " & dsoDB.Name & " - " & _
    dsoDB.Description

' Enumerate dimensions in a database.
For Each dsoDim In dsoDB.Dimensions
    Debug.Print " Dimension: " & dsoDim.Name

    ' Enumerate levels in a dimension.
    For Each dsoLev In dsoDim.Levels
        Debug.Print " Level: " & dsoLev.Name
    Next
    Next
    Next

End Sub

Add Dimensions and Levels

To add a dimension, the AddNew method of the Dimensions collection for an MDStore database object is used. After a new dimension has been created in this way, levels are added using the AddNew method of the Levels collection for the new Dimension object.

The following code example adds the Products dimension with two levels, Brand Name and Product Name, and the Stores dimension with four levels, Store Country, Store State, Store City and Store ID, to the TestDB database. In addition, the Store ID level has a member property named Store SQFT associated with it. The following diagram graphically displays the relationships.
For more information about member properties, see `clsMemberProperty`.

**Note** The **TestDB** database is created using some of the prior examples in this topic. For more information, see *Working with Databases*.

**Example**

The following code example adds two new dimensions to the **TestDB** database:

```vba
Private Sub AddDimensions()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoDS As DSO.DataSource
    Dim dsoDim As DSO.Dimension
    Dim dsoLev As DSO.Level
    Dim dsoMember As DSO.MemberProperty

    Dim strDBName As String

    ' Constants used for ColumnType property
    ' of the DSO.Level object.
    ' Note that these constants are identical to
    ' those used in ADO in the DataTypeEnum enumeration.
    Const adWChar = 130
    Const adInteger = 3
    Const adDouble = 5

    ' Initialize variables for the database name.
    strDBName = "TestDB"

    ' Create a connection to the Analysis server.
    dsoServer.Connect "LocalHost"

    ' Set the database object.
```
Set dsoDB = dsoServer.MDStores(strDBName)

' Set the data source for the database object.
' A data source is required to run this example.
If dsoDB.DataSources.Count = 0 Then
    MsgBox "Database " & dsoDB.Name & ": has no data sources."
Else
    Set dsoDS = dsoDB.DataSources(1)
End If

' Create Products dimension and levels.
Set dsoDim = dsoDB.Dimensions.AddNew("Products")
Set dsoDim.DataSource = dsoDS ' Dimension data source
dsoDim.FromClause = "product" ' Related table
dsoDim.JoinClause = "" ' Used in snowflake schema

' Add Brand Name level.
Set dsoLev = dsoDim.Levels.AddNew("Brand Name")
dsoLev.MemberKeyColumn = """"product"".""brand_name""
dsoLev.ColumnSize = 255 ' Column data size in bytes
dsoLev.ColumnType = adWChar ' Column data type
dsoLev.EstimatedSize = 1 ' Distinct members in column

' Add Product Name level.
Set dsoLev = dsoDim.Levels.AddNew("Product Name")
dsoLev.MemberKeyColumn = """"product"".""product_name""
dsoLev.ColumnSize = 255
dsoLev.ColumnType = adWChar
dsoLev.EstimatedSize = 1

' Update the Products dimension.
dsoDim.Update
'Inform the user.
MsgBox "Dimension " & dsoDim.Name & " added to " & _
    dsoDim.DataSource.Name & " data source."

' Create Stores dimension and levels.
Set dsoDim = dsoDB.Dimensions.AddNew("Stores")
Set dsoDim.DataSource = dsoDS ' Dimension data source
dsoDim.FromClause = "store" ' Related table
dsoDim.JoinClause = "" ' Used in snowflake schema

' Add Store Country level.
Set dsoLev = dsoDim.Levels.AddNew("Store Country")
dsoLev.MemberKeyColumn = "store"."store_country"
dsoLev.ColumnSize = 50 ' Column data size in bytes
dsoLev.ColumnType = adWChar ' Column data type
dsoLev.EstimatedSize = 1 ' Distinct members in column

' Add Store State level.
Set dsoLev = dsoDim.Levels.AddNew("Store State")
dsoLev.MemberKeyColumn = "store"."store_state"
dsoLev.ColumnSize = 50
dsoLev.ColumnType = adWChar
dsoLev.EstimatedSize = 1

' Add Store City level.
Set dsoLev = dsoDim.Levels.AddNew("Store City")
dsoLev.MemberKeyColumn = "store"."store_city"
dsoLev.ColumnSize = 50
dsoLev.ColumnType = adWChar
dsoLev.EstimatedSize = 1

' Add Store ID level.
Set dsoLev = dsoDim.Levels.AddNew("Store ID")
dsoLev.MemberKeyColumn = "store"."store_ID"
dsoLev.ColumnSize = 4
dsoLev.ColumnType = adInteger
dsoLev.EstimatedSize = 1

' Add a member property to the Store ID level.
Set dsoMember = dsoLev.MemberProperties.AddNew("Store SQFT
sbclsRegular)
dsoMember.Description = "Store size in square feet"
dsoMember.SourceColumn = "store"."store_sqft"
dsoMember.ColumnSize = 4
dsoMember.ColumnType = adDouble

' Update the Stores dimension.
dsoDim.Update

' Inform the user.
MsgBox "Dimension " & dsoDim.Name & " added to " & _
dsoDim.DataSource.Name & " data source."

End Sub

See Also

Dimensions

Levels and Members
Analysis Services Programming

**Working with Cubes and Measures**

Each database contains an **MDStores** collection of cubes (that is, objects of **ClassType clsCube**). A cube is the central object in a multidimensional database. A cube contains dimensions and their levels, measures, data sources, roles, and commands. Each cube also contains an **MDStores** collection of partitions (that is, objects of **ClassType clsPartition**).

The previous examples created a new database, added a data source, and added shared dimensions and levels. The following three examples demonstrate how to list, add, and remove a cube.

**List Cubes**

Because each **MDStore** database object contains a collection of **MDStore** cube objects, it is easy to list the cubes and their properties for each database.

**Example**

The following code example prints a list of cubes for each database on a given server to the Immediate window. The **SubClassType** and **SourceTable** properties for each cube are also printed.

```vba
Private Sub ListCubes()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore

    ' Create a connection to the Analysis server.
    dsoServer.Connect "LocalHost"

    ' Step through the databases in the server object.
    For Each dsoDB In dsoServer.MDStores
        ' Print the name and description of the database
        Debug.Print "DATABASE: " & dsoDB.Name & " - " & _
```
dsoDB.Description

' Step through the cubes in the database object.
If dsoDB.MDStores.Count = 0 Then
    Debug.Print " Cube: None"
Else
    For Each dsoCube In dsoDB.MDStores
        ' Print the name of the cube.
        Debug.Print " Cube: " & dsoCube.Name

        ' Check to see whether the cube is regular or virtual.
        If dsoCube.SubClassType = sbclsRegular Then
            Debug.Print " SubClassType: Regular"
            Debug.Print " SourceTable: " & _
            dsoCube.SourceTable
        Else
            Debug.Print " SubClassType: Virtual"
        End If
    Next
End If
Next
End Sub

Add a Cube

The following example illustrates how to add a cube, named TestCube, to the MDStores collection of the database:

1. Add the cube to the MDStores collection of the database using the AddNew method.

2. Add a data source to the new cube.
3. Set the **SourceTable** property of the cube.

4. Set the **EstimatedRows** property to the approximate number of rows in the table.

5. Add the shared dimensions that you created in the Working With Dimensions and Levels topic to the cube's **Dimensions** collection with the **AddNew** method.

6. Create an SQL INNER JOIN clause for connecting the dimension table to the source table, and then assign it to the cube's **JoinClause** property.

7. Make the changes to this cube permanent by calling the cube's **Update** method.

**Example**

The following code example adds a new cube, TestCube, to the **TestDB** database:

```vba
Private Sub AddCube()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore

    Dim strDBName As String
    Dim strCubeName As String
    Dim strJoin As String

    ' Initialize variables for the database and ' cube names.
    strDBName = "TestDB"
```
strCubeName = "TestCube"

' Define joins between the fact table and the dimension tables
' to be used later in the subroutine.

' Join the fact table to the Product table.
' sales_fact_1998.product_id = product.product_id
strJoin = "('sales_fact_1997'."product_id"='product'."product_id")"
strJoin = strJoin & " AND "

' Join the fact table to the Store table.
' sales_fact_1998.store_id = store.store_id
strJoin = strJoin & "('sales_fact_1997'."store_id"='store'."store_id")"

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Ensure the database exists first.
If dsoServer.MDStores.Find(strDBName) = False Then
    MsgBox "Database " & strDBName & " is not found."
Else
    Set dsoDB = dsoServer.MDStores(strDBName)

    ' Check for existing data sources, dimensions, and
    ' cubes.
    If dsoDB.DataSources.Count = 0 Then
        ' No data source
        MsgBox "Database " & strDBName & " has no data sources."
    ElseIf dsoDB.Dimensions.Count = 0 Then
        ' No dimensions
        MsgBox "Database " & strDBName & " has no dimensions."
    ElseIf dsoDB.MDStores.Find(strCubeName) Then
        ' Cube already exists
MsgBox "Cube " & strCubeName & " already exists " & _ 
"in database" & strDBName
Else
' Add the cube to the database.
Set dsoCube = dsoDB.MDStores.AddNew(strCubeName)

' Further define the cube.
With dsoCube
' Provide the data source for the cube.
.DataSources.AddNew dsoDB.DataSources(1).Name

' Provide the fact table for the cube.
.SourceTable = ""sales_fact_1997"

' Provide an estimated number of rows.
.EstimatedRows = 100000

' Add the Products and Stores shared dimensions.
.Dimensions.AddNew "Products"
.Dimensions.AddNew "Stores"

' Join the fact table with the dimension tables.
.JoinClause = strJoin

' Update the database.
.Update
End With

' Inform the user.
MsgBox "Cube " & strCubeName & _ 
" created and dimensions added"

End If
End If

End Sub

**Remove a Cube**

The process of removing a cube from a database is performed by the **Remove** method of the database object's **MDStores** collection. The following example code illustrates this by removing the TestCube cube created in the previous code example.

**Example**

The following code example removes the TestCube cube from the **TestDB** database:

Private Sub RemoveCube()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim strDBName As String
    Dim strCubeName As String

    ' Initialize variables for the database and cube names.
    strDBName = "TestDB"
    strCubeName = "TestCube"

    ' Create a connection to the Analysis server.
    dsoServer.Connect "LocalHost"

    ' Ensure the database exists on the server.
    If dsoServer.MDStores.Find(strDBName) = False Then
        MsgBox "Database " & strDBName & " is not found on this server."
    End If
End Sub
Else
Set dsoDB = dsoServer.MDStores(strDBName)

' Ensure the cube exists in the database.
If dsoDB.MDStores.Find(strCubeName) = False Then
    MsgBox "Cube " & strCubeName & " is not found" & _
    " in database " & strDBName & "."
Else
    ' Remove the cube from the database.
dsoDB.MDStores.Remove strCubeName

    ' Inform the user.
    MsgBox "Cube " & strCubeName & " removed" & _
    " from database " & strDBName
End If
End If
End Sub

List Measures

Collections of measures are contained within objects of **ClassType clsCube**, **clsPartition**, and **clsAggregation**. The measure objects contained within each of these collections are **ClassTypes clsCubeMeasure**, **clsPartitonMeasure**, and **clsAggregationMeasure**.

The following code example uses the **Measures** collection of an **MDStore** cube object to list the measures associated with the cubes of each database on a given Analysis server.

Example

The following code example lists each measure of each cube in each database on the local Analysis server, printing basic properties to the Immediate window:

Private Sub ListMeasures()
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore
Dim dsoMea As DSO.Measure

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Step through the databases in the MDStores collection
' of the server object.
For Each dsoDB In dsoServer.MDStores
    Debug.Print "DATABASE: " & dsoDB.Name & " - " & _
    dsoDB.Description

' Step through the cubes in the database collection.
For Each dsoCube In dsoDB.MDStores
    Debug.Print "   Cube: " & dsoCube.Name

    ' Step through measures for the cube.
    For Each dsoMea In dsoCube.Measures
        Debug.Print "     Measure: " & dsoMea.Name
    Next
Next
Next

End Sub

Add Measures

The task of adding measures to an MDStore cube object is performed by the AddNew method of the Measures collection, as illustrated by the following code example.

Example
The following code example adds four measures, **Product ID**, **Store Sales**, **Store Cost**, and **Unit Sales**, to the **TestCube** cube created by previous code examples:

Private Sub AddMeasures()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoMea As DSO.Measure

    Dim strDBName As String
    Dim strCubeName As String

    ' Constants used for ColumnType property
    ' of the DSO.Level object.
    ' Note that these constants are identical to
    ' those used in ADO in the DataTypeEnum enumeration.
    Const adSmallInt = 2

    ' Initialize variables for the database and
cube names.
    strDBName = "TestDB"
    strCubeName = "TestCube"

    ' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

    ' Ensure the database exists first.
    If dsoServer.MDStores.Find(strDBName) = False Then
        MsgBox "Database " & strDBName & " is not found."
    Else
        Set dsoDB = dsoServer.MDStores(strDBName)

        ' Check for existing data sources, dimensions and
' cubes.
If dsoDB.DataSources.Count = 0 Then
' No data source
    MsgBox "Database " & strDBName & " has no data sources."
ElseIf dsoDB.Dimensions.Count = 0 Then
' No dimensions
    MsgBox "Database " & strDBName & " has no dimensions."
ElseIf dsoDB.MDStores.Find(strCubeName) = False Then
' Cube already exists
    MsgBox "Cube " & strCubeName & " does not exist " & _
        "in database" & strDBName
Else
' Add the cube to the database.
    Set dsoCube = dsoDB.MDStores(strCubeName)

    Set dsoMea = dsoCube.Measures.AddNew("Product ID")
    dsoMea.SourceColumn = """"sales_fact_1997"""".""product_id""""
    dsoMea.SourceColumnType = adSmallInt  'The data type for the column
    dsoMea.AggregateFunction = aggSum  'The method for the column
    'aggSum aggregates the column by summation.

    Set dsoMea = dsoCube.Measures.AddNew("Store Sales")
    dsoMea.SourceColumn = """"sales_fact_1997"""".""store_sales""""
    dsoMea.SourceColumnType = adSmallInt
    dsoMea.AggregateFunction = aggSum

    Set dsoMea = dsoCube.Measures.AddNew("Store Cost")
    dsoMea.SourceColumn = """"sales_fact_1997"""".""store_cost""""
    dsoMea.SourceColumnType = adSmallInt
    dsoMea.AggregateFunction = aggSum

    Set dsoMea = dsoCube.Measures.AddNew("Unit Sales")
    dsoMea.SourceColumn = """"sales_fact_1997"""".""unit_sales""""
dsoMea.SourceColumnType = adSmallInt
dsoMea.AggregateFunction = aggSum

dsoCube.Update
End If
End If

End Sub

**Process a Cube**

A database, shared dimensions and their levels, and a cube and its measures are now in place, and the cube can be processed.

To process a cube, use the **Process** method of the **MDStore** cube object as shown in the following code example.

Processing the cube can take several minutes. You can view the cube data using Analysis Manager after processing is complete.

**Example**

The following code example processes the **TestCube** cube created in the previous code examples:

Private Sub ProcessCube()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoMea As DSO.Measure

    Dim strDBName As String
    Dim strCubeName As String

    ' Initialize variables for the database and
    ' cube names.
    strDBName = "TestDB"
strCubeName = "TestCube"

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Ensure the database exists first.
If dsoServer.MDStores.Find(strDBName) = False Then
    MsgBox "Database " & strDBName & " is not found."
Else
    Set dsoDB = dsoServer.MDStores(strDBName)

    ' Check for existing data sources, dimensions, and cubes.
    If dsoDB.DataSources.Count = 0 Then
        ' No data source
        MsgBox "Database " & strDBName & " has no data sources."
    ElseIf dsoDB.MDStores.Find(strCubeName) = False Then
        ' Cube already exists
        MsgBox "Cube " & strCubeName & " does not exist " & _
            "in database" & strDBName
    Else
        ' Retrieve the cube from the database.
        Set dsoCube = dsoDB.MDStores(strCubeName)

        ' Ensure the cube is correctly constructed.
        If dsoCube.Dimensions.Count = 0 Then
            ' No dimensions associated with the cube
            MsgBox "Cube " & strCubeName & _
                " has no dimensions."
        ElseIf dsoCube.Measures.Count = 0 Then
            ' No measures associated with the cube
            MsgBox "Cube " & strCubeName & _
                " has no measures."
Else
  ' Process the cube.
  dsoCube.Process

  ' Inform the user.
  MsgBox "Cube " & strCubeName & " has been processed."
End If
End If
End If
End If

End Sub
Data Mining Examples

The basic example provided in this topic illustrates the common data mining operations, such as the creation of data mining models, that can be performed with Decision Support Objects (DSO). The following table lists the topic covered.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Data Mining Models</td>
<td>Details how to create relational and OLAP data mining models, including the use of mining model roles</td>
</tr>
</tbody>
</table>
Building Data Mining Models

To create a new relational or OLAP data mining model programmatically using Decision Support Objects (DSO), follow these basic steps:

1. Connect to the target Analysis server and select a database from the MDStores collection of the Server object.

2. Create a new data mining model object using the MiningModels collection of the Database object, with the appropriate SubClassType for the relational or OLAP data mining model.

3. If needed, create and assign mining model roles to the new relational or OLAP mining model object.

4. Set the properties needed for the relational or OLAP mining model object. The following table displays the differences between the needed properties for relational and OLAP data mining models.

<table>
<thead>
<tr>
<th>Property</th>
<th>OLAP mining model</th>
<th>Relational mining model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CaseDimension</strong></td>
<td>Defines the case dimension used by the data mining model.</td>
<td>Not used.</td>
</tr>
<tr>
<td><strong>CaseLevel</strong></td>
<td>Defines the case level within the case dimension used by the data mining model. A read-only property, it identifies the lowest level in the dimension whose data mining model column has its</td>
<td>Not used.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>IsDisabled property</td>
<td>set to False.</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Contains a user-friendly description of the data mining model.</td>
<td></td>
</tr>
<tr>
<td>FromClause</td>
<td>Not used. Defines the case table, in the form of a FROM clause, used by the data mining model.</td>
<td></td>
</tr>
<tr>
<td>JoinClause</td>
<td>Not used. Defines any supporting tables, in the form of a JOIN clause, used by the data mining model.</td>
<td></td>
</tr>
<tr>
<td>MiningAlgorithm</td>
<td>Defines the data mining algorithm provider, such as Microsoft_Decision_Trees or Microsoft_Clustering, used by both types of data mining models.</td>
<td></td>
</tr>
<tr>
<td>SourceCube</td>
<td>Defines the OLAP cube used by the data mining model for training data.</td>
<td></td>
</tr>
<tr>
<td>SubClassType</td>
<td>Is set to sbclsOlap when the MiningModel object is created.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Is set to sbclsRelational when the MiningModel object is created.</td>
<td></td>
</tr>
<tr>
<td>TrainingQuery</td>
<td>Defines the Multidimensional Expressions (MDX) query used to insert training data into the data mining model. In most instances, this property is left blank; DSO will construct an appropriate training query if this property is not used.</td>
<td></td>
</tr>
</tbody>
</table>

5. Create a new data mining model column in the **Columns** collection of the **MiningModel** object.
6. Set the properties needed for the new data mining model column. The following table displays the differences between the needed column properties for relational and OLAP data mining models.

<table>
<thead>
<tr>
<th>Property</th>
<th>OLAP mining model</th>
<th>Relational mining model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataType</strong></td>
<td>Defines the expected data type of the data mining column.</td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Contains a user-friendly description of the data mining model column.</td>
<td></td>
</tr>
<tr>
<td><strong>ContentType</strong></td>
<td>Should contain a value from the SUPPORTED_CONTENT_TYPES column of the MINING_SERVICES schema rowset. For example, if the column contained text data that corresponded to income ranges for customers, the <strong>ContentType</strong> property would be set to DISCRETE to reflect the discrete valuations of the data. If, on the other hand, the column contained actual salaries, the property would be set to either CONTINUOUS or DISCRETIZED, depending on the capabilities of the data mining algorithm provider.</td>
<td></td>
</tr>
<tr>
<td><strong>IsKey</strong></td>
<td>Not used. This property is read-only, and is automatically set to True for the lowest enabled level in the case dimension specified in the <strong>CaseDimension</strong> property of the mining model.</td>
<td>Defines the key columns for the data mining model. Set to True to specify a key column in the case set.</td>
</tr>
<tr>
<td><strong>IsInput</strong></td>
<td>Defines the input columns for the data mining model. For a set of related columns, changing the <strong>IsInput</strong> property for one of the columns</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>IsPredictable</td>
<td>Defines the predictable columns for the data mining model. A column can have both <strong>IsInput</strong> and <strong>IsPredictable</strong> set to True. For a set of related columns, changing the <strong>IsPredictable</strong> property for one of the columns automatically changes the property for the other related columns.</td>
<td></td>
</tr>
<tr>
<td>IsDisabled</td>
<td>Defines the columns to be used in analysis for the data mining model.</td>
<td></td>
</tr>
<tr>
<td>Distribution</td>
<td>This property is used to optimize the mining model by giving the mining model algorithm some indication of the statistical nature of the data in the column. The values for this property should come from the <code>SUPPORTED_DISTRIBUTION_FLAGS</code> of the MINING_SERVICES schema rowset.</td>
<td></td>
</tr>
<tr>
<td>SourceOlapObject</td>
<td>The value of this property is an object within the OLAP cube. For instance, this property might contain a DSO level object or a DSO member property object. Not used.</td>
<td></td>
</tr>
<tr>
<td>SourceColumn</td>
<td>Not used.</td>
<td></td>
</tr>
</tbody>
</table>

There are other differences in how column properties are handled.
between OLAP and relational models. For more information about data mining model columns, see clsColumn.

7. Save the mining model object using the Update method.

To optionally train the newly created data mining model, the following additional steps should be used. Although a new data mining model does not need to be processed, the data mining model cannot be browsed until processing is completed.

8. Lock the mining model object using the olapLockProcess flag.

9. Train the mining model object using the Process method.

10. Unlock the mining model object.

Locking the data mining model during processing prevents access by other users until the mining model is unlocked, improving performance during the training of the mining model and ensuring that repository integrity is maintained.

Creating an OLAP Data Mining Model

The following code example creates an OLAP data mining model, following the steps outlined earlier in this topic, that attempts to predict the salary range of a customer in the Sales cube of the FoodMart 2000 database based on gender, marital status and education.

Unlike the process of creating a relational data mining model, the column structure is directly drawn from the source cube specified in the SourceCube property of the mining model object. To determine which columns are to be processed by the data mining model, the column objects stored in the Columns collection of the mining model object can be changed. The IsDisabled property determines which columns are to be used as part of the analysis, while the IsInput and IsPredictive properties of each column object can be set to determine the behavior of the column, including whether it will serve as an input, predictive, or input and predictive column in the data mining model.

Because the structure of the OLAP data mining model is drawn from the structure of the source cube, all source OLAP objects used by the mining model
must be visible to the mining model. The following requirements must be met:

- The source cube must be visible.

- The case dimension must be visible.

- The **SourceOlapObject** property for each data mining column must contain a visible source OLAP object.

**Example**

This code example creates and processes an OLAP data mining model named `CustSalesModelOLAP`, based on the Sales cube of the **FoodMart 2000** database, that analyzes salaries for customers based on gender, marital status and education:

```vba
Public Sub CreateOLAPMiningModel()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoDMM As DSO.MiningModel
    Dim dsoColumn As DSO.Column
    Dim dsoRole As DSO.Role

    ' Constants used for DataType property
    ' of the DSO.Column object.
    ' Note that these constants are identical to
    ' those used in ADO in the DataTypeEnum enumeration.
    Const adInteger = 3
    Const adWChar = 130

    ' Connect to the server on this computer.
    dsoServer.Connect "LocalHost"

    ' Select the FoodMart 2000 database.
```
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Check for the existence of the model on this computer.
If Not dsoDB.MiningModels("CustSalesModelOLAP") Is Nothing Then
    ' If this model exists, delete it.
    dsoDB.MiningModels.Remove "CustSalesModelOLAP"
End If

' Create a new OLAP mining model called CustSalesModelOLAP.
Set dsoDMM = dsoDB.MiningModels.AddNew("CustSalesModelOLAP", sbclsOlap)

' Create a new mining model role called All Users
Set dsoRole = dsoDMM.Roles.AddNew("All Users")

' Set the needed properties for the new mining model.
With dsoDMM
    .DataSources.AddNew "FoodMart", sbclsRegular
    ' Set the description of the model.
    .Description = "Analyzes the salaries of customers"
    ' Select the algorithm provider for the model.
    .MiningAlgorithm = "Microsoft_Decision_Trees"
    ' Set the source cube for the model to the Sales cube.
    .SourceCube = "Sales"
    ' Set the case dimension for the model to the
    ' Customers shared dimension.
    .CaseDimension = "Customers"
    ' Let DSO define the training query.
    .TrainingQuery = ""
    ' Let DSO add the cube structure to the
    ' data mining model structure, automatically
' creating needed data mining model columns.
 .Update
 End With

' Set the column properties pertinent to the new model.
' Note that, when columns are automatically added to
' the model in this fashion, they are disabled. You
' must choose which columns are to be enabled
' before you can process the
' model, and at least one column must be enabled,
' or an error will result.

' Enable the Name column. As this column is the
' lowest enabled level on the Customers case dimension,
' it becomes the case level for the data mining model.
Set dsoColumn = dsoDMM.Columns("Name")
dsoColumn.IsDisabled = False

' Enable the Gender column as an input column.
Set dsoColumn = dsoDMM.Columns("Gender")
dsoColumn.IsInput = True
dsoColumn.IsDisabled = False

' Enable the Marital Status column as an input column.
Set dsoColumn = dsoDMM.Columns("Marital Status")
dsoColumn.IsInput = True
dsoColumn.IsDisabled = False

' Enable the Education column as an input column.
Set dsoColumn = dsoDMM.Columns("Education")
dsoColumn.IsInput = True
dsoColumn.IsDisabled = False
' Enable the Unit Sales column as a predictable column.
Set dsoColumn = dsoDMM.Columns("Yearly Income")
dsoColumn.IsPredictable = True
dsoColumn.IsDisabled = False

' Save the data mining model.
With dsoDMM
  ' Set the LastUpdated property of the new mining model
  ' to the present date and time.
  .LastUpdated = Now
  ' Save the model definition.
  .Update
End With

' Process the data mining model.
With dsoDMM
  ' Lock the mining model for processing
  .LockObject olapLockProcess, _
  "Processing the data mining model in sample code"
  ' Fully process the new mining model.
  ' This may take up to several minutes.
  .Process processFull
  ' Unlock the model after processing is complete.
  .UnlockObject
End With

' Clean up objects and close server connection
Set dsoRole = Nothing
Set dsoColumn = Nothing
Set dsoDMM = Nothing

dsoServer.CloseServer
Set dsoServer = Nothing
Creating a Relational Data Mining Model

The process of creating a relational data mining model is similar to the process of creating an OLAP data mining model, covered earlier in this topic. The major difference between the two, other than the type of data the model will process, is the handling of data mining model columns. Unlike an OLAP data mining model, a relational data mining model does not draw its structure directly from its case and supporting tables. Instead, each column is manually created and defined. The following code example demonstrates the difference in creating a relational data mining model by creating a mining model that duplicates the analysis of the OLAP data mining model created earlier.

This duplication is by design, to give you a direct comparison in structural differences between an OLAP and a relational data mining model.

Example

The following code example creates the **CustSalesModelRel** relational data mining model that analyzes salaries for customers in the **Customer** table based on gender, marital status, and education in the **FoodMart 2000** database:

```vbnet
Public Sub CreateRelMiningModel()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoDS As DSO.DataSource
    Dim dsoDMM As DSO.MiningModel
    Dim dsoColumn As DSO.Column
    Dim dsoRole As DSO.Role

    Dim strLQuote As String, strRQuote As String
    Dim strFromClause As String

    ' Constants used for DataType property
    ' of the DSO.Column object.
```
' Note that these constants are identical to
' those used in ADO in the DataTypeEnum enumeration.
Const adInteger = 3
Const adWChar = 130

' Connect to the server on this computer.
dsoServer.Connect "LocalHost"

' Select the FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Retrieve the open and close quote characters for
' the FoodMart data source.
strLQuote = dsoDB.DataSources("FoodMart").OpenQuoteChar
strRQuote = dsoDB.DataSources("FoodMart").CloseQuoteChar

' The Customer table is the fact table for this
' relational data mining model; this variable will
' make it easier to understand the code that
' follows.
strFromClause = strLQuote & "customer" & strRQuote

' Check for the existence of the model on this computer.
If Not dsoDB.MiningModels("CustSalesModelRel") Is Nothing Then
    ' If this model exists, delete it.
    dsoDB.MiningModels.Remove "CustSalesModelRel"
End If

' Create a new relational mining model
' called CustSalesModelRel.
Set dsoDMM = dsoDB.MiningModels.AddNew("CustSalesModelRel", sbclsRelational)
' Create a new mining model role called All Users
Set dsoRole = dsoDMM.Roles.AddNew("All Users")

' Set the needed properties for the new mining model.
With dsoDMM
  .DataSources.AddNew "FoodMart", sbclsRegular
  ' Set the description of the model.
  .Description = "Analyzes the salaries " & _
    "of customers"
  ' Set the case table for the model to the
  ' Customer table.
  .FromClause = strFromClause
  ' Select the algorithm provider for the model.
  .MiningAlgorithm = "Microsoft_Decision_Trees"
  ' Let DSO define the training query.
  .TrainingQuery = ""
  ' Save the existing structure.
  .Update
End With

' Create the columns pertinent to the new model.

' Create the CustomerID column as a key column.
Set dsoColumn = dsoDMM.Columns.AddNew("CustomerID", _
  sbclsRegular)
' Set the column properties for the new column.
With dsoColumn
  ' Set the source field from the case table for
  ' the column.
  .SourceColumn = strFromClause & "." & strLQuote & _
    "customer_id" & strRQuote
  .DataType = adInteger
  .IsKey = True
.IsDisabled = False
End With

' Create the Gender column as an attribute column.
Set dsoColumn = dsoDMM.Columns.AddNew("Gender", _
    sbclsRegular)
With dsoColumn
    .ContentType = "DISCRETE"
    .SourceColumn = strFromClause & "." & strLQuote & _
        "gender" & strRQuote
    .DataType = adWChar
    .IsDisabled = False
End With

' Create the Marital Status column as an attribute column.
Set dsoColumn = dsoDMM.Columns.AddNew("Marital Status", _
    sbclsRegular)
With dsoColumn
    .ContentType = "DISCRETE"
    .SourceColumn = strFromClause & "." & strLQuote & _
        "marital_status" & strRQuote
    .DataType = adWChar
    .IsDisabled = False
End With

' Create the Education column as an attribute column.
Set dsoColumn = dsoDMM.Columns.AddNew("Education", _
    sbclsRegular)
With dsoColumn
    .ContentType = "DISCRETE"
    .SourceColumn = strFromClause & "." & strLQuote & _
        "education" & strRQuote
    .DataType = adWChar
Create the Yearly Income column as an predictable column.
Set dsoColumn = dsoDMM.Columns.AddNew("Yearly Income", _
    sbclsRegular)
With dsoColumn
    .ContentType = "DISCRETE"
    .SourceColumn = strFromClause & "." & strLQuote & 
        "yearly_income" & strRQuote
    .DataType = adWChar
    .IsInput = False
    .IsPredictable = True
    .IsDisabled = False
End With

Save the data mining model.
With dsoDMM
    ' Set the LastUpdated property of the new mining model
    ' to the present date and time.
    .LastUpdated = Now
    ' Save the model definition.
    .Update
End With

Process the data mining model.
With dsoDMM
    ' Lock the mining model for processing
    .LockObject olapLockProcess, _
        "Processing the data mining model in sample code"
    ' Fully process the new mining model.
    ' This may take up to several minutes.
    .Process processFull
' Unlock the model after processing is complete.
  .UnlockObject
End With

' Clean up objects and close server connection
Set dsoRole = Nothing
Set dsoColumn = Nothing
Set dsoDMM = Nothing

dsoServer.CloseServer
Set dsoServer = Nothing

End Sub
Advanced Examples

The examples in this topic assume that you are familiar with using Decision Support Objects (DSO) to create databases, data sources, dimensions and cubes. These advanced examples include the creation of virtual and linked cubes and virtual dimensions, as well as the capability to perform incremental updates on a cube. The following table lists the examples covered.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Working with Virtual Cubes</strong></td>
<td>Gives information and examples on creating virtual cubes in DSO</td>
</tr>
<tr>
<td><strong>Working with Linked Cubes</strong></td>
<td>Provides information and examples in DSO on creating linked cubes</td>
</tr>
<tr>
<td><strong>Working with Virtual Dimensions</strong></td>
<td>Describes the creation of virtual dimensions in DSO</td>
</tr>
<tr>
<td><strong>Working with Roles</strong></td>
<td>Details the differences between database, cube, and mining model roles, providing examples on the creation of roles in DSO</td>
</tr>
<tr>
<td><strong>Incremental Updates</strong></td>
<td>Provides information and examples on processing incremental updates in DSO</td>
</tr>
</tbody>
</table>
Working with Virtual Cubes

A cube object with a SubClassType of **sbclsVirtual** is a virtual cube. A virtual cube is used to encapsulate a subset of the measures, dimensions, and levels contained in one or more cubes. A virtual cube, like a view in a relational database, is a logical construct that itself contains no data. Just as a view is a join of multiple relations, a virtual cube is a join of multiple cubes.

The basic rule for using virtual cubes is that you add them to a database as a cube with the SubClassType parameter set to **sbclsVirtual**, and then add dimensions and measures to them as needed. However, the dimensions and measures are derived from previously defined cubes within the database rather than from a dimension table. Any levels associated with a dimension that has been added to a virtual cube automatically apply to the dimension in the virtual cube. Partitions and aggregations do not apply to virtual cubes.

If the structure for a virtual cube is changed, you must reprocess the virtual cube so that, just as with a regular cube, the data supporting the structure change can be reprocessed. The same holds true if you alter the structure of a regular cube used by a virtual cube; the regular cube needs to be reprocessed, and then the virtual cube also needs to be reprocessed.

Source Cubes and Source Dimensions

A virtual cube can contain any number of source cubes, including linked cubes, as long as they are from the same database.

Virtual cubes do not inherit the roles, calculated members, or actions that are assigned to their source cubes. After a virtual cube has been created, you must re-create these objects (or design different ones). The information needed to re-create the roles, calculated members or actions can be derived by reading the structures of the underlying regular cubes.

Other shared dimensions from the database but not from an included cube are also acceptable in a virtual cube. These dimensions require custom rollup expressions on their levels. Without the custom expressions, the server will not be able to find the dimension's data because the dimension references columns
are not in the fact table.

**Differences of Virtual Cubes**

Because a virtual cube is based on the contents of one or more existing cubes, some of the properties and collections for a virtual cube object are not available, or they have a different meaning from their counterparts in a regular cube. An attempt to set or retrieve an unavailable property results in an error. The following table lists the properties and collections that are different for virtual cubes.

<table>
<thead>
<tr>
<th>Property or collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregationPrefix</td>
<td>A virtual cube does not use aggregation prefixes.</td>
</tr>
<tr>
<td>Analyzer</td>
<td>A virtual cube does not have an analyzer object.</td>
</tr>
<tr>
<td>DataSources</td>
<td>A virtual cube does not have a DataSources collection.</td>
</tr>
<tr>
<td>EstimatedRows</td>
<td>For a virtual cube, this property is read-only and contains the number of rows in all underlying cubes.</td>
</tr>
<tr>
<td>FromClause</td>
<td>A virtual cube does not have a FROM clause.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>A virtual cube does not have a JOIN clause.</td>
</tr>
<tr>
<td>MDStores</td>
<td>For a virtual cube, this collection contains the underlying cubes instead of the cube partitions.</td>
</tr>
<tr>
<td>OlapMode</td>
<td>A virtual cube does not use the OlapMode property.</td>
</tr>
<tr>
<td>SourceTable</td>
<td>A virtual cube does not have its own fact table.</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>A virtual cube does not have its own fact table.</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>A virtual cube does not have its own fact table.</td>
</tr>
<tr>
<td>Dimension.DataSource</td>
<td>The dimensions in a virtual cube do not have data sources.</td>
</tr>
</tbody>
</table>

**Add a Virtual Cube**

The process of adding a virtual cube is largely the same as the process of adding a regular cube. There are minor differences as noted in the table and as illustrated in the following code example.
Example

Note The following example depends on the existence of the TestCube regular cube, created in previous code examples.

The following code example creates a new virtual cube, named VirtualCube, based on the TestCube cube in the TestDB database, created in earlier code examples:

Private Sub AddVirtualCube()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoDim As DSO.Dimension
    Dim dsoMea As DSO.Measure

    Dim strDBName As String
    Dim strCubeName As String

    ' Initialize variables for the database and
    ' virtual cube names.
    strDBName = "TestDB"
    strCubeName = "VirtualCube"

    ' Create a connection to the Analysis server.
    dsoServer.Connect "LocalHost"

    If dsoServer.MDStores.Find(strDBName) = False Then
        MsgBox "Database " & strDBName & " is not found."
    Else
        ' Retrieve the database from the server.
        Set dsoDB = dsoServer.MDStores(strDBName)

        ' Create the virtual cube in the MDStores collection
Set dsoCube = dsoDB.MDStores.AddNew(strCubeName, sbclsVi

' Set the description for the virtual cube.
dsoCube.Description = "The TestDB virtual cube"

' Set the source dimensions for the virtual cube by
' copying the dimensions from the underlying cube.
For Each dsoDim In dsoDB.MDStores("TestCube").Dimensions
  dsoCube.Dimensions.AddNew dsoDim.Name
Next

' Add measures to the virtual cube from the underlying cube.
' Measures for virtual cubes have the format
' [Cube Name].[Measure Name]

' Create the Unit Sales measure.
Set dsoMea = dsoCube.Measures.AddNew("Unit Sales")
dsoMea.SourceColumn = ".[Unit Sales]"

' Create the Store Sales measure.
Set dsoMea = dsoCube.Measures.AddNew("Store Sales")
dsoMea.SourceColumn = ".[Store Sales]"

' Save cube structure changes.
dsoCube.Update

' Process the cube so that it can be used by client applications.
dsoCube.Process

End If

End Sub
See Also

Virtual Cubes

Working with Dimensions and Levels

Working with Cubes and Measures
Analysis Services Programming

**Working with Linked Cubes**

A linked cube is an `MDStore` cube object with a `SubClassType` of `sbclsLinked`. The contents of a linked cube are based on another cube that is defined and stored on a different Analysis server. Unlike a virtual cube, which can contain portions of one or more cubes, a linked cube references the entire contents of a single cube.

A subscribing server is an Analysis server that contains a linked cube. A publishing server contains the source cube upon which the linked cube is based. To be a subscribing server, the Analysis server service (MSSQLServerOLAPService) must run under an account that has query permissions on each publishing server to which it connects. This account can be an account that belongs to the OLAP Administrators group on the publishing server, or an account that has query permissions established by a role on each source cube to which the subscribing server needs access. There are no requirements an Analysis server has to meet in order to become a publisher. Any processed cube on the publishing server can be made available for linking, subject to network and cube security; the cube must be available for use by the subscribing server as if the subscribing server were a client querying the cube on the publishing server.

All dimensions in a linked cube are treated as private dimensions on the subscribing server. This means that other regular cubes in the subscribing database cannot use these dimensions. A linked cube can be included in a virtual cube.

Linked cubes cannot be created from regular cubes that employ shared or private ROLAP dimensions.

**Differences of Linked Cubes**

Because a linked cube is based on the contents of an existing cube, some of the properties for the linked cube object are not supported, or they have a meaning that is different from their counterparts in a regular cube. An attempt to set or retrieve an unsupported property results in an error. The properties that are
different for linked cubes are listed in the following table.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube.OlapMode</td>
<td>Read-only. It is taken from the published cube.</td>
</tr>
<tr>
<td>Dimension.SubClassType</td>
<td>Always sbclsLinked for a dimension in a linked cube.</td>
</tr>
<tr>
<td>Measure.AggregateFunction</td>
<td>Read-only. It is taken from the measure in the published cube.</td>
</tr>
<tr>
<td>Measure.ColumnType</td>
<td>Read-only. It is taken from the measure in the published cube.</td>
</tr>
<tr>
<td>Partition.OlapMode</td>
<td>Always olapmodeROLAP for a partition in a linked cube.</td>
</tr>
</tbody>
</table>

**Add a Linked Cube**

The following code example illustrates the steps needed to create a linked cube.

**Note** This procedure must involve two different servers: a publishing server and a subscribing server. Attempting to create a link to a cube on the same server causes an error.

**Example**

The following code example creates a linked cube by joining the Sales cube from the **FoodMart 2000** database on the publishing server to a new cube, named LinkedCube, to the **TestDB** database on the subscribing server:

```vbscript
Private Sub AddLinkedCube()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoLDS As DSO.DataSource

    Dim strDBName As String
    Dim strCubeName As String
    Dim strServerName As String
```
Initialize variables for the database and linked cube names.
strDBName = "TestDB"
strCubeName = "LinkedCube"

The following variable should be set to the name of the publishing server.
strServerName = "servername"

Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

If dsoServer.MDStores.Find(strDBName) = False Then
    MsgBox "Database " & strDBName & " is not found."
Else
    ' Get a reference for the database that will contain the linked cube.
    Set dsoDB = dsoServer.MDStores(strDBName)

    ' Create a new data source for the linked cube.
    Set dsoLDS = dsoDB.DataSources.AddNew("PublishingServer")

    ' Set the connection string, so that the data source points to an Analysis server running SQL Server 2000 Analysis Service or later, installed with the FoodMart 2000 database.
    dsoLDS.ConnectionString = "Provider=MSOLAP;" & _
        "Data Source=" & strServerName & ";" & _
        "Initial Catalog=FoodMart 2000;"

    ' Save this data source in the repository.
    dsoLDS.Update
Create a new cube on the local server, mark it as linked.
Set dsoCube = dsoDB.MDStores.AddNew(strCubeName, sbclsLinked)

Add dsoLDS to the DataSources collection of the linked cube.
dsoCube.DataSources.Add dsoLDS

Use the name of the published cube as the
' source table for the subscribed cube.
dsoCube.SourceTable = """" & "Sales" & """

Update the cube. This creates the link.
dsoCube.Update

Commit the changes to the subscribing server.
dsoCube.Process processFull
End If

End Sub

See Also

Linked Cubes
Working with Virtual Dimensions

The steps for creating virtual dimensions with Decision Support Objects (DSO) are similar to those used to create regular dimensions. To create a virtual dimension based on the columns of another dimension, create the dimension normally, but set the `IsVirtual` property to True and set the `DependsOnDimension` property to the name of the source dimension. Creating a virtual dimension based on the member properties of a regular dimension is more complicated. The procedure is outlined in the code sample at the end of this topic.

Differences of Virtual Dimensions

Because a virtual dimension is based on the contents of an existing dimension, many of the properties for the virtual dimension object and its level objects are read-only and do not need to be set before the dimension is processed. The remaining properties for the dimension and level objects must be set to refer to the underlying dimension and/or member properties that provide the source data for the virtual dimension.

The following table lists dimension and level properties that are read-only or ignored for virtual dimensions.

<table>
<thead>
<tr>
<th>Object property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Dimension.FromClause</code></td>
<td>Read-only. It is taken from the source dimension.</td>
</tr>
<tr>
<td><code>Dimension.JoinClause</code></td>
<td>Read-only. It is taken from the source dimension.</td>
</tr>
<tr>
<td><code>Dimension.StorageMode</code></td>
<td>Always <code>storeasMOLAP</code> for a virtual dimension.</td>
</tr>
<tr>
<td><code>Dimension.SourceTableFilter</code></td>
<td>Read-only. It is taken from the source dimension.</td>
</tr>
<tr>
<td><strong>Dimension.SourceTableAlias</strong></td>
<td>Read-only. It is taken from the source dimension.</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>Level.EstimatedSize</strong></td>
<td>Not used for a level in a virtual dimension.</td>
</tr>
<tr>
<td><strong>Level.Grouping</strong></td>
<td>Always <strong>groupingNone</strong> for a level in a virtual dimension.</td>
</tr>
<tr>
<td><strong>Level.HideMemberIf</strong></td>
<td>Always <strong>hideNever</strong> for a level in a virtual dimension.</td>
</tr>
</tbody>
</table>

**Add a Virtual Dimension**

Use the following code example to create a virtual dimension. The virtual dimension, except as noted in the table, is treated as any other dimension.

**Example**

The following code example creates the Store Size in SQFT virtual dimension in the **TestDB** database. This virtual dimension is based on a member property, Store SQFT, of the Stores source dimension:

```vbscript
Private Sub AddVirtualDimension()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoDS As DSO.DataSource
    Dim dsoDim As DSO.Dimension
    Dim dsoLevel As DSO.Level

    Dim strDBName As String
    Dim strLQuote As String
    Dim strRQuote As String

    ' Define constants used for the ColumnType property
    ' of the DSO.Level object.
    ' Note that these constants are identical to
    ' those used in ADO in the DataTypeEnum enumeration.
    Const adDouble = 5
```
' Initialize variable for the database.
strDBName = "TestDB"

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Ensure that the server has an existing database.
If dsoServer.MDStores.Find(strDBName) = False Then
    MsgBox "Database " & strDBName & " is not found."
Else
    ' Retrieve the database from the server.
    Set dsoDB = dsoServer.MDStores(strDBName)

    ' Retrieve a data source from the database.
    Set dsoDS = dsoDB.DataSources("FoodMart")

    ' Get the delimiter characters from the data source.
    strLQuote = dsoDS.OpenQuoteChar
    strRQuote = dsoDS.CloseQuoteChar

    ' Create the new dimension in the Dimensions collection of the database object.
    Set dsoDim = dsoDB.Dimensions.AddNew("Store Size in SQFT")

    ' Set the description of the dimension.
    dsoDim.Description = "The Store Size in SQFT virtual dimension"

    ' Set the data source of the dimension.
    Set dsoDim.DataSource = dsoDS

    ' Set the dimension type, make it virtual,
' and identify its underlying source dimension.
dsoDim.DimensionType = dimRegular
dsoDim.IsVirtual = True
dsoDim.DependsOnDimension = "Stores"

' Next, create the levels.
' Start with the (All) level.
Set dsoLevel = dsoDim.Levels.AddNew("(All)")

' Set the level type.
dsoLevel.LevelType = levAll

' Set the MemberKeyColumn of the (All) level to a constant
' that also acts as the name of the level's only member.
dsoLevel.MemberKeyColumn = "(All Store Sizes)"

' Create the Store SQFT level. This holds the SQFT value.
Set dsoLevel = dsoDim.Levels.AddNew("Store Size")

' Name the source column for this level.
' The format for this is "table_name"."column_name".
' Database-specific delimiter characters are required.
dsoLevel.MemberKeyColumn = strLQuote & "store" & strRQuote & _
strLQuote & "store_sqft" & strRQuote

' Set the following properties to be identical to their
' counterparts in the member property object that provides
' this level with its data.
dsoLevel.ColumnType = adDouble
dsoLevel.ColumnSize = 4

' Check to see that you set the level and
' dimension properties correctly, and that the rest
of the dimension structure is correct. If so, update the repository and exit the function.
If dsoLevel.IsValid And dsoDim.IsValid Then
    ' Update the dimension.
    dsoDim.Update

    ' Inform the user.
    MsgBox "Virtual dimension has been added."
End If
End If

End Sub

See Also

Virtual Dimensions
Working with Roles

Roles are used in Microsoft® SQL Server™ 2000 Analysis Services to provide security for databases, cubes, and mining models. Decision Support Objects (DSO) provides the Role object for administering all three types of roles.

Database Roles

A database role applies to a single Analysis Services database, and it includes a list of Microsoft Windows NT® 4.0 or Microsoft Windows® 2000 user accounts and groups. A database role does not control administrative access to an Analysis Services object; instead it determines read and write capabilities when a user is connected to an Analysis Services database through a client application. Database roles can be used to manage the dimension security for shared dimensions in a database for multiple cubes. For more information about dimension security, see Dimension Security.

When a database role is assigned to a cube or mining model, Analysis Services creates a corresponding cube or mining model role. The property values specified in the database role are then propagated to the newly created cube or mining model role. A cube or mining model role cannot exist without a corresponding database role.

If the property value of a database role is changed, any cube or mining model role based on that database role is also changed, but only if the cube or mining model role still uses the value propagated from the database role. In other words, if you change a cube or mining model role property directly, changing the corresponding database role property does not override the changed cube or mining model role property.

For more information about database roles, see Database Roles.

A database role is represented in DSO by a Role object of ClassType clsDatabaseRole. The available properties for the Role object are different for each type of role, and the behavior of some properties changes as well. For more information about the Role object, see Role Interface.
Use the following example to create a new database role. The database role created in the example will be used by the other examples in this topic.

**Example**

This code example creates a new database role, named TestRole, in the **FoodMart 2000** database:

Private Sub CreateDatabaseRole()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoRole As DSO.Role

    Dim sDimensionSecurity As String

    ' Connect to the local server.
    dsoServer.Connect "LocalHost"

    ' Connect to the FoodMart 2000 database.
    Set dsoDB = dsoServer.MDStores.Item("FoodMart 2000")

    ' Create a new database role named TestRole.
    Set dsoRole = dsoDB.Roles.AddNew("TestRole", sbclsRegular)

    ' Create the XML syntax to be used for the SetPermissions method of the Role object.
    sDimensionSecurity = "<MEMBERSECURITY " & _
        "DefaultMember=""[Store].[Store Country].&[USA]"" " & _
        "VisualTotalsLowestLevel=""[Store].[Store City]"">" & _
        "<PERMISSION Access=""Read"" " & _
        "DeniedSet=""{[Store].[Store Country].&[Canada],[Store].[Store Country].&[Mexico]}"" " & _
        "Description=""USA Store Restriction""/>" & _
"</MEMBERSECURITY>"

' The preceding XML syntax limits the users of the database role
to viewing only stores in the USA, by denying read access to
stores associated with the Mexico and Canada members of the
[Store Country] level of the Stores dimension.

' Change the role properties for TestRole
With dsoRole
  ' Lock the database role.
  .LockObject olapLockRead, "Creating Role"

  ' Set the list of users assigned to this role.
  .UsersList = "Everyone"

  ' Set the role description.
  .Description = "Test role"

  ' Set the EnforcementLocation permission key to enforce
  ' the role on the server side.
  .SetPermissions "EnforcementLocation", "Server"

  ' Set the Dimension key for the Store dimension to
  ' restrict users to viewing only USA stores.
  .SetPermissions "Dimension:Store", sDimensionSecurity

  ' Unlock the database role.
  .UnlockObject
End With

' Update the database role.
dsoRole.Update
A database role, as a major object, can update itself using the Update method as shown in the preceding code example. The code example also features the use of Extensible Markup Language (XML) to set dimension security with the SetPermissions method of the Role object. For more information about the XML syntax of the Dimension key for the SetPermissions method, see SetPermissions.

Cube Roles

A cube role applies to a single cube in an Analysis Services database, includes a list of Windows NT 4.0 or Windows 2000 user accounts and groups, and indicates the objects in the cube those accounts can access and the kind of access they have to those objects. The cube role is based on a database role, but it supplies additional security measures for restricting the viewing of cells within a cube as well. For more information about cell security, see Cell Security.

Properties in a cube role can be changed to reflect different security options for a single cube. However, changes to some specifications in a cube role propagate to the corresponding database role and all cube roles with the same name as the changed cube role. These specifications include the list of user accounts and groups and read/write permissions for dimensions.

For more information about cube roles, see Cube Roles.

A cube role is represented in DSO by a Role object of ClassType clsCubeRole. The following example creates a new cube role for the TestCube cube, using the new TestRole database role created in the previous code example.

Example

The TestRole cube role is created in the Sales cube, and receives its default property values from the TestRole database role created in the FoodMart 2000 database.

Private Sub CreateCubeRole()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore
Dim dsoRole As DSO.Role

' Connect to the local server.
dsoServer.Connect "LocalHost"

' Connect to the TestDB database.
Set dsoDB = dsoServer.MDStores.Item("FoodMart 2000")

' Connect to the TestCube cube.
Set dsoCube = dsoDB.MDStores("Sales")

' As the cube is about to be changed, lock the cube.
dsoCube.LockObject olapLockRead

' Create a new cube role named TestRole, based on the database role
' named TestRole.
Set dsoRole = dsoCube.Roles.AddNew("TestRole", sbclsRegular)

' Change the role properties for the cube role.
With dsoRole
' All of the other properties are propagated from the
' TestRole database role.

' Prevent the users associated with this role from
' reading the [Store Cost] measure.
.SetPermissions "CubeRead", _
"Measures.CurrentMember.Name <> "]Store Cost]"
End With

' Update the cube role by updating the cube.
dsoCube.Update
'Unlock the cube.
dsoCube.UnlockObject

End Sub

Because it is a minor object, a cube role cannot update itself using the **Update** method. A cube role is saved when the cube to which it is associated is updated, as demonstrated by the preceding code example. The code example also demonstrates the use of Multidimensional Expressions (MDX) syntax to set cell security with the **SetPermissions** method of the **Role** object. For more information about the use of MDX syntax of the **CellRead** key for the **SetPermissions** method, see [SetPermissions](#).

**Mining Model Roles**

A mining model role applies to a single data mining model and includes a list of Windows NT 4.0 or Windows 2000 user accounts and groups that have access to the data mining model.

A mining model role is similar in many respects to a cube role. The user membership of a mining model role is directly related to its corresponding database role; changes to role membership in a mining model role propagate to the database role and all mining model roles with the same name as the changed mining model role.

For more information about mining model roles, see [Mining Model Roles](#).

A mining model role is represented in DSO by a **Role** object of **ClassType clsMiningModelRole**. The following code example creates a mining model role, also based on the TestRole database role created in an earlier code example.

**Example**

This code example creates a new mining model role, named TestRole, in the Member Card RDBMS mining model of the **FoodMart 2000** database.

Private Sub CreateMiningModelRole()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
Dim dsoDMM As DSO.MiningModel
Dim dsoRole As DSO.Role

' Connect to the local server.
dsoServer.Connect "LocalHost"

' Connect to the TestDB database.
Set dsoDB = dsoServer.MDStores.Item("FoodMart 2000")

' Connect to the Member Card RDBMS mining model.
Set dsoDMM = dsoDB.MiningModels("Member Card RDBMS")

' Because the mining model is about to be changed, lock the mining model.
dsoDMM.LockObject olapLockProcess, "Changing mining model"

' Create a new mining model role named TestRole, based on the database role named TestRole.
Set dsoRole = dsoDMM.Roles.AddNew("TestRole", sbclsRegular)
' All of the other properties are propagated, such as the user list, from the TestRole database role.

' Update the mining model role by updating the mining model.
dsoDMM.Update

' Unlock the mining model.
dsoDMM.UnlockObject

End Sub

A mining model role, like a cube role, is a minor object. A mining model role cannot update itself using the Update method; the role is written to the database when its associated data mining model is updated.
Incremental Updates

Incremental updates allow you to keep the contents of a cube current without requiring you to reprocess the cube in full when you add new data. An incremental update involves creating a temporary partition, filling it with updated source data, processing the temporary partition, and then merging it into another partition in the cube.

Data to be added to a cube can come from the original fact table or from a separate fact table with a structure identical to the original. If you add data from the original fact table, take care not to duplicate data that already exists in the cube. Set the `SourceTableFilter` property before processing the created temporary partition to restrict the data that is imported from the fact table. Temporary partitions created for this purpose are indicated by a tilde (~) character preceding the name of the temporary partition.

If data to be added to a cube comes from the fact table from which the cube was originally created, a risk of duplicate aggregation occurs. The cube uses the `SourceTableFilter` property to screen incoming data from a fact table; when performing an incremental update, it adds the aggregations computed from the fact table to the aggregations stored by the cube. If the same table is run twice, once to construct the original aggregations, and again as part of an incremental update, you will receive the same data twice, added together in the cube. The `SourceTableFilter` property can be used to screen out existing, already aggregated data in the fact table, preventing duplicate aggregation.

If you add data from a fact table that includes new members of a dimension, you must also reprocess the affected dimension using the `processRefreshData` for the `Process` method of the dimension object.

Perform an Incremental Update

The following code example shows how to do an incremental update using a temporary partition based on a separate fact table.

Example
The following code example performs an incremental update on the TestCube cube of the **TestDB** database by creating a temporary partition, adding a new table, and combining the partitions:

Private Sub IncrementalUpdate()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoPartition As DSO.MDStore
    Dim dsoClonePartition As DSO.MDStore
    Dim dsoMeasure As DSO.Measure

    ' Connect to the local server.
    dsoServer.Connect "LocalHost"

    ' Set up the MDStore objects:
    ' database, cube, and partition.
    Set dsoDB = dsoServer.MDStores.Item("TestDB")
    Set dsoCube = dsoDB.MDStores.Item("TestCube")
    Set dsoPartition = dsoCube.MDStores.Item("TestCube")

    ' Create a temporary partition to store the new data.
    ' Use the tilde character to indicate to the server
    ' that the partition is not permanent.
    Set dsoClonePartition = dsoCube.MDStores.AddNew("~New TestCube")

    ' Clone the main partition to the temporary one.
    dsoPartition.Clone dsoClonePartition, cloneMinorChildren

    ' Because this partition uses a different source table,
    ' the properties that identify the table must be changed
    ' If this update involved the original fact table, these
    ' properties would remain unchanged, but the SourceTableFilter
' property would have to be updated to prevent duplicate data
' from being imported during processing.
dsoClonePartition.SourceTable = "sales_fact_1998"
dsoClonePartition.FromClause =_
    Replace(dsoClonePartition.FromClause, "1997", "1998")
dsoClonePartition.JoinClause =_
    Replace(dsoClonePartition.JoinClause, "1997", "1998")

' It is also necessary to update the measures in the
' partition, because they reference the old fact table, too.
For Each dsoMeasure In dsoClonePartition.Measures
dsoMeasure.SourceColumn =_
    Replace(dsoMeasure.SourceColumn, "1997", "1998")
Next

' Although this is not the case in this example, if the
' MemberKeyColumn or MemberNameColumn properties of any of
' dimension levels are based on the fact table, they too must
' be updated.

' Process the temporary partition.
dsoClonePartition.Process processDefault

' Merge the two partitions.
dsoPartition.Merge "~New TestCube Data"

' Close the server object and exit the subroutine.
dsoServer.CloseServer

End Sub

See Also
Merging Partitions

Merge
Analysis Services Programming
# Additional Considerations

This topic contains additional information, examples, and suggestions to help you use Decision Support Objects (DSO) more effectively.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Considerations for Naming Decision Support Objects</td>
<td>Covers the naming conventions for DSO objects, and discusses the importance of unique object names</td>
</tr>
<tr>
<td>Object Locking with Decision Support Objects</td>
<td>Discusses the process of object locking in DSO in multiuser situations</td>
</tr>
<tr>
<td>Tips for Creating Member Properties in Multiple Languages</td>
<td>Explains the support for member properties to handle multiple language requirements</td>
</tr>
<tr>
<td>Using Earlier Versions of Analysis Services</td>
<td>Discusses the use of previous versions of Microsoft® SQL Server™ 2000 Analysis Services with the current version of the DSO library</td>
</tr>
</tbody>
</table>
Considerations For Naming Decision Support Objects

When naming Decision Support Objects (DSO) objects, you should follow a consistent naming convention and ensure that the name of each object is unique. Most of the naming conventions supplied in this topic are optional; some are required, as in the cases of virtual and private dimensions. The following naming convention assists in understanding the sometimes complex hierarchies formed by the DSO object model; the use of unique DSO object names also speeds performance.

DSO Object Naming Requirements

When you create a DSO object in an application, you must set a value for the object's Name property to differentiate it from similar objects in use and/or stored on the Analysis server. To prevent errors during execution time caused by characters that are not valid, follow these guidelines when you name objects:

- All names must begin with a letter, with the exception of virtual dimensions. The name of a virtual dimension must start with a tilde (~) character.

- A dimension can contain a single period (.) in its name if it contains multiple hierarchies. This period serves to separate the dimension name from the hierarchy name. (For example, consider MyDim.Hier1 and MyDim.Hier2, where MyDim is a dimension with two hierarchies, named Hier1 and Hier2.) Private dimensions must contain the cube name followed by a caret (^) character and the dimension name.

- Avoid most symbol characters. Some objects have specific limitations regarding nonalphanumerics, while other objects supply meaning to certain nonalphanumeric characters, such as the tilde (~) and caret (^) characters. The following table lists characters that are not allowed.
The following reserved names should not be used for DSO objects:

- AUX
- CLOCK$
- COM1 through COM9 (COM1, COM2, COM3, and so on)
- CON
- LPT1 through LPT9 (LPT1, LPT2, LPT3, and so on)
- NUL
- PRN

<table>
<thead>
<tr>
<th>Object</th>
<th>Invalid characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>The name must follow the rules for Microsoft® Windows NT® 4.0 and Windows® 2000 computer names. (IP addresses are not valid.)</td>
</tr>
<tr>
<td>Data source</td>
<td>: / \ *</td>
</tr>
<tr>
<td>Level</td>
<td>. , ; ` : / \ *</td>
</tr>
<tr>
<td>Dimension</td>
<td>. , ; ` : / \ *</td>
</tr>
<tr>
<td>All other objects</td>
<td>. , ; ` : / \ *</td>
</tr>
</tbody>
</table>

**The Importance of Unique Names**

Object names in Multidimensional Expressions (MDX) queries are resolved in a specific order. For best results with cube speed and accuracy, make the effort to use unique names for all objects you create in a database on an Analysis server. If using unique names is not an option, make an effort to qualify names as
completely as possible in your queries, especially in cases where identical names appear in different dimensions and levels. The following paragraphs outline the order in which name conflicts in MDX statements are resolved.

When matching names to cube objects in an MDX query, the Analysis server first tries to match the initial portion of the name to a dimension, then a level, and finally, a member. When the server is satisfied that it has located one of these objects, it then uses the final element in the name to search within the bound object. For example, suppose there is a dimension [D1], a level [L1], and a member [M]. The statement [D1].[L1].[M] is broken down and [D1].[L1] is bound to the level. The server then searches the level for the member [M].

If a level in a dimension has a name identical to another dimension that is not its parent, that level will not be searched by a poorly constructed query. For example, suppose there are two dimensions [D1] and [D2]. [D2] has a level named [D1]. If a query refers to a member as [D1].[M], the Analysis server binds the name [D1] to the dimension and searches for [M] there. If it cannot find [M] in [D1], the query fails (once the server has bound a name to an object, it does not continue to the next object in the collection if the search fails). For this type of query to succeed, it should include the complete hierarchy of the dimension to locate the member: [D2].[D1].[M].

The same rules apply to members with children. Suppose there is a dimension [D1] with both a level and a member named [L1]. The member [L1] also has a child [C]. In an attempt to reference [C], [D1].[L1].[C] fails because the server binds [D1].[L1] to the level and searches for [C] as a member. For this type of query to succeed, it should include the complete hierarchy of the dimension to locate the child: [D1].[L1].[L1].[C].

A first-fit algorithm solves ambiguities in member names. If a member is referred to as [M] (without a corresponding parent dimension), the server searches the dimensions in the order they are listed in the cube's Dimensions collection until it finds the member. Although this can help resolve ambiguous member names, this process is slow and can affect performance. If the member is located in two different levels of the same dimension, the server returns the member closest to the root of the dimension tree.
Analysis Services Programming

**Object Locking with Decision Support Objects**

If you are developing applications for use in a multiple-administrator environment, you should become familiar with the `LockObject` and `UnlockObject` methods. These methods provide for repository stability when changes are made to Microsoft® SQL Server™ 2000 Analysis Services objects. When one application obtains a lock on an object, the options available to other applications are restricted until the lock is released. The use of object locking can provide performance benefits when working with Decision Support Objects (DSO) objects, and it can forestall some errors in multiple user object access situations.

When an application disconnects from the server, all locks it left in place are automatically released. This prevents objects from being locked indefinitely from unexpected circumstances.

The types of locks defined by the `OlapLockTypes` enumeration are: `olapLockExtendedRead`, `olapLockProcess`, `olapLockRead`, and `olapLockWrite`.

**olapLockExtendedRead**

The object's properties can be read by other applications, but cannot be changed or processed. This lock is used to prevent processing of dependent objects of an object that is being processed, such as dimensions that are shared by multiple cubes. Multiple `olapLockExtendedRead` locks may be applied to an object by multiple applications. However, no application can lock the object for processing or writing until all `olapLockExtendedRead` locks have been released.

**olapLockProcess**

The object's `Process` method can be called and other applications can read the object's properties only until the lock is released. Only one `olapLockProcess` lock can be applied to an object at a time, and other applications can apply `olapLockRead` locks only while the `olapLockProcess` lock is in place.
olapLockRead

The properties of the object can be read from the repository and cannot be changed by another application until the lock is released. Other applications can issue olapLockRead, olapLockExtendedRead, and olapLockProcess locks, but not olapLockWrite locks, while the initial olapLockRead lock is in place.

olapLockWrite

The properties of the object can be modified in the repository using the Update method, and are not available to other applications for any use until the lock is released. No other locks of any type can be applied to the object by another DSO client until the olapLockWrite lock is released.

The different lock types are not valid for all objects that have a LockObject method. Review the different method descriptions for information about the lock types each object supports.

It is sometimes possible for an application to obtain an additional lock on an already locked object. The following table defines what lock options are available to an application (App2) that wants to request a lock on an object that is currently locked by another application (App1).

<table>
<thead>
<tr>
<th>App1 lock obtained</th>
<th>olapLockRead</th>
<th>olapLockExtendedRead</th>
<th>olapLockProcess</th>
<th>olapLockWrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>olapLockRead</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>olapLockExtendedRead</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>olapLockProcess</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>olapLockWrite</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

If a lock request is denied, DSO raises the error mderrLockCannotBeObtained. If the lock request was denied because the object is already locked by another application, the Description property of the Error object contains the name of the user holding the lock, the computer name where the lock was obtained, and the description the application provided when
it obtained the lock.

In certain situations, an application can delete an object and cause another application to fail if both applications are using the same object. For example, assume that App1 creates an object and obtains an `olapLockWrite` lock, and App2 obtains an `olapLockRead` lock on the same object. If App1 deletes the object, the object reference in App2 will now not be valid. You can check the validity of an object in this situation by examining its `Parent` property, which does not contain a valid parent object if the object reference is not valid.

**See Also**

- [OlapLockTypes](#)
- [LockObject](#)
- [UnlockObject](#)
Tips for Creating Member Properties for Multiple Languages

The Caption and Language properties of the DSO.MemberProperty object allow you to tailor member properties to users with specific language requirements. With this feature, a single cube can serve groups of users without a common language.

When a client application's query involves member properties with identical captions, the Analysis server uses the member property object whose Language property most closely matches the application's LocaleID value. Multiple member properties can have identical values for Caption only if they each have a different value for the Language property, so that the member property most appropriate for the LocaleID value of the client application can be used. For a cube that serves client applications in only one language, the Language property for each member property object should be set to languageAny.

The Name property of a clsMemberProperty object contains the name of the source column for the data contained in the member property. The Caption property contains the name of the member property as it appears to the client application.

Add Members Properties for Multiple Languages

The following code example creates two member properties for Store Manager in the TestDB database. One is for English-speaking users, the other for Spanish-speaking users.

Example

The following code example adds a new member property, Store Manager, for English and Spanish languages:

Private Sub AddMultiLangMembers()
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
Dim dsoDS As DSO.DataSource
Dim dsoDim As DSO.Dimension
Dim dsoLevel As DSO.Level
Dim dsoMember As DSO.MemberProperty

Dim strDBName As String
Dim strLQuote As String
Dim strRQuote As String

' Define constants used for the ColumnType property
' of the DSO.Level object.
' Note that these constants are identical to
' those used in ADO in the DataTypeEnum enumeration.
Const adWChar = 130

' Initialize variables for the database name.
strDBName = "TestDB"

' Create a connection to the Analysis server.
dsoServer.Connect "LocalHost"

' Set the database object.
Set dsoDB = dsoServer.MDStores(strDBName)

' Set the data source for the database object.
' A data source is required to run this example.
If dsoDB.DataSources.Count = 0 Then
    MsgBox "Database " & dsoDB.Name & "," & 
        " has no data sources."
Else
    Set dsoDS = dsoDB.DataSources(1)
End If
' Get database-specific delimiter characters.
strLQuote = dsoDS.OpenQuoteChar
strRQuote = dsoDS.CloseQuoteChar

' Retrieve the Store dimension.
Set dsoDim = dsoDB.Dimensions("Stores")

' Retrieve the Store ID level.
Set dsoLevel = dsoDim.Levels("Store ID")

' First, create the English (and default) member property.
Set dsoMember = dsoLevel.MemberProperties.AddNew("Store Manager")
    dsoMember.SourceColumn = strLQuote & "store" & strRQuote & ".
    strLQuote & "store_manager" & strRQuote
    dsoMember.ColumnSize = 255
    dsoMember.ColumnType = adWChar
    dsoMember.Caption = "Store Manager"
    dsoMember.Language = languageAny
' Next, create an identical one for Spanish users.
Set dsoMember = _
    dsoLevel.MemberProperties.AddNew("Encargado de Almacén")
    dsoMember.SourceColumn = strLQuote & "store" & strRQuote & ".
    strLQuote & "store_manager" & strRQuote
    dsoMember.ColumnSize = 255
    dsoMember.ColumnType = adWChar
    dsoMember.Caption = "Store Manager"
    dsoMember.Language = languageSpanish

' Update the Stores dimension.
If dsoLevel.IsValid And dsoDim.IsValid Then
    dsoDim.Update
End If
End Sub
Using Earlier Versions of Analysis Services

You can create objects supported by previous versions of Microsoft® SQL Server™ 2000 Analysis Services. Decision Support Objects (DSO) can read and process objects created in previous versions of Analysis Services that support such objects. In particular, the creation of virtual dimensions is now supported in a more efficient fashion; although code written for SQL Server 7.0 OLAP Services to create virtual dimensions will continue to function, the more efficient process of virtual dimension creation supported in SQL Server 2000 Analysis Services is recommended.

See Also

Virtual Dimensions Created in Version 7.0
Analysis Services Programming
Microsoft® SQL Server™ 2000 Analysis Services offers substantial opportunity for you to create and integrate custom applications. The server object model, Decision Support Objects (DSO), provides interfaces and objects that can be used with any COM automation programming language such as Microsoft® Visual Basic® (version 5.0 and later). Additionally, Microsoft Visual C++® programmers can use DSO. DSO objects support both early and late binding.

The Programmer's Reference provides detailed information about DSO for use in developing custom applications that interact with the Analysis server. For more information about other programming tools for Analysis Services, see Programming Analysis Services Applications.

Topics in this section cover the following subjects.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>The specifics of each interface that your program uses to manipulate DSO objects, including collections, methods, and properties managed by the interface.</td>
</tr>
<tr>
<td>Events</td>
<td>Details on events supported by DSO objects, including database reporting and object processing events.</td>
</tr>
<tr>
<td>Objects</td>
<td>Information about DSO objects. Topics for objects that provide their own default interfaces also include collections, methods, and properties specific to those objects.</td>
</tr>
<tr>
<td>Enumerations</td>
<td>The details of the enumerations provided by DSO, and information about how to use them.</td>
</tr>
<tr>
<td>Collections</td>
<td>Information about the collections used in DSO, including the generic methods and properties that apply to these collections.</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Interfaces

There are a number of interfaces in Decision Support Objects (DSO). Objects that have similar functionality implement a common interface.

For example, databases, cubes, partitions, and aggregations implement the MDStore interface. An MDStore object is a container of multidimensional data. Databases contain cubes of related information, cubes contain partitions that store data, and aggregations are precalculated summaries of data associated with partitions. MDStore objects have similar structures. They contain collections of dimensions that categorize the data, the data sources that specify which relational database management system (RDBMS) contains fact and dimension tables, the roles that define the security permissions, and so on.

Given a reference to an MDStore interface or any other DSO interface, you can determine which type of the object you are dealing with by examining the ClassType property. The objects that implement the MDStore interface can have the following class types: clsDatabase, clsCube, clsPartition, and clsAggregation. Throughout the programmer's reference, DSO objects are identified using the notational prefix "cls." The DSO ClassTypes enumeration contains the complete list of all DSO class types.

Not all objects that implement a common interface implement the interface in the same way. Some objects do not implement all of the interface properties, methods, and collections. For example, database objects (clsDatabase) implement the BeginTrans method of the MDStore interface, but cube objects (clsCube) do not. Some objects restrict access to certain properties so that they become read-only rather than read/write. If you attempt to access a property or invoke a method that is not implemented, DSO raises an error.

DSO exposes the Command, Dimension, Level, MDStore, Measure, and Role interfaces.

The following table lists the DSO interfaces and the types of objects that implement them.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Implemented by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>The Command interface exposes</td>
<td>clsDatabaseCommand</td>
</tr>
</tbody>
</table>
functionality for defining and managing Multidimensional Expressions (MDX) statements to be executed on the Analysis server.

**Dimension**

The **Dimension** interface defines the properties, methods, and collections that you can use to manipulate different types of dimensions: database dimensions, cube dimensions, partition dimensions, and aggregation dimensions.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>clsDatabaseDimension</th>
<th>clsCubeDimension</th>
<th>clsPartitionDimension</th>
<th>clsAggregationDimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>clsDatabaseLevel</td>
<td>clsCubeLevel</td>
<td>clsPartitionLevel</td>
<td>clsAggregationLevel</td>
</tr>
<tr>
<td>MDStore</td>
<td>clsDatabase</td>
<td>clsCube</td>
<td>clsPartition</td>
<td>clsAggregation</td>
</tr>
<tr>
<td>Measure</td>
<td>clsCubeMeasure</td>
<td>clsPartitionMeasure</td>
<td>clsAggregationMeasure</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>clsDatabaseRole</td>
<td>clsCubeRole</td>
<td>clsMiningModelRole</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some DSO objects do not implement a common interface. You access these objects by using their default interface: clsServer, clsDataSource, clsMiningModel, clsColumn, clsCubeAnalyzer, clsPartitionAnalyzer, and clsMemberProperty.

**Remarks**

The DSO type library exposes several object classes, such as Cube, Database,
Partition, CubeDimension, and so on. These objects are reserved for future use and are not intended to be used in DSO applications. You should use the named interfaces instead. For example, use the following code to create a new object of ClassType clsDatabase (a database object implements the MDStore interface):

'Assume an object (dsoServer) of ClassType clsServer exists. 'Add database object to server's MDStores collection. Dim dsoDB As MDStore ' Declare the object by the interface. Set dsoDB = dsoServer.MDStores.AddNew("MyDB")

Always use the appropriate interface for the object. For example, major objects typically use the MDStore interface as their appropriate interface. The following code example shows the appropriate interface to use for cube objects:

Private dsoCube As DSO.Cube ' INCORRECT - DO NOT USE. Private dsoCube As DSO.MDStore ' CORRECT

The only exceptions to this rule are Database objects. If your client application needs to trap database events, the internal interface of the object should be used instead of the MDStore interface, as shown in the following code example:

Public dsoDB as DSO.MDStore ' Use this statement if you do not need Public WithEvents dsoDB as DSO.Database ' Use this statement if you
Command Interface

In Decision Support Objects (DSO), certain objects can contain a series of user-defined commands that are automatically executed on the PivotTable® Service client when you access the object. These commands can include expressions written in Multidimensional Expressions (MDX) that define calculated members, named sets, library references, and other commands.

The objects that implement the Command interface, CubeCommand, DBCommand, and RoleCommand, have a ClassType property value of clsCubeCommand, clsDatabaseCommand, or clsRoleCommand respectively. The Command interface provides collections, methods, and properties to manipulate these objects.

Applies To

- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

Collections

The Command interface contains the following collection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties for the command object</td>
</tr>
</tbody>
</table>

Methods

The Command interface contains the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type</td>
</tr>
</tbody>
</table>
**LockObject**
Locks an object

**UnlockObject**
Unlocks a previously locked object

**Update**
Saves the definition of the command object in the metadata repository

---

**Properties**

The **Command** interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific class type</td>
</tr>
<tr>
<td>CommandType</td>
<td>Returns an enumeration constant that identifies the command's use</td>
</tr>
<tr>
<td>Description</td>
<td>Sets or returns the description of the command object</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the <strong>Name</strong> and <strong>Statement</strong> properties are empty and whether the command object belongs to a collection</td>
</tr>
<tr>
<td>Name</td>
<td>Sets or returns the name of the command object</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the command object in the <strong>Commands</strong> collection of the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td>ParentObject</td>
<td>Returns a reference to the parent object of which this object is a child</td>
</tr>
<tr>
<td>Statement</td>
<td>Sets or returns the text of the command statement</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

---

**See Also**

**Commands**
Collections, Command Interface

The **Command** interface contains the following collection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties for the command object</td>
</tr>
</tbody>
</table>

**Access**

Read/write

**See Also**

[Command Interface](#)
Methods, Command Interface

The Command interface contains the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks an object</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks a previously locked object</td>
</tr>
<tr>
<td>Update</td>
<td>Saves the definition of the command object in the metadata repository</td>
</tr>
</tbody>
</table>

For the Command interface, these methods apply only to objects of ClassType clsDatabaseCommand.

See Also

Command Interface
Clone (Command Interface)

The **Clone** method of the **Command** interface copies the properties of an existing object to a target object of the same class type. The target object must exist prior to using the **Clone** method.

**Applies To**

*clsDatabaseCommand*

**Syntax**

```csharp
object.Clone(ByVal TargetObject As Command, [ByVal Options As CloneOptions = cloneMajorChildren])
```

- **object**
  - The object whose property values are to be copied.

- **TargetObject**
  - A previously created object of the same class type.

- **Options**
  - For objects of **ClassType clsDatabaseCommand**, the **CloneOptions** argument has no effect and is ignored.

**Example**

The following example clones a command object:

'Assume a command object (dsoCmd) exists.
Dim dsoCmdCopy as new DSO.Command
dsoCmd.Clone dsoCmdCopy

**See Also**
Command Interface
LockObject (Command Interface)

The LockObject method of the Command interface locks an object to prevent multiple users from concurrently changing the object.

Applies To

clsDatabaseCommand

Syntax

object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String)

object

The object to lock.

LockType

One of the lock types of the OlapLockTypes enumeration. For more information, see OlapLockTypes.

LockDescription

A string that contains a description of the lock. This argument is available to other applications attempting to obtain a lock.

Remarks

Of the four types of locks defined by the OlapLockTypes enumeration, only OlapLockRead and OlapLockWrite apply to the Command interface.

<table>
<thead>
<tr>
<th>Lock type</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlapLockRead</td>
<td>Applications can read the properties of the command object from the repository but cannot make changes until the lock is released (this includes the application that created the lock).</td>
</tr>
</tbody>
</table>
The application that created the lock can modify the object's properties and save them in the repository using the **Update** method. Other applications cannot read the properties of the object until the lock is released.

For more information about object locking, see [LockObject](#).

### Example

The following example locks a command object so that it can be modified. It then unlocks the object and updates the repository information for that object.

'Assume a command object (dsoCmd) exists.  
```
dsoCmd.LockObject OlapLockRead, "Updating command, please wait."
' (Insert code to change command object here.)
```
dsoCmd.Update  
dsoCmd.UnlockObject

### See Also

[Command Interface](#)
UnlockObject (Command Interface)

The UnlockObject method of the Command interface releases the lock previously established on a command object by the LockObject method.

Applies To

clsDatabaseCommand

Syntax

object.UnlockObject

object

The object to unlock.

Remarks

If an application that created one or more locks terminates before it can free them using the UnlockObject method, the Analysis server automatically releases the locks when the connection with the application is closed.

Example

The following example locks a command object so that it can be modified. It then unlocks the object and updates its repository information.

'Assume a command object (dsoCmd) exists.

dsoCmd.LockObect OlapLockRead, "Updating command, please wait."
' (Insert code to change command object here.)
dsoCmd.Update
dsoCmd.UnlockObject

See Also
Command Interface
Update (Command Interface)

The Update method of the Command interface saves the definition of a command object in the meta data repository.

Applies To

.clsDatabaseCommand

Syntax

.object.Update

.object

The command object to update.

Remarks

Use this method when you want to save changes to an object. Any changes made to an object will have session scope until this method is executed.

Example

The following example locks a command object so that it can be modified. It then unlocks the object and updates its repository information.

'Assume a command object (dsoCmd) exists.
dsoCmd.LockObect OlapLockRead, "Updating command, please wait."
'(insert code to change command object here)
dsoCmd.Update
dsoCmd.UnlockObject

See Also

.Command Interface
Properties, Command Interface

The **Command** interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific class type</td>
</tr>
<tr>
<td><strong>CommandType</strong></td>
<td>Returns an enumeration constant that identifies the command option</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Sets or returns the description of the command object</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the <strong>Name</strong> and <strong>Statement</strong> properties are empty and whether the command object belongs to a collection</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Sets or returns the name of the command object</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the command object in the <strong>Commands</strong> collection of the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>ParentObject</strong></td>
<td>Returns a reference to the parent object of which this object is a child</td>
</tr>
<tr>
<td><strong>Statement</strong></td>
<td>Sets or returns the text of the command statement</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

Access Cross-Reference

The following table shows whether the property is read/write (R/W) or read-only (R) for different objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>clsDatabaseCommand</th>
<th>clsCubeCommand</th>
<th>clsRoleCommand</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>CommandType</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>IsValid</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Name</td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Parent</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>ParentObject</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Statement</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>SubClassType</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

**See Also**

[Command Interface](#)
ClassType (Command Interface)

The **ClassType** property of the **Command** interface contains an enumeration constant that identifies the class designation in the Decision Support Objects (DSO) object model.

### Applies To

- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

### Data Type

**ClassTypes**

### Access

Read-only

### Example

Use the following code to return the class type of a command object and determine which object class has been returned:

'Assume an object (dsoCommand) of ClassType clsCommand exists.
Dim enuClassType As DSO.ClassTypes
enuClassType = dsoCommand.ClassType
Select Case enuClassType
    Case clsDatabaseCommand
        ' Insert code for a database command.
    Case clsCubeCommand
        ' Insert code for a cube command.
    Case clsRoleCommand

' Insert code for a role command
Case Else
  ' Insert code for when this is not a command object.
End Select

See Also

Command Interface
CommandType (Command Interface)

The `CommandType` property of the `Command` interface contains an enumeration constant that identifies the command option.

**Applies To**
- `clsCubeCommand`
- `clsDatabaseCommand`
- `clsRoleCommand`

**Data Type**
- `CommandTypes`

**Access**
Read/write

**Remarks**
The value of `CommandType` determines how the Analysis server interprets the command object. The following table lists the possible values.

<table>
<thead>
<tr>
<th>Command type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdCreateAction</td>
<td>The command contains a CREATE ACTION statement. For more information, see <a href="#">CREATE ACTION Statement</a>.</td>
</tr>
<tr>
<td>cmdCreateMember</td>
<td>The command defines one or more calculated members.</td>
</tr>
<tr>
<td>cmdCreateSet</td>
<td>The command defines one or more sets of existing members.</td>
</tr>
<tr>
<td>cmdUseLibrary</td>
<td>The command specifies a third-party DLL that contains functions to be registered for use in</td>
</tr>
</tbody>
</table>
Multidimensional Expressions (MDX).

**cmdUnknown**
The command defines statements that are not included in any of the other command types, such as DROP MEMBER statements or new statements that may be added to future versions.

**cmdCreateCellCalculation**
The command defines a calculated cells definition.

**IMPORTANT** To ensure compatibility with Analysis Manager, you should create only one action, calculated member, named set, or calculated cells definition per command.

### Examples

#### A. Specifying the Command Type
Use the following code to specify a command type for an existing command object:

```vbnet
CommandObject.CommandType = cmdCreateMember
```

#### B. Determining the Command Type
Use the following code to determine the type of command in use:

```vbnet
Dim CommandType As DSO.CommandType
CommandType = CommandObject.CommandType
Select Case CommandType
    Case cmdCreateAction
        ' Insert code to create an action.
    Case cmdCreateMember
        ' Insert code to define a calculated member.
    Case cmsCreateSet
        ' Insert code to define a named set of existing members.
    Case cmdUseLibrary
        ' Insert code to use a third-party library.
```
Case cmdCreateCellCalculation
    ' Insert code to create a calculated cells definition.
Case cmdUnknown
    ' Insert code to define other statements.
End Select

See Also

Command Interface
**Description (Command Interface)**

The **Description** property of the **Command** interface contains the description of the command object.

**Applies To**
- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

**Data Type**
String

**Access**
Read/write

**Remarks**
The primary mechanism for identifying individual command objects is the **Name** property. The purpose of the **Description** property is to provide additional descriptive information.

**Example**
Use the following code to set a command object description:

'Assume a command object (dsoCmd) exists.
dsoCmd.Description = "Create a new profit member as sales-cost."

**See Also**
- Command Interface
Name
IsValid (Command Interface)

The IsValid property of the Command interface indicates whether the Name and Statement properties are empty and whether the command object belongs to a collection.

Applies To

clsCubeCommand
clsDatabaseCommand
clsRoleCommand

Data Type

Boolean

Access

Read-only

Remarks

Because of the flexible nature of command objects, only the following checks are performed:

- The Name property of the command object is not empty.
- The Command object belongs to a Commands collection.
- The Statement property of the command object is not empty.

Executing a command is the only way to determine whether the command functions correctly.
**Example**

Use the following code to return the validity status of a command object:

'Assume a command object (dsoCmd) exists.
If dsoCmd.IsValid Then
    'Insert code to save the command.
Else
    'Return an error with one or more properties.
End If

**See Also**

[Command Interface](#)

[Name](#)

[Statement](#)
Analysis Services Programming

**Name (Command Interface)**

The **Name** property of the **Command** interface contains the name of a command object. This property is the primary mechanism for identifying individual command objects.

**Applies To**

- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

**Data Type**

String

**Access**

Read/write (read-only after the object has been named)

**Remarks**

Within a cube, each command must have a unique name. For more information about naming conventions for Decision Support Objects (DSO) objects, see **Considerations For Naming Decision Support Objects**.

**Example**

Use the following code to print a command object's name in the immediate window:

'Assume a command object (dsoCmd) exists.
dbg.print dsoCmd.Name

**See Also**
Command Interface
OrdinalPosition (Command Interface)

The **OrdinalPosition** property of the **Command** interface contains the ordinal position of a command in the **Commands** collection in the parent **MDStore** object.

**Applies To**

- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

**Data Type**

Integer

**Access**

Read-only

**Remarks**

Ordinal position determines the order in which commands are executed. This order is important when one command depends on another command (for example, a CREATE SET command that uses a member defined in a CREATE MEMBER command). In this case, the CREATE MEMBER command should have a lower **OrdinalPosition** property value than the CREATE SET command. However, the ordinal position of the command does not affect the solve order of the calculated member or calculated cells definition the command may create. For more information on how solve order affects calculated cells and calculated members, see [Understanding Pass Order and Solve Order](#).

**Example**

The following code creates three new commands in the **Commands** collection
of the **FoodMart 2000** database. It then enumerates the collection, printing the **OrdinalPosition** and **Name** properties to the Debug window. Then, the code example deletes and re-creates the first command, and again enumerates the collection to demonstrate the change in ordinal position on the other commands.

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCmd As DSO.Command

' Connect to the local Analysis server.
dsoServer.Connect "LocalHost"

' Open the FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Add three sample commands.
Set dsoCmd = dsoDB.Commands.AddNew("Command3")
Set dsoCmd = dsoDB.Commands.AddNew("Command1")
Set dsoCmd = dsoDB.Commands.AddNew("Command2")

' Iterate through the commands for the database.
For Each dsoCmd In dsoDB.Commands
    ' Print its name and ordinal position
    Debug.Print dsoCmd.OrdinalPosition & " = " & dsoCmd.Name
Next

' Now, delete the Command3 command and add it again.
dsoDB.Commands.Remove "Command3"
Set dsoCmd = dsoDB.Commands.AddNew("Command3")

' Iterate again through the commands for the database.
Debug.Print "-----"
For Each dsoCmd In dsoDB.Commands
    ' Print its name and ordinal position
```
Debug.Print dsoCmd.OrdinalPosition & " = " & dsoCmd.Name
Next

See Also

Command Interface
CREATE SET Statement
CREATE MEMBER Statement
**Parent (Command Interface)**

The **Parent** property of the **Command** interface contains a reference to the parent **MDStore** object of which this object is a child.

**Applies To**

- **clsCubeCommand**
- **clsDatabaseCommand**
- **clsRoleCommand**

**Data Type**

**MDStore**

**Access**

Read-only

**Remarks**

The class of the parent object depends on the class of the command object.

<table>
<thead>
<tr>
<th>Command object class</th>
<th>Parent object class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseCommand</td>
<td>clsDatabase</td>
</tr>
<tr>
<td>ClsCubeCommand</td>
<td>clsCube</td>
</tr>
<tr>
<td>ClsRoleCommand</td>
<td>clsCubeRole clsDatabaseRole clsMiningModelRole</td>
</tr>
</tbody>
</table>

**Example**

The following example creates a cube command and assigns it to the first cube in the **MDStores** collection of the **FoodMart 2000** database. It then prints some
of the properties of the parent object by using the **Parent** property of the cube command object.

```
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore
Dim dsoDBCmd As DSO.Command
Dim dsoCubeCmd As DSO.Command

' Connect to the Analysis server.
   dsoServer.Connect "LocalHost"

' Get a reference to the FoodMart 2000 database.
   Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Get the first cube in the database's collection.
   Set dsoCube = dsoDB.MDStores(1)
   Debug.Print "Cube.Name = " & dsoCube.Name

' Add the command to the cube's collection.
   Set dsoCubeCmd = dsoCube.Commands.AddNew("TempCommand")
   Debug.Print "Cube.Commands(""TempCommand"").Name =" & _
                  dsoCube.Commands("TempCommand").Name

' Print the properties of the command's
' Parent object.
   Debug.Print "   .Parent properties" & vbCrLf & _
             "   -----------"
   Debug.Print "   TypeName(dsoCubeCmd.Parent) = " & _
              TypeName(dsoCubeCmd.Parent)
   If dsoCubeCmd.Parent.ClassType = clsCube Then
      Debug.Print "   .ClassType = clsCube"
   Else
      Debug.Print "This line should never be executed."
```
End If
Debug.Print " .Description = " & dsoCubeCmd.Parent.Description
Debug.Print " .Name = " & dsoCubeCmd.Parent.Name

dsoCube.Commands.Remove ("TempCommand")

See Also

Command Interface
**ParentObject (Command Interface)**

The **ParentObject** property returns a reference to the default interface of the parent object.

**Applies To**

- `clsCubeCommand`
- `clsDatabaseCommand`
- `clsRoleCommand`

**Data Type**

Object

**Access**

Read-only

**Remarks**

This property returns the default interface for the parent of the command object. For objects whose **ClassType** property is `clsRoleCommand`, this property returns an object of **ClassType** `clsCubeRole` or `clsDatabaseRole`.

The returned object and its class type depend on the class type of the current command object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Parent object interface</th>
<th>Parent object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseCommand</td>
<td>MDStore</td>
<td>clsDatabase</td>
</tr>
<tr>
<td>clsCubeCommand</td>
<td>MDStore</td>
<td>clsCube</td>
</tr>
<tr>
<td>clsRoleCommand</td>
<td>Role</td>
<td>classDatabaseRole, clsCubeRole, clsMiningModelRole</td>
</tr>
</tbody>
</table>
Example

Comparing the Parent and ParentObject Properties

The following example compares the use of the Parent property with the ParentObject property:

Dim dsoDB As DSO.MDStore
Dim dsoRole As DSO.Role
Dim dsoRoleCmd As DSO.Command
Dim dsoDatabaseRole As DSO.Role

dsoServer.Connect "LocalHost"
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

'Get the first role in the database.
'This should be the Everyone role.
Set dsoRole = dsoDB.Roles(1)

Debug.Print "DatabaseRole.Name = " & dsoRole.Name
'Add a new command to the role.
Set dsoRoleCmd = dsoRole.Commands.AddNew("RoleCmd")

'Print the properties of the Parent object.
Debug.Print ".Parent properties" & vbCrLf & "--------
Debug.Print "Interface type = " & TypeName(dsoRoleCmd.Parent)
If dsoRoleCmd.Parent.ClassType = clsDatabase Then
  Debug.Print ".ClassType = clsDatabase"
Else
  Debug.Print "This should never be printed."
End If
Debug.Print ".Description = " & dsoRoleCmd.Parent.Description
Debug.Print ".Name = " & dsoRoleCmd.Parent.Name

'Print the properties of the ParentObject object
Debug.Print ".ParentObject properties" & vbCrLf & "--------"
Debug.Print "Interface type = " & TypeName(dsoRoleCmd.ParentObject)
Set dsoDatabaseRole = dsoRoleCmd.ParentObject
If dsoDatabaseRole.ClassType = clsDatabaseRole Then
   Debug.Print ".ClassType = clDatabaseRole"
Else
   Debug.Print "This should never be printed."
End If
Debug.Print ".ClassType = " & dsoDatabaseRole.ClassType
Debug.Print ".Description = " & dsoDatabaseRole.Description
Debug.Print ".Name = " & dsoDatabaseRole.Name

See Also

Command Interface
**Statement (Command Interface)**

The **Statement** property of the **Command** interface contains the text of the command statement.

**Applies To**

- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

**Data Type**

String

**Access**

Read/write

**Remarks**

For more information about types of command statements, see [CommandType](#).

The names used in statements that define actions, calculated members, named sets, and calculated cells must be unique within a cube. For example, if you create an action named Test, you cannot create a calculated cells definition named Test. For more information about naming conventions for Decision Support Objects, see [Considerations For Naming Decision Support Objects](#).

**Examples**

**A. Creating a Command Object (cmdCreateAction)**

Use the following code to create a **cmdCreateAction** command object:
CommandObject.Statement = "CREATE ACTION Sales.ShowCustDetails FOR [Customer] MEMBERS As " \\
"IIf(Customers.CurrentMember.Properties("'Existing Customer'"))" \\
" = "'True','" & \\
"'http://MyServer/CustomerDetails.ASP?CustID='" + " & \\
"Customers.CurrentMember.ID,") " & \\
"TYPE = URL " & \\
"APPLICATION = 'IE' " & \\
"DESCRIPTION = 'Launch the customer details page for " & \\
"this specific customer.'"

You can use CURRENTCUBE in command statements to refer to the cube that contains the command object. This syntax makes it easier to copy command objects between cubes. For example, you can reconstruct the previous statement as:

CommandObject.Statement = "CREATE ACTION CURRENTCUBE..:' FOR [Customer] MEMBERS As " & \\
"IIf(Customers.CurrentMember.Properties("'Existing Customer'"))" \\
" = "'True','" & \\
"'http://MyServer/CustomerDetails.ASP?CustID='" + " & \\
"Customers.CurrentMember.ID,") " & \\
"TYPE = URL " & \\
"APPLICATION = 'IE' " & \\
"DESCRIPTION = 'Open the customer details page for " & \\
"this specific customer.'"

**B. Creating a Command Object (cmdCreateMember)**

Use the following code to create a `cmdCreateMember` command object:

CommandObject.Statement = "CREATE MEMBER Sales.Measures.PROFIT AS " \\
"'Measures.Sales - Measures.Cost'"

**C. Creating a Command Object (cmdCreateSet)**
Use the following code to create a cmdCreateSet command object:

```
```

D. Creating a Command Object (cmdUseLibrary)

Use the following code to create a cmdUseLibrary command object:

```
CommandObject.Statement = "USE LIBRARY "MyStatFunctions.dll"
```

E. Creating a Command Object (cmdCreateCellCalculation)

Use the following code to create a cmdCreateCellCalculation command object:

```
CommandObject.Statement = CREATE SESSION CELL CALCULATION [Sales].[Mexico Adjustments] FOR '(Descendants([Mexico], [City], SELF))' AS '<expression>', CONDITION = '[Time].CURRENTMEMBER.NAME=[2000]'"
```

F. Returning a Command Object Statement

Use the following code to return a command object statement:

```
Dim strCommandStatement As String
strCommandStatement = CommandObject.Statement
```

See Also

- Command Interface
- CREATE MEMBER Statement
- CREATE SET Statement
- MDX (Administrative Tools)
SubClassType (Command Interface)

The SubClassType property of the Command interface contains an enumeration constant that identifies the subclass type of the object. Objects that implement the Command interface always have a SubClassType of sbclsRegular.

Applies To

- clsCubeCommand
- clsDatabaseCommand
- clsRoleCommand

Data Type

SubClassTypes

Access

Read-only

See Also

- Command Interface
- CREATE MEMBER Statement
- CREATE SET Statement
- MDX (Administrative Tools)
**Dimension Interface**

In Decision Support Objects (DSO), dimensions are represented by objects that contain collections of other objects that, in turn, represent levels in the dimensions. For example, a Time dimension often contains the levels Year, Quarter, Month, and Day. The levels of a cube are defined by columns in a dimension table that is stored in the data warehouse database. When a dimension object is processed, the Analysis server constructs the dimension and its levels and then populates them with members from the dimension table. For more information about DSO, see *Introducing Decision Support Objects*.

All DSO dimension objects implement the **Dimension** interface, and you manipulate these objects through this interface. The **ClassType** property of the dimension object specifies the dimension's type. The **ClassType** value of a database dimension is **clsDatabaseDimension**. The **ClassType** values of cube, partition, and aggregation dimensions (supported by the **CubeDimension**, **PartitionDimension**, **DbDimension**, and **AggregationDimension** objects) are **clsCubeDimension**, **clsPartitionDimension**, and **clsAggregationDimension**, respectively. The **Dimension** interface provides collections, methods, and properties to manipulate these objects. Dimensions reside in the **Dimensions** collection of the **MDStore** object that represents a database, cube, partition, or aggregation.

To use dimensions, create them in the **Dimensions** collection of a database object. Then assign some or all of the dimensions to a cube. The dimensions assigned to a cube automatically apply to its partitions and aggregations, and you can explicitly associate them with virtual cubes that use the cube.

This topic discusses different types, varieties, and uses of dimensions and provides information about how to work with them in DSO. For more information about dimensions, see *Dimensions*.

**Types of DSO Dimensions**

The following table describes each type of dimension and the context in which it is used.
<table>
<thead>
<tr>
<th>Dimension type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database dimension</td>
<td>The dimensions are contained in a database</td>
</tr>
<tr>
<td>Cube dimension</td>
<td>The dimensions are contained in a cube</td>
</tr>
<tr>
<td>Partition dimension</td>
<td>The dimensions are used in a partition</td>
</tr>
<tr>
<td>Aggregation dimension</td>
<td>The dimensions are contained in an aggregation</td>
</tr>
</tbody>
</table>

Database dimensions define the structure of the dimension and the data source where the dimension tables exist.

Any of the database dimensions can be used in a cube, if a join can be made between the dimension table and the fact table of the cube. Cube dimensions have the same name as their corresponding database dimensions. When an existing database dimension is added to the Dimensions collection of a cube, DSO automatically defines and creates all of the cube level objects for the cube dimension. A cube dimension inherits its properties from the corresponding database dimension; some of those properties can be customized in the cube. For example, you can specify how a cube dimension is used in the design of aggregations by setting the AggregationUsage property of the dimension.

The dimensions of a partition relate to the dimensions of its parent cube in the same way that cube dimensions relate to database dimensions. Each partition dimension has a corresponding cube dimension and has the same number of levels as the cube dimension.

Aggregation dimensions are the dimensions used within an aggregation. They are different from their corresponding partition dimensions in that they usually have fewer levels. This is because the aggregation dimensions represent the level of granularity of the data in that aggregation.

For example, the following illustration represents two aggregations. The first summarizes sales by year for store cities and product brand. The Time aggregation dimension in this case has only one level: Year. The Store and Product dimensions have all of their respective levels. The second aggregation summarizes sales by Quarter and Region for products by category. The Time aggregation dimension has two levels: Year and Quarter. The Store and Products dimensions, on the other hand, contain only the first levels of each dimension: Region and Category.
Shared and Private Dimensions

DSO dimensions can be either shared or private. A shared dimension is one that can be used in multiple cubes, but a private dimension can only be used in a single cube. For more information about shared and private dimensions, see Shared and Private Dimensions.

Private dimensions use a special naming convention to identify the cubes to which they belong. In all other aspects, private dimensions are identical to shared dimensions. The name of a private dimension is constructed by using the cube name, followed by the caret character (^), followed by the dimension name. To create a private dimension in DSO, name the dimension according to this convention. This naming convention allows private dimensions in different cubes to have the same name, and the cube name prefix ensures uniqueness within the dimension collection of the database. For example, NorthWestSales^Stores represents a private dimension of stores created for use in the NorthWestSales cube.

You can programmatically determine whether a dimension is shared or private by reading its IsShared property. DSO determines the value of this property by examining the name of the dimension. All cubes, partitions, and aggregations that use the dimension inherit its IsShared property.

Parent-Child Dimensions

Parent-child dimensions contain self-joining hierarchies. Because the level hierarchy is variable, rather than rigidly set, parent-child dimensions are more flexible than regular dimensions. For more information about parent-child dimensions, see Parent-Child Dimensions.

In DSO, a parent-child dimension has a maximum of two levels: the (All) level, which is optional, and a second level that acts as a template for building other levels.

You can create a parent-child dimension by setting the SubClassType property to sbclsParentChild. When you define a parent-child dimension, the system uses source table data to build a dimension that has a level hierarchy of variable depth.
Virtual Dimensions

Virtual dimensions are based on the columns or member properties of existing regular dimensions, and they can provide additional information in cubes that use these regular dimensions. For more information about virtual dimensions, see Virtual Dimensions.

To build a virtual dimension using DSO, create a regular dimension and set its IsVirtual property to True. Next, set the DependsOnDimension property to refer to the underlying source dimension. Finally, create levels for the virtual dimension and configure the MemberKeyColumn property of each virtual level to point to a source level or member property in the source dimension.

Although it is possible to create a shared virtual dimension that is based on a private dimension, this virtual dimension will work only in the cube that contains the private dimension.

Note In earlier versions of Microsoft® SQL Server™ 2000 Analysis Services, virtual dimensions were limited to having an (All) level and a second level whose SubClassType was set to sbclsVirtual. The dimension itself had a SubClassType of sbclsRegular. This convention is still supported for compatibility, but the new method of setting the dimension's IsVirtual property to True should be used, because it is more flexible and efficient.

Applies To

clsAggregationDimension
clsCubeDimension
clsDatabaseDimension
clsPartitionDimension

Examples

A. Adding a Parent-Child Dimension to a Database

The following example shows the addition of a parent-child dimension to a database:
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDim As DSO.Dimension
Dim dsoLevel As DSO.Level

' Connect to local Analysis server
dsoServer.Connect "LocalHost"

' Open FoodMart 2000 database
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Add a new dimension and set the data source
Set dsoDim.DataSource = dsoDB.DataSources("FoodMart")

' Add a new level to the new dimension.
Set dsoLevel = dsoDim.Levels.AddNew("Template Level", sbclsParentChild)

' Set the member key, parent key, and member name columns for the new level.
With dsoLevel
    .MemberKeyColumn = "Account.account_id"
    .ParentKeyColumn = "Account.account_parent"
    .MemberNameColumn = "Account.account_description"
End With

' Update the dimension.
dsoDim.Update

B. Creating a Database with Two Dimensions
This example shows how to create a database with two dimensions, DbDimA and DbDimB, and three cubes, CubeX, CubeY, and CubeZ. DbDimA is shared, but DbDimB is private to CubeZ. DbDimA can be associated with any or all of the cubes, but DbDimB can be associated only with CubeZ.

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDS As DSO.DataSource
Dim dsoCubeX As DSO.MDStore
Dim dsoCubeY As DSO.MDStore
Dim dsoCubeZ As DSO.MDStore
Dim dsoDbDimA As DSO.Dimension
Dim dsoDbDimB As DSO.Dimension

Dim strDSName As String, strDSConnect As String

' Set data source name and connection string
' to be used later.
strDSName = "FoodMart"
strDSConnect = "Provider=MSDASQL.1;User ID=sa;" & 
"Data Source=FoodMart;Connect Timeout=15"

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"

' Create a new database.
Set dsoDB = dsoServer.MDStores.AddNew("TestDB")

' Create a new data source.
Set dsoDS = dsoDB.DataSources.AddNew(strDSName)
' Set the data source name and connection string
' properties for the data source.
With dsoDS
    .Name = strDSName
    .ConnectionString = strDSConnect
```

Update
End With

' Create three new cubes.
Set dsoCubeX = dsoDB.MDStores.AddNew("CubeX")
Set dsoCubeY = dsoDB.MDStores.AddNew("CubeY")
Set dsoCubeZ = dsoDB.MDStores.AddNew("CubeZ")

' Create a shared dimension.
Set dsoDbDimA = dsoDB.Dimensions.AddNew("DbDimA") 'Shared

' Associate the shared dimension with CubeX and CubeY.
dsoCubeX.Dimensions.AddNew "DbDimA"
dsoCubeY.Dimensions.AddNew "DbDimA"

' Create a private dimension. The use of the
' caret (^) character separates the cube name
' and the private dimension name.
Set dsoDbDimB = dsoDB.Dimensions.AddNew("CubeZ^DbDimB"

' Associate the private dimension with CubeZ.
dsoCubeZ.Dimensions.AddNew ("CubeZ^DbDimB")

See Also

AggregationUsage
Collections, Dimension Interface
Dimensions
IsShared
IsVirtual
MDStore Interface
Methods, Dimension Interface
Properties, Dimension Interface
SubClassType
Collections, Dimension Interface

The **Dimension** interface implements the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td><strong>Levels</strong></td>
<td>The set of level objects associated with a dimension object</td>
</tr>
</tbody>
</table>

Access Cross-Reference

The following table shows whether the collection is read/write (R/W) or read-only (R) for different objects.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Database dimension</th>
<th>Cube dimension</th>
<th>Partition dimension</th>
<th>Aggregation dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CustomProperties</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Levels</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

See Also

[Dimension Interface](#)
**CustomProperties (Dimension Interface)**

The **CustomProperties** collection allows you to assign unique properties to objects that implement the **Dimension** interface.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Access**

Read/write

**Remarks**

The **CustomProperties** collection contains properties that accept user-defined values that are stored in the repository and can be used as needed. For example, an application can use this collection to store user interface parameters that are specific to this dimension (and might change) rather than storing them in the registry.

**Example**

The following example associates a custom property called Icon with a Geography dimension and gives it a string value of "GeographyIcon":

```
' Assume the existence of a Geography dimension object (dsoGeography) of ClassType clsDimension.
' Add a custom property to the dimension.
  Dim dsoProp As DSO.Property
  Set dsoProp = dsoGeographyDim.CustomProperties.Add("GeographyIcon", "Icon", vbString)
```
' Retrieve custom property values.
Dim dsoProp2 As DSO.Property
Set dsoProp2 = dsoDim.CustomProperties("Icon")
Debug.Print dsoProp2.Name, dsoProp2.Value

See Also

[CustomProperties](#)
[Dimension Interface](#)
Levels (Dimension Interface)

The Levels collection of the Dimension interface defines the set of level objects associated with a dimension object.

Applies To

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

See Also

- ClassType
- Dimension Interface
Methods, Dimension Interface

The **Dimension** interface supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks an object to prevent multiple users from concurrently changing the object</td>
</tr>
<tr>
<td>Process</td>
<td>Processes a dimension object</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks a previously locked dimension object</td>
</tr>
<tr>
<td>Update</td>
<td>Updates the definition of a dimension object in the metadata repository</td>
</tr>
</tbody>
</table>

For the **Dimension** interface, these methods apply only to objects of **ClassType clsDatabaseDimension**.

See Also

[Dimension Interface](#)
Clone (Dimension Interface)

The Clone method of the Dimension interface copies the properties and levels of an existing object to a target object of the same class type.

Applies To

clsDatabaseDimension

Syntax

object.Clone(ByVal TargetObject As Dimension, [ByVal Options As CloneOptions = cloneMajorChildren])

object

The Dimension object whose properties and levels are to be copied.

TargetObject

A previously created object of the same class type.

Options

The options to tell the method to what extent the source object should be duplicated. If no value is specified, the cloneMajorChildren option is used.

Remarks

The following values for Options are valid for cloning a dimension.

<table>
<thead>
<tr>
<th>Clone option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloneObjectProperties</td>
<td>The values of the properties of the source dimension are copied to the target dimension</td>
</tr>
<tr>
<td>cloneMinorChildren</td>
<td>The values of the properties and levels contained in the source dimension are copied to the target dimension</td>
</tr>
<tr>
<td>cloneMajorChildren</td>
<td>For dimension objects, this is the same as</td>
</tr>
</tbody>
</table>
**Example**

The following example copies the properties and levels of dimension DimA to dimension DimB:

```vbnet
dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDimA As DSO.Dimension
Dim dsoDimB As DSO.Dimension

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"

' Open the FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Create a new dimension, named DimA, to be used
' as a source dimension from which to clone.
Set dsoDimA = dsoDB.Dimensions.AddNew("DimA")
dsoDimA.Description = "First dimension"

' Create the target dimension, named DimB, and
' give it a different description.
Set dsoDimB = dsoDB.Dimensions.AddNew("DimB")
dsoDimB.Description = "Second dimension"

' Now, clone the DimA dimension into DimB.
dsoDimA.Clone dsoDimB, cloneMajorChildren

' Print the description of the DimB dimension
' to the Debug window.
Debug.Print dsoDimB.Description
```
See Also

Dimension Interface
LockObject (Dimension Interface)

The LockObject method of the Dimension interface locks an object to prevent multiple users from concurrently changing the object.

Applies To

clsDatabaseDimension

Syntax

object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String)

object

The Dimension object to lock.

LockType

One of the enumerated constants of the OlapLockTypes enumeration. For more information, see OlapLockTypes.

LockDescription

A string containing the description of the lock, available to other applications attempting to obtain a lock.

Remarks

This table explains how each value that can be specified in LockType affects a lock made on a dimension object.

<table>
<thead>
<tr>
<th>Lock type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlapLockRead</td>
<td>Applications can read the properties of the dimension object from the repository but cannot make changes until the lock is released (this includes the application that created the lock).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OlapLockWrite</td>
<td>The application that created the lock can modify the dimension object's properties and save them in the repository using the <strong>Update</strong> method. Other applications cannot read the properties of the object until the lock is released.</td>
</tr>
<tr>
<td>OlapLockExtendedRead</td>
<td>The properties of the dimension object and all of its dependent objects can be read (but not changed or processed) by other applications until the lock is released. This lock is used to prevent processing of dependent objects of a locked object (for example, dimensions that are shared by multiple cubes).</td>
</tr>
<tr>
<td>OlapLockProcess</td>
<td>This lock is similar to <strong>OlapLockExtendedRead</strong>, except the dimension object's <strong>Process</strong> method can be called by the application that created the lock. Other applications can read (but cannot change) the object's properties while the lock is in effect.</td>
</tr>
</tbody>
</table>

**Example**

The following example locks the Product dimension of the **FoodMart 2000** database, completely reprocesses it, and then unlocks it so others can make changes:

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDim As DSO.Dimension

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"

' Open FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")
```
'Open the Product dimension.
Set dsoDim = dsoDB.Dimensions("Product")

' Lock the dimension for processing.
dsoDim.LockObject olapLockProcess, "Locked for processing."

' Completely reprocess the dimension.
dsoDim.Process processFull

' Once complete, unlock the dimension.
dsoDim.UnlockObject

' Clean up.
Set dsoDim = Nothing
Set dsoDB = Nothing
dsoServer.CloseServer
Set dsoServer = Nothing

See Also

UnlockObject
Process (Dimension Interface)

The **Process** method of the **Dimension** interface creates and populates a dimension on the Analysis server.

**Applies To**

```
clsDatabaseDimension
```

**Syntax**

```
object.Process(ByVal Options As ProcessTypes)
```

- **object**
  - The **Dimension** object to process.

- **Options**
  - One of the valid **ProcessTypes** enumeration constants. For more information, see **ProcessTypes**.

**Remarks**

The following values for *Options* are valid for processing a dimension.

<table>
<thead>
<tr>
<th>Process type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processDefault</td>
<td>The default option. Setting this option causes the system to default to the necessary processing option based on the changes found in the data. In most cases, the system refreshes the dimension object's data (<strong>processRefreshData</strong>). However, if the structure of the dimension has changed, or the dimension has not yet been processed, the system fully processes the dimension (<strong>processFull</strong>).</td>
</tr>
<tr>
<td>processFull</td>
<td>Causes the dimension object to be fully processed or rebuilt. The object's structure is changed if needed</td>
</tr>
</tbody>
</table>
and its data is refreshed (that is, discarded and repopulated). This is the most complete type of processing supported. This operation occurs inside a transaction, allowing users to continue using current data while the transaction takes place. After the transaction is committed, the new data is available.

<table>
<thead>
<tr>
<th>processRefreshData</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causes the dimension object data to be refreshed (that is, discarded and repopulated), but does not change the object's structure. This operation occurs inside a transaction, allowing users to continue using current data while the transaction takes place. After the transaction is committed, the new data is available.</td>
</tr>
</tbody>
</table>

**Example**

The following example locks the Product dimension of the **FoodMart 2000** database, completely reprocesses it, and then unlocks it so others can make changes:

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDim As DSO.Dimension

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"

' Open FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Open the Product dimension.
Set dsoDim = dsoDB.Dimensions("Product")

' Lock the dimension for processing.
dsoDim.LockObject olapLockProcess, "Locked for processing."
```
' Completely reprocess the dimension.
dsoDim.Process processFull

' Once complete, unlock the dimension.
dsoDim.UnlockObject

' Clean up.
Set dsoDim = Nothing
Set dsoDB = Nothing
dsoServer.CloseServer
Set dsoServer = Nothing

See Also

Dimension Interface
Dimension Processing
UnlockObject (Dimension Interface)

The UnlockObject method of the Dimension interface releases a lock on a dimension object previously established by the LockObject method.

Applies To

clsDatabaseDimension

Syntax

object.UnlockObject

object

The Dimension object to unlock.

Remarks

Calling the UnlockObject method without first calling the LockObject method raises an error.

Example

The following example locks the Product dimension of the FoodMart 2000 database, completely reprocesses it, and then unlocks it so others make changes:

    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoDim As DSO.Dimension

    ' Connect to local Analysis server.
    dsoServer.Connect "LocalHost"

    ' Open FoodMart 2000 database.
    Set dsoDB = dsoServer.MDStores("FoodMart 2000")
'Open the Product dimension.
Set dsoDim = dsoDB.Dimensions("Product")

' Lock the dimension for processing.
dsoDim.LockObject olapLockProcess, "Locked for processing."

' Completely reprocess the dimension.
dsoDim.Process processFull

' Once complete, unlock the dimension.
dsoDim.UnlockObject

' Clean up.
Set dsoDim = Nothing
Set dsoDB = Nothing
dsoServer.CloseServer
Set dsoServer = Nothing

**See Also**

Dimension Interface

LockObject
**Update (Dimension Interface)**

The `Update` method of the `Dimension` interface updates the definition of a dimension object in the metadata repository.

**Applies To**

`clsDatabaseDimension`

**Syntax**

`object.Update`

`object`

The `Dimension` object to update.

**Remarks**

Objects of `ClassType clsAggregationDimension, clsCubeDimension, and clsPartitionDimension` do not implement the `Update` method. They are automatically updated when the `Update` method of the parent `MDStore` object is called.

**Example**

The following example changes the `Description` and `LastUpdated` properties for the Product dimension of the `FoodMart 2000` database and updates the dimension on the Analysis server:

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDim As DSO.Dimension

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"
```
' Open FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Open the Product dimension.
Set dsoDim = dsoDB.Dimensions("Product")

' Change the description and update date
' of the dimension.
dsoDim.Description = "Product dimension"
dsoDim.LastUpdated = Now

' Update the dimension.
dsoDim.Update

' Clean up.
Set dsoDim = Nothing
Set dsoDB = Nothing
dsoServer.CloseServer
Set dsoServer = Nothing

See Also

clsAggregationDimension
clsCubeDimension
clsPartitionDimension
Dimension Interface
Properties, Dimension Interface

The **Dimension** interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AggregationUsage</strong></td>
<td>Specifies how aggregations are to be designed for a dimension.</td>
</tr>
<tr>
<td><strong>AllowSiblingsWithSameName</strong></td>
<td>Indicates whether two or more children of the same parent member can have the same name.</td>
</tr>
<tr>
<td><strong>AreMemberKeysUnique</strong></td>
<td>Indicates whether member keys are unique for all members in the dimension.</td>
</tr>
<tr>
<td><strong>AreMemberNamesUnique</strong></td>
<td>Indicates whether member names are unique for all members in the dimension.</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
</tr>
<tr>
<td><strong>DataMemberCaptionTemplate</strong></td>
<td>Contains a template string that is used to generate captions for system-generated data members.</td>
</tr>
<tr>
<td><strong>DataSource</strong></td>
<td>The data source object of a dimension object.</td>
</tr>
<tr>
<td><strong>DefaultMember</strong></td>
<td>Defines the default member of the dimension.</td>
</tr>
<tr>
<td><strong>DependsOnDimension</strong></td>
<td>Names a dimension to which the current dimension is related.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of a dimension.</td>
</tr>
<tr>
<td><strong>DimensionType</strong></td>
<td>Returns an enumeration constant identifying the specific type of dimension.</td>
</tr>
<tr>
<td><strong>EnableRealTimeUpdates</strong></td>
<td>Indicates whether or not the dimension supports real-time updates.</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>The SQL FROM clause for a dimension.</td>
</tr>
<tr>
<td><strong>IsChanging</strong></td>
<td>Indicates whether members and/or levels are expected to change on a regular basis.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>Indicates whether dimension writebacks are available to client applications that have appropriate permissions.</td>
</tr>
<tr>
<td>IsShared</td>
<td>Indicates whether a dimension can be shared among cubes.</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>Indicates whether an object is temporary.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether a dimension structure is valid.</td>
</tr>
<tr>
<td>IsVirtual</td>
<td>Indicates whether a dimension is virtual.</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the dimension is visible to clients.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>The SQL JOIN clause for a dimension.</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time when a dimension was last processed.</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>A user-specified date. It is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td>MembersWithData</td>
<td>Determines which members in a dimension can have associated data in the fact table.</td>
</tr>
<tr>
<td>Name</td>
<td>The dimension name.</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the dimension object in the Dimensions collection of its parent object.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent MDStore object.</td>
</tr>
<tr>
<td>SourceTable</td>
<td>Returns the name of the source table for the dimension.</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>Returns the alias of the source table for the dimension.</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>Restricts members included in a dimension.</td>
</tr>
<tr>
<td>State</td>
<td>Returns an enumeration constant that indicates the difference between the dimension object referenced by the client application and the corresponding dimension on the Analysis server.</td>
</tr>
</tbody>
</table>
**StorageMode**
Determines the method of storing dimension contents.

**SubClassType**
Returns an enumeration constant that identifies the subclass type of the object.

---

**Access Cross-Reference**

The following table shows whether the property is read/write (R/W), read-only (R), or not applicable (n/a) for different objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>clsDatabase dimension</th>
<th>clsCube dimension</th>
<th>clsPartition dimension</th>
<th>clsAggregation dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregationUsage</td>
<td>n/a</td>
<td>R/W</td>
<td>R</td>
<td>n/a</td>
</tr>
<tr>
<td>AllowSiblingsWithSameName</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>AreMemberKeysUnique</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>AreMemberNamesUnique</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>ClassType</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>DataMemberCaptionTemplate</td>
<td>R/W*</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>DataSource</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>DefaultMember</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>DependsOnDimension</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Description</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>n/a</td>
</tr>
<tr>
<td>DimensionType</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>EnableRealTimeUpdates</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>FromClause</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
<tr>
<td>IsChanging</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsShared</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsValid</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsVirtual</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsVisible</td>
<td>n/a</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Property</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>JoinClause</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LastProcessed**</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>MembersWithData</td>
<td>R/W*</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Name</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Parent</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>SourceTable</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>n/a</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>n/a</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>State</td>
<td>R</td>
<td>n/a</td>
<td>n/a</td>
<td>R</td>
</tr>
<tr>
<td>StorageMode</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>SubClassType</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

* This property is read/write only for objects of `SubClassType sbclsParentChild`.

** The `LastProcessed` property of objects that implement the `Dimension` interface are not initialized until the dimension is processed for the first time. An error is raised if `LastProcessed` is accessed when the `State` property of the dimension object equals `olapStateNeverProcessed`.

### See Also

**Dimension Interface**
AggregationUsage (Dimension Interface)

The AggregationUsage property of the Dimension interface specifies how aggregations are to be designed for the dimension levels.

Applies To

- clsCubeDimension
- clsPartitionDimension

Data Type

DimensionAggUsageTypes

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCubeDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

When aggregations are designed for a dimension, the value of this property determines which levels of a dimension may have aggregations created for them. The following table describes how the value of this property affects the consideration of levels for aggregation.

<table>
<thead>
<tr>
<th>Aggregation usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimAggUsageCustom</td>
<td>Aggregations are created only for those levels whose EnableAggregations property is set to...</td>
</tr>
<tr>
<td><strong>dimAggUsageDetailsOnly</strong></td>
<td>Aggregations are created only for the lowest level in the dimension.</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>dimAggUsageStandard</strong></td>
<td>All levels are considered by the aggregation design algorithm.</td>
</tr>
<tr>
<td><strong>dimAggUsageTopOnly</strong></td>
<td>Aggregations are created only for the top (All) level.</td>
</tr>
<tr>
<td><strong>dimAggUsageTopAndDetailsOnly</strong></td>
<td>Aggregations are created only for the top (All) and lowest levels in the dimension.</td>
</tr>
</tbody>
</table>

**Note**  For virtual dimensions, **AggregationUsage** is read-only and automatically set to **dimAggUsageStandard**. An error is raised if you attempt to set this property on a virtual dimension.

**Example**

The following example changes the aggregation behavior of the Product dimension for the Sales cube in the **FoodMart 2000** database to create aggregations only for the topmost level, and then it updates and reprocesses the Sales cube:

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore
Dim dsoDim As DSO.Dimension

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"

' Open FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Open the Sales cube.
Set dsoCube = dsoDB.MDStores("Sales")
```
'Open the Product dimension in the Sales cube.
Set dsoDim = dsoCube.Dimensions("Product")

'Set the dimension to create aggregations only
'for the topmost (or All) level.
dsoDim.AggregationUsage = dimAggUsageTopOnly

'Update the cube.
dsoCube.Update

'Process the cube.
dsoCube.Process processFull

'Clean up.
Set dsoDim = Nothing
Set dsoCube = Nothing
Set dsoDB = Nothing
dsoServer.CloseServer
Set dsoServer = Nothing

See Also

Dimension Interface
EnableAggregations
**AllowSiblingsWithSameName (Dimension Interface)**

The `AllowSiblingsWithSameName` property of the `Dimension` interface determines whether children of a single member in a hierarchy can have identical names.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

Boolean

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

If this property is set to True, different members of a dimension can have identical names and positions. The members are distinguished by their key values. For example, two John Smiths may live in Seattle. In this case, the members are siblings because they are children of the same parent member, Seattle. If you set this property to True, the Analysis server disambiguates the
members using their key values rather than raising an error.

**Example**

The following example creates a dimension called Customers and sets its `AllowSiblingsWithSameName` property to True:

'Assume an object (dsoDimCust) of ClassType clsDatabase exists.
Dim dsoDimCust As DSO.Dimension
Set dsoDimCust = dsoDB.Dimensions.AddNew("Customer")
DsoDimCust.AllowSiblingsWithSameName = TRUE

**See Also**

[Dimension Interface]
AreMemberKeysUnique

The *AreMemberKeysUnique* property of the *Dimension* interface indicates whether member keys are unique throughout the dimension and whether these members can be referred to by their keys.

**Applies To**

- *clsAggregationDimension*
- *clsCubeDimension*
- *clsDatabaseDimension*
- *clsPartitionDimension*

**Data Type**

Boolean

**Access**

Access depends on the value of the *ClassType* property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

This property can be set to True only if the *AreMemberKeysUnique* property is set to True for all of the levels in the dimension.

This property can be used in two situations:

- When you know that the position of a member (and that of the children
of the member) may move within a dimension hierarchy. You can make the method for generating unique names more consistent by using this property to determine whether or not a member, name, or key is guaranteed to be unique within the dimension (or level).

- When you need to improve server performance. Data can be more easily validated if you use this property to inform the server that a dimension key is not unique within the dimension or level.

**Example**

' Assume the existence of a database dimension named dsoDim.

```plaintext
dsoDim.AreMemberKeysUnique = True
```

**See Also**

- [AreMemberNamesUnique](#)
- [AreMemberKeysUnique](#)
- [Dimension Interface](#)
AreMemberNamesUnique (Dimension Interface)

The `AreMemberNamesUnique` property of the `Dimension` interface determines whether member names are unique throughout the dimension.

Applies To

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

Data Type

Boolean

Access

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

This property can be set to True only if the `AreMemberNamesUnique` property is set to True for all of the levels in the dimension.

If `AreMemberNamesUnique` is set to False, each member name is assumed to be unique only among the siblings of that member. In this case, the unique name of a member includes the names of its ancestors to ensure uniqueness throughout
the dimension. If `AreMemberNamesUnique` is set to True, each member name is assumed to be unique throughout the entire dimension. In this case, the unique name of the member includes the dimension name and the member name.

**Note** Setting `AreMemberNamesUnique` to True is important in changing dimensions to preserve the identity of members in calculated members on other persistent expressions. For example, if a product [Product 1] is moved from [Category A] to [Category B], the old name [Products].[Category A].[Product 1] is no longer valid. However, if `AreMemberNamesUnique` is set to True, the unique name remains [Products].[Product 1] and is valid before and after the change.

The following table provides examples for each setting of `AreMemberNamesUnique`.

<table>
<thead>
<tr>
<th>AreMemberNamesUnique</th>
<th>Member names</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>[Product].[All Products].[Drink].[Beverages]</td>
</tr>
<tr>
<td></td>
<td>[Time].[1997].[Q1].[1/1/1997]</td>
</tr>
<tr>
<td></td>
<td>[Regions].[All Regions].[Asia]</td>
</tr>
<tr>
<td>True</td>
<td>[Product].[Beverages]</td>
</tr>
<tr>
<td></td>
<td>[Time].[1/1/1997]</td>
</tr>
<tr>
<td></td>
<td>[Regions].[Asia]</td>
</tr>
</tbody>
</table>

**Example**

' Assume the existence of a database dimension named dsoDim. dsoDim.AreMemberNamesUnique = True

**See Also**

- `AreMemberKeysUnique`
- `AreMemberNamesUnique`
- `Dimension Interface`
ClassType (Dimension Interface)

The `ClassType` property of the `Dimension` interface contains an enumeration constant that identifies the specific class type.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

**ClassTypes**

For dimension objects, `ClassType` returns one of the following:

- `clsDatabaseDimension`
- `clsCubeDimension`
- `clsPartitionDimension`
- `clsAggregationDimension`

**Access**

Read-only

**Example**

Use the following code to obtain the class type of a dimension object:
Dim dsoDim As DSO.Dimension
Dim objClassType As DSO.ClassTypes

' Insert code here for setting the dsoDim
dimension object to a valid dimension.

' Get the ClassType property of the dimension.
objClassType = dsoDimDimensionObject.ClassType

' Check the class type.
Select Case objClassType
    Case clsDatabaseDimension
        ' Insert commands for a database dimension.
    Case clsCubeDimension
        ' Insert commands for a cube dimension or virtual cube dimension.
    Case clsPartitionDimension
        ' Insert commands for a partition dimension.
    Case clsAggregationDimension
        ' Insert commands for an aggregation dimension.
    Case Else
        ' Insert other commands.
End Select

See Also

Dimension Interface
DataMemberCaptionTemplate (Dimension Interface)

The **DataMemberCaptionTemplate** property of the **Dimension** interface contains a template string that is used to create captions for system-generated data members.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W*</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read/write for objects of ClassType clsDatabaseDimension with a SubClassType of sbclsParentChild or sbclsMining, read-only for all others.*

**Remarks**

This property is used in conjunction with the **MembersWithData** property. It is used to automatically generate captions for system-generated data members.

The data member captions are generated by substituting the caption of the
associated member for the asterisk (*) placeholder character in the property string. The default data member caption template is "(* Data)".

For example, the template "(* Salary)" produces the caption "(John Doe Salary)" for the system-generated data member associated with the member John Doe.

This property is ignored when the MembersWithData property is dataforLeafMembersOnly.

Example

The following code example constructs a template for data members that will supply captions similar to (Subtotals for John Doe):

' Assume the existence of a database dimension named dsoDim.
dsoDim.DataMemberCaptionTemplate = "(Subtotals for *)"

See Also

Dimension Interface

MembersWithData
Analysis Services Programming

**DataSource (Dimension Interface)**

The **DataSource** property of the **Dimension** interface specifies the source of the data to be used by a dimension object.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

`clsDataSource`

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td>R/W*</td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td>R**</td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td>R**</td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td>R</td>
</tr>
</tbody>
</table>

*Not implemented for virtual dimensions created in previous versions of Microsoft® SQL Server™ 2000 Analysis Services.*

**Example**

Use the following code to add a data source to a database object:

'Assume an object (dsoDB) of ClassType clsDatabase exists.
'Add a new data source to database.
Dim dsoDS as DSO.DataSource
Set dsoDS = dsoDB.DataSources.AddNew("FoodMart 2000")

dsoDS.Name = "FoodMart"
dsoDS.ConnectionString = "Provider=MSDASQL.1;" & 
"Persist Security Info=False;" & 
"Data Source=FoodMart; " & 
"Connect Timeout=15"

'Update the database.
dsoDB.Update

See Also

Dimension Interface
DefaultMember (Dimension Interface)

The DefaultMember property of the Dimension interface defines the default member of the dimension.

Applies To

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Data Type

String

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

DefaultMember is used to evaluate Multidimensional Expressions (MDX) expressions in which no context for the dimension is available. This property contains an MDX expression that evaluates to a single member of the dimension to which the property belongs. If blank, the Analysis server uses one of the members on the topmost level of the dimension as the default member. For example, if you define Redmond as the default member of the Geography
dimension, tuples such as (Sales, 1997) can be evaluated as (Sales, 1997, Redmond).

Example
The following code example sets the default member to Food for the Product dimension in the FoodMart 2000 database:

' Assume the existence of a database dimension named dsoDim.
dsoDim.DefaultMember = "Food"

See Also
Dimension Interface
Custom Rules in Dimension Security
Set Default Member Dialog Box
DependsOnDimension (Dimension Interface)

The **DependsOnDimension** property of the **Dimension** interface contains the name of a dimension to which the current dimension depends.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

The value of this property tells the Analysis server that the contents of the current dimension are dependent on the contents of another dimension. The Analysis server uses this property to more accurately predict the size of cubes and better optimize the design of aggregations.

Usually, a dimension is related to another dimension if the two represent
different aspects of the same entity. For example, if the Customers dimension contains ten customers and the Customer Gender dimension contains two genders, the cross product contains only ten customer-gender combinations (no customer is measured once as male and once as female). In this case, the Customer Gender dimension depends on the Customers dimension. In contrast, if there are five items in the Products dimension, the cross product of Products and Customers contains a maximum of fifty customer-product combinations. In this case, Customers and Products are independent dimensions.

The property has different implications for virtual dimensions (those with the IsVirtual property set to True). The DependsOnDimension property for a virtual dimension contains the name of the underlying source dimension. All of the member key columns in the virtual dimension must appear in the source dimension.

See Also

Dimension Interface

Dependent Dimensions

Virtual Dimensions
Description (Dimension Interface)

The **Description** property of the **Dimension** interface contains the description of a dimension object.

**Applies To**

- **clsCubeDimension**
- **clsDatabaseDimension**
- **clsPartitionDimension**

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

The primary mechanism for identifying the data in a dimension is the **Name** property. The purpose of the **Description** property is to provide additional descriptive information.

**Example**

Use the following code to set the description of a dimension object:

'Assume an object (dsoDim) of ClassType clsDatabaseDimension exist
dsoDim.Description = "Total dollar sales"

See Also

Dimension Interface
Name
DimensionType (Dimension Interface)

The **DimensionType** property of the **Dimension** interface contains an enumeration constant that identifies the specific type of dimension.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

**DimensionTypes**

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

The **DimensionType** property provides both the Analysis server and client applications with information about the contents of the dimension. For example, a client application may benefit from knowing that a dimension is based upon geography, and another on time. When accepting user inputs for these dimensions (say, in a filter statement), the client application can use a Map control for the Geography dimension and a Calendar control for the Time...
dimension.

**Example**

Use the following code to set the `DimensionType` property of a dimension object:

'DAssume an object (dsoDim) of ClassType clsDatabaseDimension exist
dsoDim.DimensionType = dimRegular

**See Also**

[Dimension Interface](#)
EnableRealTimeUpdates (Dimension Interface)

The EnableRealTimeUpdates property of the Dimension interface indicates whether or not the object supports real-time updates.

Applies To

clsAggregationDimension
clsCubeDimension
clsDatabaseDimension
clsPartitionDimension

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

To enable a clsDatabaseDimension object to support real-time updates for real-time cubes, the object must use a ROLAP partition (the StorageMode property of the clsDatabaseDimension object must be storeasROLAP) and a Microsoft® SQL Server™ 2000 data source.
See Also

Dimension Interface

Real-Time Cubes
FromClause (Dimension Interface)

The **FromClause** property of the **Dimension** interface contains the SQL FROM clause for the dimension.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Example**

A database contains the following tables:

- Sales_Facts
- Customer
- Product
- Product_Class
- Promotion
- Store
- Calendar

The following diagram illustrates the relationships of these tables.

Use the following code to create a dimension, assign a data source, and set the FromClause property:

```vbnet
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDS As DSO.DataSource
Dim dsoDim As DSO.Dimension

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"
```
'Open TestDB database.
Set dsoDB = dsoServer.MDStores("TestDB")

' Create a Product dimension for the database.
Set dsoDim = dsoDB.Dimensions.AddNew("Products")

' Create a data source.
Set dsoDS = dsoDB.DataSources.AddNew("FoodMart 2000")
dsoDS.ConnectionString = "Provider=MSDASQL.1;" & _
    "Persist Security Info=False;" & _
    "Data Source=FoodMart;" & _
    "Connect Timeout=15"

' Assign the data source to the dimension.
Set dsoDim.DataSource = dsoDS  'Set the dimension data source.
dsoDim.FromClause = "product"  'Set the source dimension table.

' Update the database.
dsoDB.Update

See Also

Dimension Interface
IsChanging (Dimension Interface)

The **IsChanging** property of the **Dimension** interface indicates whether the members and/or levels of the dimension are expected to change on a regular basis.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

Boolean

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W*</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

* For more information about read/write access, see Remarks later in this topic.

**Remarks**

For all **Dimension** objects, the **IsChanging** property always returns a certain value depending on other property settings for the object:

- If the **SubClassType** property is **sbclsLinked**, the **IsChanging** property
returns False.

- If the SubClassType property is sbclsParentChild or sbclsMining, or if the object is a virtual dimension, or if the StorageMode property is storeasROLAP, the IsChanging property returns True.

- If the object is a virtual dimension created by an earlier version of Microsoft® SQL Server™ 2000 Analysis Services, the property returns False.

For objects of ClassType clsDatabaseDimension, read/write access is further qualified by several other factors. The IsChanging property for objects of ClassType clsDatabaseDimension is read-only if:

- The SubClassType property is sbclsParentChild, sbclsMining, or sbclsLinked.
  -or-

- The StorageMode property is set to storeasROLAP.
  -or-

- The object of ClassType clsDatabaseDimension is a virtual dimension created using an earlier version of Analysis Services.

If IsChanging is set to True, levels other than the first and last of the dimension can be added, moved, deleted, or changed without requiring that you fully reprocess all cubes that use the dimension. Also, members that belong to any level other than the first or last level can be added, moved, deleted, or changed without reprocessing cubes that use the dimension.

For dimensions that are not virtual, the IsUnique property must be set to True for the last level of the dimension before the IsChanging property can be set to True.

In addition to these requirements, the value of the IsChanging property can restrict the acceptable values for the AggregationUsage property of a dimension that is not virtual. The following table shows the values that are valid for dimensions with the IsChanging property set to True.
<table>
<thead>
<tr>
<th>Aggregation usage</th>
<th>Dimensions with an (All) level</th>
<th>Dimensions without an (All) level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DimAggUsageCustom</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>DimAggUsageDetailsOnly</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>DimAggUsageStandard</td>
<td>valid</td>
<td>valid</td>
</tr>
<tr>
<td>DimAggUsageTopOnly</td>
<td>valid</td>
<td>n/a</td>
</tr>
<tr>
<td>dimAggUsageTopAndDetailsOnly</td>
<td>valid</td>
<td>n/a</td>
</tr>
<tr>
<td>DimAggUsageStandard</td>
<td>valid</td>
<td>valid</td>
</tr>
</tbody>
</table>

For more information about processing requirements for ROLAP dimensions, see [StorageMode](#).

**See Also**

- [Dimension Interface](#)
- [StorageMode](#)
IsReadWrite (Dimension Interface)

The **IsReadWrite** property of the **Dimension** interface determines whether dimension writebacks are available to end users who have been granted appropriate permissions.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

Boolean

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W*</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read/write for clsDatabaseDimension objects with a SubClassType of sbclsParentChild, read-only for all other clsDatabaseDimension objects.*

**Remarks**

Setting the **IsReadWrite** property to True allows end users who have been granted appropriate permissions to modify the contents of a parent-child dimension. They can add or remove dimension levels, members, and member properties.
A write-enabled dimension cannot be added to a cube that contains a remote partition. An error occurs if you try to:

- Add a write-enabled dimension to a cube with at least one remote partition.

- Add a remote partition to a cube that contains at least one write-enabled dimension.

- Set the `IsReadWrite` property to True for a dimension used by a cube that has a remote partition.

**See Also**

[Dimension Interface]
**IsShared (Dimension Interface)**

The **IsShared** property of the **Dimension** interface indicates whether a dimension object can be shared among multiple **MDStore** objects.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

Boolean

**Access**

Read-only

**Remarks**

DSO sets the **IsShared** property by reading the **Name** property of the **Dimension** object. A dimension that is private to a cube has a name of the form *MyCube^MyDim*, where *MyCube* is the cube that contains the dimension. A dimension object that can be shared does not have the cube name as a prefix.

The value of the **IsShared** dimension of a database property is inherited by the associated cubes (and their partitions, aggregations, and virtual cubes).

**Example**

Use the following code to determine whether a dimension object can be shared with other dimension objects:
'Assume an object (dsoDim) of ClassType clsDatabaseDimension exists.
Dim bShared As Boolean
bShared = dsoDim.IsShared
If bShared Then
    'Insert code for a shared dimension.
Else
    'Insert code for a private dimension.
End If

See Also

Dimension Interface
IsTemporary (Dimension Interface)

The **IsTemporary** property of the **Dimension** interface indicates whether an object is temporary. Temporary objects are local to the session in which they are created, cannot be saved, and are not available to other users. To create a temporary dimension object, preface the name with the tilde (~) character.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

Boolean

**Access**

Read-only

**Remarks**

A temporary object is not stored in the repository and is not available to other users. Temporary objects persist only during the session in which they are created, unless they are renamed or cloned to another object that has the same class type. Objects that are subordinate to a temporary object, such as levels for a dimension, internally inherit the **IsTemporary** setting of the parent object.

**Note** Only temporary objects can be renamed by changing the **Name** property. Removing the tilde character from the name of a temporary object means that it is no longer temporary and prevents any subsequent renaming of the object. Also, executing the **Update** method of a temporary object has no effect. The object is not saved to the repository until it is renamed without the tilde prefix.
**Example**

Use the following code to create a temporary dimension and then make it permanent:

' Assume an object (dsoCube) of ClassType clsCube exists.
Dim tmpDim As DSO.Dimension
' Create a temporary dimension, using the tilde character in
' the dimension name.
Set tmpDim = dsoCube.Dimensions.AddNew("~MyDim")

' Add levels, add member properties, process, and so on.

' If you want to save the dimension permanently,
' drop the tilde prefix.
tmpDim.Name = "MyDim"
' Update the dimension.
tmpCube.Update

**See Also**

Dimension Interface
IsValid (Dimension Interface)

The **IsValid** property of the **Dimension** interface indicates whether the structure of a dimension object is valid. A structure is valid if it is fully and correctly defined. For example, a dimension object whose data source has not been defined is not valid.

**Applies To**
- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**
Boolean

**Access**
Read-only

**Remarks**
Depending on the value of the **ClassType** property of the object, the **IsValid** property checks different structure elements to confirm validity, as described in the following table.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseDimension</td>
<td>The <strong>Name</strong> and <strong>Parent</strong> properties; the database</td>
</tr>
<tr>
<td>clsCubeDimension</td>
<td>The <strong>Name</strong> and <strong>Parent</strong> properties; the source cube; the dimension and levels</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>The <strong>Name</strong> and <strong>Parent</strong> properties</td>
</tr>
<tr>
<td>clsAggregationDimension</td>
<td>The <strong>Name</strong> and <strong>Parent</strong> properties; the <strong>IsValid</strong></td>
</tr>
</tbody>
</table>
Example

Use the following code to determine whether the structure of a dimension object is valid:

'Assume an object (dsoDim) of ClassType clsDimension exists.
Dim bValid As Boolean
bValid = dsoDim.IsValid
If bValid Then
   'Insert code to process a valid dimension.
Else
   'Something is not valid - handle errors.
End If

See Also

Dimension Interface
IsVirtual (Dimension Interface)

The IsVirtual property of the Dimension interface indicates whether the dimension is virtual.

Applies To

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

A virtual dimension is based upon the contents of another dimension. The DependsOnDimension property contains the name of source dimension of the virtual dimension.

For objects of ClassType clsDatabaseDimension, the IsVirtual property cannot be set to True if the SubClassType is sbclsParentChild or sbclsMining, or if
the **IsVirtual** property is set to True for any level of the dimension.

**See Also**

[DependsOnDimension](#)

[Dimension Interface](#)
Analysis Services Programming

**IsVisible (Dimension Interface)**

The `IsVisible` property of the `Dimension` interface determines whether the dimension is visible to client applications. Calculated members can still refer to the dimension.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsPartitionDimension`

**Data Type**

Boolean

The default value is True.

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsCubeDimension</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

This property is used to hide a dimension from the client application. For example, you might decide that it is useful to include a gender virtual dimension in a cube but hide the customer source dimension on which it is based.

**See Also**
Dimension Interface
Analysis Services Programming

JoinClause (Dimension Interface)

The **JoinClause** property of the **Dimension** interface contains the SQL JOIN clause for the dimension.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W*</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R/W</td>
</tr>
</tbody>
</table>

* Read-only for objects of **ClassType clsDatabaseDimension** that represent a virtual dimension created in an earlier version of Microsoft® SQL Server™ 2000 Analysis Services.

**Remarks**

For objects of **ClassType clsDatabaseDimension**, if it represents a virtual dimension created in an earlier version of Analysis Services, the **JoinClause** property returns an empty string.

**Example**
A database contains the following tables:

- **Sales_Facts**
- **Customer**
- **Product**
- **Product_Class**
- **Promotion**
- **Store**
- **Calendar**

The following diagram illustrates the relationships of these tables.

Use the following code to return the **JoinClause** of the dimension object:

'Assume an object (dsoDim) of ClassType clsDimension exists
'and is associated with the Product dimension.
Dim strJoinClause As String
strJoinClause = dsoDim.JoinClause

"The immediate window displays the following: "
"product"."SKU"="product_class"."SKU"
Debug.Print " Join Clause: " & strJoinClause

See Also

Dimension Interface
LastProcessed (Dimension Interface)

The `LastProcessed` property of the `Dimension` interface contains the date and time when a dimension was last processed.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

Date

**Access**

Read-only

**Remarks**

If the value of the `State` property is `olapStateNeverProcessed`, the `LastProcessed` property for an object is undefined, and it raises an error.

**Example**

Use the following code to determine when a dimension object was last processed:

'Assume an object (dsoDim) of ClassType clsDimension exists. If dsoDim.State <> olapStateNeverProcessed Then
  If dsoDim.LastProcessed < Date Then
    'Insert code to process the dimension.
End If
End If

See Also

Dimension Interface
LastUpdated
LastUpdated (Dimension Interface)

The **LastUpdated** property of the **Dimension** interface is not used by Microsoft® SQL Server™ 2000 Analysis Services. You can set this to any date/time value you want. For example, you can use it to indicate when the source data was last changed.

**Applies To**

- **clsAggregationDimension**
- **clsCubeDimension**
- **clsDatabaseDimension**
- **clsPartitionDimension**

**Data Type**

Date

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseDimension</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

The **LastUpdated** property is not automatically set by any Decision Support Objects (DSO) method. It is provided as a means for client applications to indicate a date or time that can assist in validating information. For example, a date of 12/31/96 may mean that the information stored in a cube is not valid after
December of 1996.

**Example**

Use the following code to update an out-of-date dimension:

'Assume an object (dsoDim) of ClassType clsDatabaseDimension exist
If dsoDim.LastUpdated < date Then
   'Insert ode to update dimension.
End If

**See Also**

[Dimension Interface](#)

[LastProcessed](#)
**MembersWithData (Dimension Interface)**

The `MembersWithData` property of the `Dimension` interface determines which members in a dimension can have associated data in the fact table.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

`MembersWithDataValues`

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td><code>R/W*</code></td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td><code>R</code></td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td><code>R</code></td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td><code>R</code></td>
</tr>
</tbody>
</table>

*Read/write for objects of `ClsDatabaseDimension` with a `SubClassType` of `sbclsParentChild` or `sbclsMining`, read-only for all other objects.*

**Remarks**

When a nonleaf member has associated data in the fact table, a processing error occurs if this property is equal to `dataforLeafMembersOnly`. Otherwise, that nonleaf data is associated with a system-generated leaf member. Depending on the property setting, this system-generated data member may or may not be visible on query axes and in schema rowsets.
By default, the data for nonleaf members is aggregated with the values of the regular children of that member. This behavior can be overridden by defining a custom rollup function for the member (or the entire level). The system-generated data member is always available through a Multidimensional Expressions (MDX) statement by way of the `<Member>.DataMember` syntax. (This syntax returns the original member if that member is already a leaf.)

Consider the example of a cube based on human resources data. If a parent-child dimension contains all of the employees of an organization, a problem will exist in tracking the salaries for people at higher levels of the organization's hierarchy. Their salaries will be an aggregate of the salaries of the people who report to them (that is, the value of their member will be the aggregate of all the children of that member). By setting the value of this property to `DataForNonLeafMembersHidden`, you can build a cube in which nonleaf members of the dimension will also have data.

Using this method may present a problem, however, because the numbers for nonleaf nodes no longer add up as might be expected. You can solve the problem by setting this property to `DataForNonLeafMembersVisible`. In this scenario, the members at each level contain the correct numbers for the aggregation, but the data for that individual member is available under the `DataMember` property.

**See Also**

- [Dimension Interface](#)
- [DataMemberCaptionTemplate](#)
Name (Dimension Interface)

The Name property of the Dimension interface contains the name of a dimension object.

Applies To

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Data Type

String

Access

Read/write (read-only after the object has been named)

Remarks

The primary mechanism for identifying the data in a dimension is the Name property. The purpose of the Description property is to provide additional descriptive information.

Example

Use the following code to print the names of the dimensions in a database:

'Assume an object (dsoDB) of ClassType clsDatabase exists.
Dim dsoDim As DSO.Dimension
Dim dimCounter As Integer
For dimCounter = 1 To dsoDB.Dimensions.Count
Set dsoDim = dsoDB.Dimensions(dimCounter)
Debug.Print dsoDim.Name
Next dimCounter

See Also

Dimension Interface
Description
IsShared
IsTemporary
**OrdinalPosition (Dimension Interface)**

The **OrdinalPosition** property of the **Dimension** interface contains the ordinal position of the dimension object within the **Dimensions** collection of its parent object.

**Note** The ordinal position is 1 for all dimension objects in the collection of dimensions for a database. The position of a dimension within the collection is significant only when it is used in a cube, partition, or aggregation.

The **OrdinalPosition** property for dimensions is different from the **OrdinalPosition** property for levels. Dimensions are ordered by the time sequence in which you add them to their parent object. The ordinal position of levels determines the hierarchy of the dimension.

**Applies To**

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

**Data Type**

Integer

**Access**

Read-only

**Example**

Use the following code to identify the first dimension in the **Dimensions** collection:
'Assume an object (dsoDim) of ClassType clsCubeDimension exists. If dsoDim.OrdinalPosition = 1 Then
    'Insert code to handle the first dimension in the cube.
Else
    'Insert code to handle other dimensions.
End If

See Also

Dimension Interface
Analysis Services Programming

Parent (Dimension Interface)

The **Parent** property of the **Dimension** interface contains a reference to the parent **MDStore** object.

**Applies To**

- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

**MDStore**

**Access**

Read-only

**Remarks**

The value of the **ClassType** property of the parent object depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Dimension object ClassType</th>
<th>Parent object ClassType</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td><code>clsDatabase</code></td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td><code>clsCube</code></td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td><code>clsPartition</code></td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td><code>clsAggregation</code></td>
</tr>
</tbody>
</table>

**Example**

Use the following code to obtain the parent of a dimension object:
'Assume an object (dsoDim) of ClassType clsDimension exists.
Dim objClassType As ClassTypes
objClassType = dsoDim.Parent.ClassType
Select Case objClassType
    Case clsDatabase
        'Insert code for the database parent object.
    Case clsCube
        'Insert code for the cube/virtual cube parent object.
    Case clsPartiton
        'Insert code for the partition parent object.
    Case clsAggregation
        'Insert code for the aggregation parent object.
End Select

See Also

Dimension Interface
SourceTable (Dimension Interface)

The SourceTable property of the Dimension interface contains the name of a dimension object's primary source table.

Applies To

- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Data Type

String

Access

Read-only

Remarks

This property returns the name of the source table associated with the lowest level in the dimension. For example, if your cube has a dimension called Product, and the dimension has levels called ProductCategory, ProductSubCategory, and ProductName, with ordinal positions 1, 2, and 3, respectively, the SourceTable property for the Product dimension returns the name of the source table associated with level ProductName.

Example

Use the following code to obtain the name of the source table associated with level ProductName in dimension Product:

'Assume an object (dsoDim) of ClassType clsDimension exists.
If dsoDim.SourceTable = "Product" Then
'Insert code to handle the dimension.
End If

See Also

Dimension Interface
SourceTableAlias (Dimension Interface)

The SourceTableAlias property of the Dimension interface contains the alias of the source table for the dimension.

Applies To

- clsDatabaseDimension
- clsCubeDimension
- clsPartitionDimension

Data Type

String

Access

Read-only

Remarks

If the source table has no alias, the contents of this property are identical to the SourceTable property.

See Also

- Dimension Interface
SourceTableFilter (Dimension Interface)

The `SourceTableFilter` property of the `Dimension` interface restricts the members included in a dimension.

**Applies To**
- `clsAggregationDimension`
- `clsCubeDimension`
- `clsDatabaseDimension`
- `clsPartitionDimension`

**Data Type**

String

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseDimension</code></td>
<td>R/W*</td>
</tr>
<tr>
<td><code>ClsCubeDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsPartitionDimension</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsAggregationDimension</code></td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

Use this property to filter dimension members. For example, suppose you want to build a sales cube that only contains customer information from the state of California. You can set this property equal to the following SQL expression to solve this problem:

"Customer"."State" = 'California'
**Note** The expression must be entered in terms of the SQL dialect used on the source server, not Multidimensional Expressions (MDX). Thus "Customer"."State" refers to the **State** column of the **Customer** table in a Microsoft® SQL Server™ 2000 database.

The **SourceTableFilter** property contains an SQL expression such as those found in SQL WHERE clauses. That is, it must contain an SQL expression that evaluates to either True or False.

For virtual dimensions, this property is always read-only and empty.

**See Also**

[Dimension Interface](#)
State (Dimension Interface)

The State property of the Dimension interface contains an enumeration constant that indicates the difference between the dimension object referenced by the client application and the corresponding dimension on the Analysis server.

Applies To

clsDatabaseDimension

Data Type

OlapStateTypes

Access

Read-only

Remarks

The State property indicates the current status of an object of ClassType clsDimension. It is also used to determine whether the dimension needs to be processed. The following table lists the possible values for the State property.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlapStateNeverProcessed</td>
<td>The database dimension has never been processed.</td>
</tr>
<tr>
<td>olapStateStructureChanged</td>
<td>The structure of the database dimension has changed.</td>
</tr>
<tr>
<td>olapStateMemberPropertiesChanged</td>
<td>The member properties have changed.</td>
</tr>
<tr>
<td>olapStateSourceMappingChanged</td>
<td>The source mappings for the database dimension have changed.</td>
</tr>
<tr>
<td>OlapStateCurrent</td>
<td>The database dimension has been processed and is current.</td>
</tr>
</tbody>
</table>
See Also

Dimension Interface
StorageMode (Dimension Interface)

The StorageMode property of the Dimension interface determines how the contents of a dimension are stored.

Applies To

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Data Type

StorageModeValues

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseDimension</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsCubeDimension</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationDimension</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read-only for virtual dimensions and objects with a SubClassType of sbclsLinked, sbclsMining, or sbclsParentChild.

Remarks

If the StorageMode property is set to storeasMOLAP, the contents of the dimension are processed and stored on the Analysis server. If StorageMode is set to storeasROLAP, the dimension contents are not read during processing; they are left in the source relational database and retrieved as required. ROLAP storage mode provides for improved scalability of dimensions at the expense of
slower query performance.

The StorageMode property uses one of the constants, defined in the StorageModeValues enumeration, shown in the following table.

<table>
<thead>
<tr>
<th>Storage mode</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>storeasMOLAP</td>
<td>Dimension members are read during processing and are stored in the Analysis server.</td>
</tr>
<tr>
<td>storeasROLAP</td>
<td>Dimension members are not read during processing and are left in the relational data source.</td>
</tr>
</tbody>
</table>

Before the StorageMode property for a dimension can be set to storeasROLAP, the AreMemberKeysUnique property must be set to True for the last level in the dimension.

When the StorageMode property is set to storeasROLAP, the value of the IsChanging property for the dimension automatically becomes read-only and True.

**CAUTION** Any changes to the relational source table of a ROLAP dimension must be followed by an immediate reprocessing of the dimension. Failure to do so may result in inconsistent results to queries of the cubes that use the dimension. To ensure correct processing of both the source table and the dimension, use nested transactions to link the two changes together. Because an incremental update is all that is necessary, use the processRefreshData option with the Process method to update the dimension.

To set the StorageMode property to storeasROLAP, the Grouping property for all levels in the dimension must be set to None. Member groups are not supported for ROLAP dimensions.

**See Also**

Dimension Interface
SubClassType (Dimension Interface)

The SubClassType property of the Dimension interface contains an enumeration constant that identifies the subclass type of the object.

Applies To

- clsAggregationDimension
- clsCubeDimension
- clsDatabaseDimension
- clsPartitionDimension

Data Type

SubClassTypes

Access

Read-only

Remarks

For objects of ClassType clsDimension, SubClassType can be sbclsRegular or sbclsParentChild.

See Also

Dimension Interface
Level Interface

Levels describe the hierarchy within a dimension from the highest (most precalculated) to the lowest (most detailed) levels of data.

The following table shows an example of level positions of a time dimension in which the most detailed (day) values are included in the next level (week), which are, in turn, included in the next level (quarter), and so on. If the Year level has 4 members, 1994 through 1997, then the Quarter level has 4 members for each year, and the Week level has 52 members for each year.

<table>
<thead>
<tr>
<th>Level</th>
<th>Position in hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>1</td>
</tr>
<tr>
<td>Quarter</td>
<td>2</td>
</tr>
<tr>
<td>Week</td>
<td>3</td>
</tr>
<tr>
<td>Day</td>
<td>4</td>
</tr>
</tbody>
</table>

In Decision Support Objects (DSO), objects that implement the Level interface have one of the following ClassType property values.

<table>
<thead>
<tr>
<th>ClassType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregationLevel</td>
<td>The levels contained within an aggregation dimension objects levels collection</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>The levels of all dimensions assigned to a cube</td>
</tr>
<tr>
<td>clsDatabaseLevel</td>
<td>The levels of all dimensions within a database</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>The levels of all of the dimensions contained within a partition</td>
</tr>
</tbody>
</table>

Additionally, the SubClassType property for level objects can have the following values.
<table>
<thead>
<tr>
<th>Subclass type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbclsRegular</td>
<td>A regular level</td>
</tr>
<tr>
<td>sbclsParentChild</td>
<td>A parent-child level</td>
</tr>
</tbody>
</table>

The **Level** interface provides a number of properties to manipulate these objects. For more information about levels and other objects, see *Introducing Decision Support Objects*.

Differences in the implementation of the **Level** interface exist between the derived objects. Some objects provide read-only access to a few **Level** properties, while others implement a subset of the properties contained in the interface. For example, a **clsDatabaseLevel** object allows read and write access to its **LevelType** property, whereas, for a **clsPartitionLevel** object, access to this property is read-only. For more information about the **Level** interface properties and their applicability to the associated objects, see *Collections, Level Interface* and *Properties, Level Interface*.

There are no methods associated with the **Level** interface.

**Parent-Child Levels**

Parent-child levels appear only in parent-child dimensions (those whose **SubClassType** is **sbclsParentChild**). Unlike regular levels that are constructed using predefined member columns, parent-child levels function as templates for hierarchies that are dynamically built from source tables with parent-child relationships. Parent-child levels identify the **MemberKeyColumn**, **MemberNameColumn**, and **ParentKeyColumn** properties of the data source, which are used to dynamically build variable-level hierarchies.

**Applies To**

- **clsAggregationLevel**
- **clsCubeLevel**
- **clsDatabaseLevel**
- **clsPartitionLevel**
See Also

Dimension Interface
Levels and Members
MemberKeyColumn
Parent-Child Dimensions
SubClassType
SubClassTypes
Collections, Level Interface

The **Level** interface supports the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>MemberProperties</td>
<td>The collection of objects of <strong>ClassType</strong> <strong>clsMemberProperty</strong></td>
</tr>
</tbody>
</table>

Access Cross-Reference

The following table shows whether the collection is read/write (R/W), read-only (R), or not applicable (n/a) for different objects.

<table>
<thead>
<tr>
<th>Collection</th>
<th>clsDatabase Level</th>
<th>clsCubeLevel</th>
<th>clsPartition Level</th>
<th>clsAggregation Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>MemberProperties</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>n/a</td>
</tr>
</tbody>
</table>

See Also

- **clsMemberProperty**
- **Level Interface**
Properties, Level Interface

The **Level** interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreMemberKeysUnique</td>
<td>Indicates whether the members of a level are uniquely identified by their member key column</td>
</tr>
<tr>
<td>AreMemberNamesUnique</td>
<td>Indicates whether the members of a level are uniquely identified by their member name column</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td>ColumnSize</td>
<td>The size (in bytes) of members in the level member key column</td>
</tr>
<tr>
<td>ColumnType</td>
<td>The data type of the level member key column in an aggregation table</td>
</tr>
<tr>
<td>CustomRollUpColumn</td>
<td>Contains the name of the column that contains member-specific rollup instructions</td>
</tr>
<tr>
<td>CustomRollUpExpression</td>
<td>Contains a Multidimensional Expressions (MDX) expression used to override the default rollup mode</td>
</tr>
<tr>
<td>CustomRollUpPropertiesColumn</td>
<td>Contains the name of the column that contains member-specific rollup properties</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the level</td>
</tr>
<tr>
<td>EnableAggregations</td>
<td>Determines whether aggregations can be created for a level in a dimension whose AggregationUsage property is set to dimAggUsageCustom</td>
</tr>
<tr>
<td>EstimatedSize</td>
<td>The estimated number of members in the level</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>The SQL FROM clause for a level</td>
</tr>
<tr>
<td><strong>Grouping</strong></td>
<td>Indicates the type of grouping used by the Analysis server</td>
</tr>
<tr>
<td><strong>HideMemberIf</strong></td>
<td>Indicates whether a member should be hidden from client applications</td>
</tr>
<tr>
<td><strong>IsDisabled</strong></td>
<td>Indicates whether the level is disabled</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the level structure is valid</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>Indicates whether the level is visible to client applications</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>The SQL JOIN clause for the level</td>
</tr>
<tr>
<td><strong>LevelNamingTemplate</strong></td>
<td>Defines how levels in a parent-child hierarchy are named</td>
</tr>
<tr>
<td><strong>LevelType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of level</td>
</tr>
<tr>
<td><strong>MemberKeyColumn</strong></td>
<td>The name of the column or expression that contains member keys</td>
</tr>
<tr>
<td><strong>MemberNameColumn</strong></td>
<td>The name of the column or expression that contains member names</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the level</td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>Specifies how the level should be ordered</td>
</tr>
<tr>
<td><strong>OrderingMemberProperty</strong></td>
<td>Specifies a member property used to determine the ordering of members</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the level in the collection of levels</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent Dimension object</td>
</tr>
<tr>
<td><strong>ParentKeyColumn</strong></td>
<td>Identifies the parent of a member in a parent-child hierarchy</td>
</tr>
<tr>
<td><strong>RootMemberIf</strong></td>
<td>Determines how the root member or members of a parent-child hierarchy are identified</td>
</tr>
<tr>
<td><strong>SkippedLevelsColumn</strong></td>
<td>Identifies the column that holds the number of empty levels between a member and its parent in a parent-child hierarchy</td>
</tr>
</tbody>
</table>
**SliceValue**  
The name of the level member used to define a partition slice

**SubClassType**  
Returns an enumeration constant identifying the subclass type of an object

**UnaryOperatorColumn**  
The name of the column that contains member-specific rollup instructions in the form of mathematical operators

### Access Cross-Reference

The following table shows whether the property is read/write (R/W), read-only (R), or not applicable (n/a) for different objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>clsDatabase Level</th>
<th>clsCube Level</th>
<th>clsPartition Level</th>
<th>clsAggregation Level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AreMemberKeysUnique</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>AreMemberNamesUnique</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>ColumnSize</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>ColumnType</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>CustomRollUpColumn</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>CustomRollUpExpression</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>CustomRollUpPropertiesColumn</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>EnableAggregations</strong></td>
<td>n/a</td>
<td>R/W</td>
<td>R</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>EstimatedSize</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Grouping</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>HideMemberIf</strong>**</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>IsDisabled</strong></td>
<td>n/a</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>LevelNaming</strong> Template†</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>LevelType</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>MemberKey Column</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>MemberName Column</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>OrderingMemberProperty</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>ParentKeyColumn†</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>RootMemberIf</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>SkippedLevelsColumn</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>SliceValue</strong></td>
<td>n/a</td>
<td>n/a</td>
<td>R/W</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>UnaryOperatorColumn</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

** This property does not apply to levels of **SubClassType sbclsParentChild** or **sbclsMining**.
†  This property applies only to levels of **SubClassType sbclsParentChild** or **sbclsMining**.

**See Also**

- [AggregationUsage](#)
- [Level Interface](#)
AreMemberKeysUnique (Level Interface)

The `AreMemberKeysUnique` property of the `Level` interface indicates whether the members of a level can be uniquely identified within the dimension by their member key column and without a reference to a higher level.

Applies To

- `clsAggregationLevel`
- `clsCubeLevel`
- `clsDatabaseLevel`
- `clsPartitionLevel`

Data Type

Boolean

Access

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsDatabaseLevel</code></td>
<td>R/W*</td>
</tr>
<tr>
<td><code>clsCubeLevel</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsPartitionLevel</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsAggregationLevel</code></td>
<td>R</td>
</tr>
</tbody>
</table>

For more information about read/write access, see Remarks later in this topic.

Remarks

For objects of `ClassType clsDatabaseLevel`, the `AreMemberKeysUnique` column is read-only if the object meets one of the following criteria:

- The level is the first level in the dimension, or the `LevelType` property of the object is `levAll`. 
The **Grouping** property of the object is **groupingAutomatic**.

The **AreMemberKeysUnique** property is always read-only for objects of **ClassType clsDatabaseLevel** objects with a **SubClassType** of **sbclsParentChild** or **sbclsMining**.

If the **AreMemberKeysUnique** property is True, the **MemberKeyColumn** property of the level uniquely identifies all level members. For example, the following diagram shows the hierarchy of a time dimension. Because the members of Level 3 - Quarters are duplicated under each year level, it is not possible to determine the exact time slice to query without also reading the corresponding value for Level 2 - Years. To uniquely identify the time period 1997 - Q1, the member key columns for Level 2 - Years and Level 3 - Quarters must be combined.

A similar example can be made for a geographic hierarchy. While the names of states or provinces are unique within a single country, one or more cities in different countries may share the same name. In this case, the city level does not have unique values.

**Example**

If a level for months of the year with members named Month1, Month2, ... Month12 is added to a dimension below a quarter level, the **AreMemberKeysUnique** property is set to False. This is because the values in **MemberKeyColumn** do not uniquely identify each member in the month level within the dimension.

' Assume an object (dsoDim) of **ClassType clsDimension** exists
Dim dsoLevel As DSO.Level
Set dsoLevel = dsoDim.Levels.AddNew("Months")
dsol.evel.AmMembekysUnique = False

See Also

Level Interface

MemberKeyColumn
AreMemberNamesUnique (Level Interface)

The `AreMemberNamesUnique` property of the `Level` interface indicates whether the names of members are unique throughout the level and then determines a naming scheme based on the indication.

Applies To

- `clsAggregationLevel`
- `clsCubeLevel`
- `clsDatabaseLevel`
- `clsPartitionLevel`

Data Type

Boolean

Access

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsDatabaseLevel</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>clsCubeLevel</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsPartitionLevel</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsAggregationLevel</code></td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

If `AreMemberNamesUnique` is False, each member name is assumed to be unique only among its siblings. In this case, the unique name for the member is constructed using the name of the member and the names of its ancestors. If the property for the level is True, each member name is assumed to be unique for the entire level. In this case, the unique name for the member is constructed using
only the dimension name, the level name, and the member name. Other properties, such as connection string and Registry settings, can influence the method by which unique member names are generated. This property is read-only and always True for the (All) level of a dimension (that is, the level that has a LevelType of levAll). Additionally, this property is read-only and always False for levels with automatic grouping (that is, a level that has a Grouping property of groupingAutomatic).

See Also

AreMemberNamesUnique
Level Interface
**ClassType (Level Interface)**

The **ClassType** property of the **Level** interface contains an enumeration constant that identifies the specific class type.

**Applies To**
- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

**ClassTypes**

For level objects, **ClassType** is set to one of the following values:

- clsDatabaseLevel
- clsCubeLevel
- clsPartitionLevel
- clsAggregationLevel

**Access**

Read-only

**Example**

Use the following code to return the class type of a level object and to determine which object class has been returned:
'Assume an object (dsoLevel) of ClassType clsLevel exists
Dim objClass As DSO.ClassTypes
objClassType = dsoLevel.ClassType
Select Case objClassType
    Case clsAggregationLevel
        ' Insert commands for an aggregation level.
    Case clsCubeLevel
        ' Insert commands for a cube level.
    Case clsDatabaseLevel
        ' Insert commands for a database level.
    Case clsPartitionLevel
        ' Insert commands for a partition level.
End Select

**See Also**

- [ClassTypes](#)
- [Level Interface](#)
**ColumnSize (Level Interface)**

The **ColumnSize** property of the **Level** interface contains the size (in bytes) of the members in the level aggregation column in an aggregation table.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

Integer

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read-only for virtual cube levels and cube levels of a shared dimension (a parent Dimension object whose **IsShared** property is True).*

**Remarks**

The **ColumnSize** property always returns 0 for (All) levels (that is, levels whose **LevelType** property is levAll) with a **SubClassType** of sbclsRegular, sbclsParentChild, sbclsLinked or sbclsMining, and for levels whose **LevelType** property is set to sbclsVirtual.
Set **ColumnSize** to be large enough to store the data type of the level. Integer values, for example, require a minimum of four bytes. If the level contains string values, find the length of the member with the longest string. Set **ColumnSize** greater than or equal to the length of that string multiplied by the byte size of an individual character. The **ColumnSize** property cannot be set to zero.

**Example**

Use the following code to specify a column size of 40:

'Assume an object (dsoLev) of ClassType clsDatabaseLevel exists
dsoLev.ColumnSize = 40

**See Also**

[ColumnType](#)

[Level Interface](#)
**ColumnType (Level Interface)**

The **ColumnType** property of the **Level** interface contains the data type of the level member key column.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

**ADODB.DataTypeEnum**

The **ColumnType** property is set to one of the following enumerated values.

<table>
<thead>
<tr>
<th>Column type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big integer</td>
<td>adBigInt</td>
</tr>
<tr>
<td>Binary</td>
<td>adBinary</td>
</tr>
<tr>
<td>Boolean</td>
<td>adBoolean</td>
</tr>
<tr>
<td>String (Unicode)</td>
<td>adBSTR</td>
</tr>
<tr>
<td>Char</td>
<td>adChar</td>
</tr>
<tr>
<td>Currency</td>
<td>adCurrency</td>
</tr>
<tr>
<td>Date</td>
<td>adDate</td>
</tr>
<tr>
<td>Date</td>
<td>adDBDate</td>
</tr>
<tr>
<td>Time</td>
<td>adDBTime</td>
</tr>
<tr>
<td>Timestamp</td>
<td>adDBTimeStamp</td>
</tr>
<tr>
<td>Decimal</td>
<td>adDecimal</td>
</tr>
<tr>
<td>Double</td>
<td>adDouble</td>
</tr>
<tr>
<td>Integer</td>
<td>adInteger</td>
</tr>
<tr>
<td>Numeric</td>
<td>adNumeric</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>Single</td>
<td>adSingle</td>
</tr>
<tr>
<td>Small integer</td>
<td>adSmallInt</td>
</tr>
<tr>
<td>Tiny integer</td>
<td>adTinyInt</td>
</tr>
<tr>
<td>Unsigned big integer</td>
<td>adUnsignedBigInt</td>
</tr>
<tr>
<td>Unsigned integer</td>
<td>adUnsignedInt</td>
</tr>
<tr>
<td>Unsigned small integer</td>
<td>adUnsignedSmallInt</td>
</tr>
<tr>
<td>Unsigned tiny integer</td>
<td>adUnsignedTinyInt</td>
</tr>
<tr>
<td>Text (Unicode)</td>
<td>adWChar</td>
</tr>
<tr>
<td>Text</td>
<td>adChar</td>
</tr>
</tbody>
</table>

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R/W**</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read-only for levels with automatic grouping (a level whose **Grouping** property is `groupingAutomatic`).

**Remarks**

The **ColumnType** property determines how the server will bind the member key column. This property must be set to a compatible type or processing the dimension or cube will result in an error.

The **ColumnSize** property always returns **adInteger** for levels with a **SubClassType** of `sbclsRegular`, `sbclsParentChild`, `sbclsLinked`, or `sbclsMining` that use automatic grouping (that is, the **Grouping** property is `groupingAutomatic`). Additionally, this property always returns **adVarChar** for levels with a **SubClassType** of `sbclsVirtual`.

**Example**
Use the following code to specify an integer **ColumnType**:

' Assume an object (dsoDim) of ClassType clsDimension exists
Set dsoLev = dsoDim.Levels.AddNew("Store Id")
dsoLev.MemberKeyColumn = """"store"""".""""store_number"""
dsoLev.ColumnSize = 4
dsoLev.ColumnType = adInteger
dsoLev.EstimatedSize = 24

**See Also**

- ColumnSize
- Level Interface
- Partition Storage
CustomRollupColumn (Level Interface)

The CustomRollupColumn property of the Level interface contains the name of the column that stores member-specific rollup instructions.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

The string values in this column should contain valid Multidimensional Expressions (MDX) expressions. If a column is empty, the corresponding member is calculated normally. If the formula in the column is invalid, a runtime error occurs when a cell value using the member is retrieved.

Order of evaluation among dimensions with custom rollups is determined by the
order of the dimensions in the **Dimensions** collection of the parent cube. Calculated members are always evaluated before custom rollups.

**Note** Because the DISTINCT COUNT aggregation function does not support custom aggregations, the use of this aggregation function in combination with the **CustomRollupExpression** and **CustomRollupColumn** properties is not supported. If a cube uses the DISTINCT COUNT aggregation function and any of the dimensions in that cube use either the **CustomRollupExpression** property or the **CustomRollupColumn** property then the cube is considered to be invalid. Processing such a cube will raise a validation error with an error code of mderrInvalidCubeDistinctCountWithCustomRollups.

**See Also**

[Level Interface](#)

[Custom Rollup Formulas and Custom Member Formulas](#)

[CustomRollupExpression (Level Interface)](#)

[CustomRollupPropertiesColumn (Level Interface)](#)
CustomRollupExpression (Level Interface)

The CustomRollupExpression property of the Level interface contains a Multidimensional Expressions (MDX) expression that is used to override the default rollup mode for the level.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

If the CustomRollupExpression property is blank, member values are rolled up normally using the aggregation function of the current measure. If the property contains an MDX expression, that expression is used to evaluate each member of the level. The formulas contained in this member property are resolved in the same way as regular calculated members.
**CustomRollupExpression** can be set on both database and cube levels. If set on both types of levels for a given cube, the **CustomRollupExpression** for the cube level overrides the **CustomRollupExpression** for the database level. Member-specific rollup expressions in the column specified by the **CustomRollupExpression** for the level always override this property.

A common usage scenario for this property involves tracking inventory over time. Inventory counts do not aggregate along time dimensions, but they do aggregate along other types of dimensions, such as geography or sales. By defining a **CustomRollupExpression** property to use the last member of its respective level within a time dimension, closing inventory counts can be rolled up over time.

**Note** Because the DISTINCT COUNT aggregation function does not support custom aggregations, the use of this aggregation function in combination with the **CustomRollupExpression** and **CustomRollupColumn** properties is not supported. If a cube uses the DISTINCT COUNT aggregation function and any of the dimensions in that cube use either the **CustomRollupExpression** property or the **CustomRollupColumn** property then the cube is considered to be invalid. Processing such a cube will raise a validation error with an error code of *mderrInvalidCubeDistinctCountWithCustomRollups*.

For more information about CREATE MEMBER, see [CREATE MEMBER Statement](#).

**See Also**

- [Custom Rollup Formulas and Custom Member Formulas](#)
- [Custom Rollup Operators](#)
- [CustomRollupColumn](#)
- [CustomRollupPropertiesColumn](#)
- [Level Interface](#)
CustomRollupPropertiesColumn (Level Interface)

The `CustomRollupPropertiesColumn` property of the `Level` interface is used to provide properties associated with the member formulas provided in the `CustomRollupColumn` property.

Applies To

- `clsAggregationLevel`
- `clsCubeLevel`
- `clsDatabaseLevel`
- `clsPartitionLevel`

Data Type

String

Access

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsDatabaseLevel</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>ClsCubeLevel</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsPartitionLevel</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsAggregationLevel</code></td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

Any properties that can be specified for a calculated member can also be specified for the member formulas. The column assigned to this property should contain a comma-delimited list of the properties, employing the following syntax:
<property identifier> = '<property value>' [, <property identifier> = '<value>']

The <property identifier> contains the name of a valid property, while the <property value> contains the string representation of the value of the specified property. For example, the FORE_COLOR cell property accepts a long integer containing the RGB value of a given color. To set the foreground color of the member to red, the syntax would resemble the following statement.

FORE_COLOR='255'

As with calculated members, these properties are optional. If the CustomRollupColumn property of the Level object is empty, the contents of its CustomRollupPropertiesColumn property are ignored.

See Also

Custom Rollup Formulas and Custom Member Formulas
Custom Rollup Operators
CustomRollupColumn
Level Interface
Description (Level Interface)

The Description property of the Level interface contains the level description.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Example

Use the following code to set the Description property for a level object:

' Assume an object (dsoLevel) of ClassType clsLevel exists
dsoLevel.Description = "Sales for 1998"

See Also
Level Interface
EnableAggregations (Level Interface)

The EnableAggregations property of the Level interface specifies whether aggregations can be created for the level by the aggregation design algorithm.

Applies To

clsCubeLevel
clsPartitionLevel

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCubeLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

To create aggregations for a level when aggregations are designed for the parent dimension, set the EnableAggregations property of the level to True and set the AggregationUsage property of the dimension to dimAggUsageCustom.

See Also

AggregationUsage
Level Interface
EstimatedSize (Level Interface)

The EstimatedSize property of the Level interface contains the estimated number of members in the level object. This property is used by the partition analyzer when aggregations are designed.

Applies To

clsAggregationLevel
clsCubeLevel
clsDatabaseLevel
clsPartitionLevel

Data Type

Long

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

The EstimatedSize property is used during analysis of a partition when aggregations are designed. The value provided does not need to be precise, but it should be a close approximation.

Note  The EstimatedSize property for an (All) level is read-only and always set
to one (1). This is also true for levels that belong to a virtual dimension created by earlier versions of Microsoft® SQL Server™ 2000 Analysis Services.

**Example**

Use the following code to set the size and type values for a new level object:

' Assume an object (dsoDim) of ClassType clsDimension exists
Set dsoLev = dsoDim.Levels.AddNew("Store Id")
dsoLev.MemberKeyColumn = """store""."store_number""
dsoLev.ColumnSize = 4
dsoLev.ColumnType = adInteger
dsoLev.EstimatedSize = 24

**See Also**

[clsPartitionAnalyzer](#)

[Level Interface](#)
FromClause (Level Interface)

The **FromClause** property of the **Level** interface contains the SQL FROM clause for the level.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

String

**Access**

Read-only

**Remarks**

This property returns a table name that is the same value as the **SourceTable** property of the parent dimension object.

**Example**

A database contains the following tables:

- Sales_Facts
- Customer
- Product
- **Product Class**

- **Promotion**

- **Store**

- **Calendar**

This diagram illustrates the relationships of these tables.

Use the following code to print the FROM clause for a level object:

```vbscript
' Assume an object (dsoLevel) of ClassType clsLevel
' and is associated with the Store dimension and
' Store_Name level
Debug.Print "    Level: " & dsoLevel.Name
Debug.Print " From Clause: " & dsoLevel.FromClause
```

The immediate window displays the following:

```
    Level: Store_Name
    From Clause: "store"
```

**See Also**
Level Interface
Grouping (Level Interface)

The **Grouping** property of the **Level** interface determines whether members in the level are used individually or are part of groups.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

**GroupingValues**

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

You can group level members to segment and organize the data contained in a cube. For example, a department store chain may want to use a cube to monitor customer purchasing trends. Querying for data on the customers dimension without grouping might return long lists of individual purchase events that are impractical to view or analyze. Grouping level members based on gender, age group, credit history, or payment method returns more manageable and
meaningful data. When such natural groupings are not available, this property can still be used to artificially impose organization on the data.

Member groups allow you to circumvent the maximum limit of 64,000 members per level. When members are grouped, queries return smaller segments of data. Each level can contain 64,000 groups, and each group can contain 64,000 members.

This feature provides support for very large levels. A level can be added (visible or invisible) that provides grouping for a large level. If you create a duplicate of a large level and set the Grouping property to groupingAutomatic, you can ignore the 64,000 member limit.

When this property is set to groupingAutomatic, the Analysis server automatically creates and names the member groups each time the dimension is processed. The number and names of the groups can change as the data in the cube changes.

If Grouping is set to groupingAutomatic, Decision Support Objects (DSO) changes the following properties for the level:

- The AreMemberNamesUnique property is read-only and set to False.

- The AreMemberKeysUnique property is read-only and set to False.

- The Ordering property is read-only and set to orderName.

You cannot set Grouping to groupingAutomatic for a level when any of the following conditions occur:

- The IsChanging property for the parent dimension is False.

- The level is the first or last within the dimension.

- The Grouping property of an adjacent level in the dimension is set to groupingAutomatic.
The **StorageMode** property for the parent dimension is **storeasROLAP**. Member groups are not supported for ROLAP dimensions.

**See Also**

- [AreMemberKeysUnique](#)
- [AreMemberNamesUnique](#)
- [IsChanging](#)
- [Level Interface](#)
- [Member Groups](#)
- [Ordering](#)
**HideMemberIf (Level Interface)**

The **HideMemberIf** property of the **Level** interface indicates whether and when a level member should be hidden from client applications.

**Applies To**
- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

**HideIfValues**

**Access**

Access depends on the value of the **ClassType** property of the object. This property does not apply to levels whose **SubClassType** is **sbclsParentChild** or **sbclsMining**.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

*Always read-only and set to hideNever for levels of virtual dimensions created in earlier versions of Microsoft® SQL Server™ 2000 Analysis Services.*

**Remarks**

Hidden members represent empty positions in a ragged hierarchy. The members are hidden to function as placeholders for a branch of a hierarchy that contains no real members at that level. For example, a geography dimension may have
cities at its lowest level. Members on this level roll up into a state/province level, which, in turn, rolls up into a countries level. Some countries, however, do not have states or provinces. In such cases a placeholder occupies the position.

See Also

Level Interface

Ragged Dimension Support
IsDisabled (Level Interface)

The IsDisabled property of the Level interface indicates whether the level is disabled.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsPartitionLevel

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object. This property does not apply to levels whose SubClassType is sbclsParentChild or sbclsMining.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCubeLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

If a level is disabled, it does not show up in a processed cube. That is, you cannot see the level when querying the cube using Multidimensional Expressions (MDX).

This property is useful in certain cases involving shared dimensions and multiple cubes. For example, if a parent dimension is shared between two different cubes and a child level has corresponding values in only one of the cubes, you can set
the `IsDisabled` property of the level object in the other cube to True to prevent queries against nonexistent level members.

**Note** When a level is disabled, all subordinate levels must already be disabled. When a level is enabled, all higher levels must already be enabled. At least one level must be enabled.

**Example**

Use the following code to disable a level object:

' Assume an object (dsoCubeLevel) of ClassType clsCubeLevel exists
dsoCubeLevel.IsDisabled = True

**See Also**

[Level Interface](#)
isValid (Level Interface)

The IsValid property of the Level interface indicates whether the level structure is valid. A level is valid if it is fully and correctly defined. For example, a level for which the FromClause has not been defined is not valid.

Applies To

clsAggregationLevel
clsCubeLevel
clsDatabaseLevel
clsPartitionLevel

Data Type

Boolean

Access

Read-only

Example

Use the following code to have Decision Support Objects (DSO) determine the validity of a level object:

' Assume an object (dsoLevel) of ClassType clsLevel exists
If Not dsoLevel.IsValid Then
   'Code to validate level definition
End If

See Also

Level Interface
IsVisible (Level Interface)

The IsVisible property of the Level interface determines whether the level is visible to client applications.

Applies To

clsAggregationLevel
clsCubeLevel
clsDatabaseLevel
clsPartitionLevel

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

Setting this property for a level of ClassType clsCubeLevel overrides the database setting of this property. Unlike other objects in the Decision Support Objects (DSO) library, a level cannot be referenced by calculated members or other Multidimensional Expressions (MDX) statements if its IsVisible property is False.
At least one level in a dimension must be visible. DSO raises an error if all the levels of a dimension have their IsVisible property set to False.

See Also

Level Interface
Analysis Services Programming

JoinClause (Level Interface)

The JoinClause property of the Level interface describes how related tables that define the parent Dimension object are linked and takes the form of a SQL JOIN clause.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String

Access

Read-only

Remarks

The JoinClause property of a level returns a nonempty string only when multiple dimension tables are used to define the parent Dimension object of the level. This type of definition results from the snowflake schema of fact and dimension tables in the cube.

Example

A database contains the following tables:

- Sales_Facts
- Customer
- Product
- Product_Class
- Promotion
- Store

- Calendar

This diagram illustrates the relationships of these tables.

Use the following code to return the JoinClause of the level object:

' Assume an object (dsoLevel) of ClassType clsLevel
' and is associated with the Product dimension and
' SKU level
Debug.Print " Level: " & dsoLevel.Name
Debug.Print " Join Clause: " & dsoLevel.JoinClause

The display in the immediate window would show the following:

Level: SKU
Join Clause: "product"."SKU"="product_class"."SKU"

See Also

Level Interface
LevelNamingTemplate (Level Interface)

The LevelNamingTemplate property of the Level interface defines how levels in a parent-child hierarchy are named.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String

Access

Access depends on the value of the ClassType property of the object. This property applies only to levels whose SubClassType is sbclsParentChild or sbclsMining.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

There are two ways to create a level-naming template. You can design a naming pattern or you can specify a list of names. A naming pattern contains an asterisk (*) as a placeholder character for a counter that is incremented and inserted into the name of each new and deeper level. For example, a LevelNamingTemplate
value of Level * results in the level names Level 1, Level 2, Level 3, and so on, if no (All) level is defined. If a naming pattern does not contain the placeholder, it is first used as is, and then subsequent level names are formed by appending a space and a number to the end of the pattern. For example, the

**LevelNamingTemplate** Level results in the level names Level, Level 1, Level 2, and so on.

To use a specific set of names for the **LevelNamingTemplate** property, create a list of level names and separate them with semicolons. Each member of the list is used for a subsequent level name. If the number of levels exceeds the number of names in the list, the last name in the list is used as a template for any additional level names. For example, a **LevelNamingTemplate** value of Division;Group;Unit results in the level names Division, Group, Unit, Unit 1, Unit 2, and so on. By contrast, a **LevelNamingTemplate** value of Division;Group;Unit * results in the level names Division, Group, Unit 3, Unit 4, and so on.

Each name in the list is treated as a template to ensure uniqueness of level names. A **LevelNamingTemplate** value of Manager;Team Lead;Manager;Team Lead;Worker * results in the level names Manager, Team Lead, Manager 1, Team Lead 1, Worker 5, Worker 6.

To use the asterisk (*) character in a level name using **LevelNamingTemplate**, use two asterisks (**).
LevelType (Level Interface)

The LevelType property of the Level interface returns an enumeration constant that identifies the specific type of level. It tells client applications that encounter this level what kind of content the level contains.

Applies To

clsAggregationLevel
clsCubeLevel
clsDatabaseLevel
clsPartitionLevel

Data Type

LevelTypes

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read-only for virtual cube levels and for levels with a SubClassType of sbclsParentChild or sbclsMining.

Remarks

The LevelType property can be set to one of the constants enumerated by the LevelTypes enumeration. If the level is part of a relational OLAP (ROLAP) dimension, the first level must be unique and must always be of type levAll. Time levels have their LevelType property set to one of the time level constants,
such as `levTimeYears` or `levTimeMonths`. Time levels must be created according to the time hierarchy: For example, a level of type `levTimeYears` must be above a level of type `levTimeMonths`.

**Examples**

**A. Setting Level Type**

Use the following code to set the level type of a level object to days:

```vbnet
LevelObject.LevelType = levTimeDays
```

**B. Setting and Determining Level Type**

Use the following code to return the level type of a level object and to determine which level type has been returned:

```vbnet
'Assume an object (dsoLevel) of ClassType clsDatabaseLevel exists
Dim objType As DSO.LevelTypes
objType = dsoLevel.LevelType
Select Case objType
  Case levRegular
    ' Commands for levRegular, a level not time-related
  Case levAll
    ' Commands for levAll, the topmost level
  Case levTimeQuarters
    ' Commands for levTimeQuarters, a calendar quarter level
..Case levCompany
  ' Commands for levCompany, a company information level
..Case levGeoCity
  ' Commands for levGeoCity, a city name level
  Case levProduct
    ' Commands for levProduct, an individual products level
  Case Else
    ' Commands for other LevelTypes
End Select
```
See Also

Level Interface
**MemberKeyColumn (Level Interface)**

The **MemberKeyColumn** property of the **Level** interface contains the name or expression of the column that contains member keys.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel
- clsAggregationLevel

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R/W</td>
</tr>
</tbody>
</table>

*Read-only for virtual cube levels.

**Remarks**

Some general considerations concerning the **MemberKeyColumn** property are as follows:

- In relational terms, a key is a set of one or more columns that uniquely identify an entity. **MemberKeyColumn** is a key in this sense if the **AreMemberKeysUnique** property of a level is True. If
AreMemberKeysUnique is False, MemberKeyColumn uniquely identifies a member within the context of a parent member only. For example, months are unique only within the context of a given year.

- The MemberKeyColumn property controls the way the dimensions within a cube are processed. To improve cube-processing efficiency, when you create levels, make sure the lowest level has unique members.

- Often, the fact table will contain member key values but not necessarily member names. The user, however, sees the member names rather than the keys.

- MemberKeyColumn can contain any valid SQL expression that involves one or more columns from a single table. For example, either of the following is a valid expression that uses the Product.Prod_Year column:

```
"""Product"""".""""Prod_Year"""
DatePart('q',"Product"."Prod_Year")
```

**Note** If this property contains an SQL expression, the expression must be compatible with the SQL dialect supported by the OLE DB provider. Using an incompatible expression will result in an error when the cube is processed.

**Example**

Use the following code to set a level object MemberKeyColumn to the Customer_Number column in table Customer:

```
LevelObject.MemberKeyColumn = """"Customer"""".""""Customer_Numb
```

**See Also**

AreMemberKeysUnique

Level Interface

Member Names and Member Keys
MemberNameColumn
**MemberNameColumn (Level Interface)**

The **MemberNameColumn** property of the **Level** interface contains the name of the column that contains member names.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>Read-only for virtual cube levels, levels with a <strong>LevelType</strong> property of levAll, and levels with a <strong>Grouping</strong> of groupingAutomatic.</td>
</tr>
</tbody>
</table>

**Remarks**

This property is useful when the member key column may not contain information recognizable to the user.

For example, a table may contain the columns **SKU** and **Product_Name**. When you generate queries by SKU number, you may want to display the
corresponding product name for the sake of clarity. In this case, the member key column is **SKU**, but the member name column is **Product_Name**.

If you do not assign a value to **MemberNameColumn**, the Analysis server uses the **MemberKeyColumn** values for the member names.

Be careful when using expressions in **MemberNameColumn** that are based on columns other than those specified by **MemberKeyColumn**. Ensure that there is always a one-to-one correspondence in the values produced by the expressions in **MemberNameColumn** and **MemberKeyColumn**.

The **MemberNameColumn** property can contain any valid SQL expression involving one or more columns from the table that contains the member key column. Such an expression can be used to produce a calculated or concatenated string expression. For example, the following is a valid expression for a **Product.Prod_Year** member key column:

'Quarter ' & Format(DatePart('q',"Product"."Prod_Year"))

The following is another example from an **Employees** table:

"Employees"."LastName" + ', ' + "Employees"."Firstname"

**Note** If this property contains an SQL expression, the expression must be compatible with the SQL dialect supported by the OLE DB provider. Using an incompatible expression will result in an error when the cube is processed.

**Example**

Use the following code to set the **MemberNameColumn** property of a level object to the **Product_Name** column in table **Sales**:

' Assume an object (dsoLev) of ClassType clsLevel exists
dsoLev.MemberNameColumn = """"Sales"".""Product_Name"""

**See Also**

[Level Interface](#)

[MemberKeyColumn](#)
Name (Level Interface)

The Name property of the Level interface contains the name of the level object.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String (maximum length of 50 characters)

Access

Read/write (read-only after the object has been named)

Example

Use the following code to return a level object name:

```vbscript
Dim sName As String
sName = LevelObject.Name
```

See Also

- Level Interface
Analysis Services Programming

**Ordering (Level Interface)**

The **Ordering** property of the **Level** interface specifies the method to use when ordering the members of a level.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

**OrderTypes**

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

Values of the **Ordering** property affect the access and values of the **OrderingMemberProperty**. For example, to enable read/write access for **OrderingMemberProperty**, specify **orderMemberProperty** for the **Ordering** **OrderType**. If **orderKey** or **orderName** are specified, **OrderingMemberProperty** has read-only access with respective values of "Key" and "Name".
See Also

Level Interface

Ordering

OrderingMemberProperty
**OrderingMemberProperty (Level Interface)**

The **OrderingMemberProperty** of the **Level** interface specifies the member property that is used to determine the ordering of level members.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read-only for levels with an Ordering of orderKey or orderName.*

**Remarks**

This property allows the level to be ordered by a member property instead of by name or by key. To order by member property, set the value of the **Ordering** property to **orderMemberProperty**, and then set the value of the **OrderMemberProperty** property to the name of the member property. If the **Ordering** property of the object is set to **orderName** or **orderKey**, this property is read-only, and the value of this property is set to "Name" or "Key",
respectively.

See Also

Level Interface
Ordering
OrderTypes
OrdinalPosition (Level Interface)

The **OrdinalPosition** property of the **Level** interface contains the ordinal position of the level in the **Levels** collection.

**Applies To**

*clsAggregationLevel*

*clsCubeLevel*

*clsDatabaseLevel*

*clsPartitionLevel*

**Data Type**

Integer

**Access**

Read-only

**Remarks**

The **OrdinalPosition** property for a level determines the relative position of the level in the dimension hierarchy: A value of 1 defines the most aggregated level, and the maximum value defines the most detailed level. Decision Support Objects (DSO) sets the value of this property when you add the level to a dimension.

The following table shows an example of relative ordinal positions.

<table>
<thead>
<tr>
<th>Level</th>
<th>Ordinal position</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>1</td>
</tr>
<tr>
<td>Yearly</td>
<td>2</td>
</tr>
<tr>
<td>Quarterly</td>
<td>3</td>
</tr>
</tbody>
</table>
**Example**

Use the following code to return the ordinal position of a level object:

```vbnet
' Assume an object (dsoLev) of ClassType clsLevel exists
If dsoLev.OrdinalPosition = 1 Then
  'Code to handle top level
Else
  'Code to handle remaining levels
End If
```

**See Also**

[Level Interface](#)
Parent (Level Interface)

The Parent property of the Level interface returns a reference to the parent Dimension object.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

Dimension

Access

Read-only

Remarks

For each level, the ClassType of the parent object depends on the ClassType of the level object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Parent object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>clsDatabaseDimension</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>clsCubeDimension</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>clsPartitionDimension</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>clsAggregationDimension</td>
</tr>
</tbody>
</table>

See Also

- Dimensions
Level Interface
ParentKeyColumn (Level Interface)

The ParentKeyColumn of the Level interface contains the name of the parent column in a parent-child level.

Applies To

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

Data Type

String

Access

Access depends on the value of the ClassType property of the object. This property applies only to levels whose SubClassType is sbclsParentChild or sbclsMining.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseLevel</td>
<td>R/W*</td>
</tr>
<tr>
<td>ClsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

The ParentKeyColumn property contains the name of the column that stores the key values for the parents of individual members. This setting tells the Analysis server how to find the relationship information necessary to build the hierarchy of members in a parent-child dimension.
Note  Because the ParentKeyColumn defines a parent-child relationship, the column named in the ParentKeyColumn property must contain data of the same type as the column named in the MemberKeyColumn property.

See Also

Level Interface
LevelTypes
MemberKeyColumn
Parent-Child Dimensions
RootMemberIf (Level Interface)

The **RootMemberIf** property of the **Level** interface determines how the root member or members of a parent-child hierarchy are identified.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

**RootIfValues**

**Access**

Access depends on the value of the **ClassType** property of the object. This property applies only to levels whose **SubClassType** is **sbclsParentChild** or **sbclsMining**.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

This property determines how root members are identified within the level. A root member is a top-level member within a parent-child dimension.
See Also

Level Interface
Parent-Child Dimensions
SkippedLevelsColumn (Level Interface)

The SkippedLevelsColumn of the Level interface is used to define empty positions in a parent-child dimension.

Applies To

- clsAggregationLevel
- clsDatabaseLevel
- clsCubeLevel
- clsPartitionLevel

Data Type

String

Access

Access depends on the value of the ClassType property of the object. This property applies only to levels whose SubClassType is sbclsParentChild or sbclsMining.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>ClsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

This property contains the name of a column that stores the number of skipped (empty) levels between each member and its parent. This allows a parent-child hierarchy to skip levels between members. The values contained in this column must be nonnegative integers; otherwise a processing error occurs. If the column
contains no value, the current member has a level depth one below its parent.

See Also

Level Interface
Parent-Child Dimensions
Ragged Dimension Support
SubClassType
SliceValue (Level Interface)

The SliceValue property of the Level interface contains the level slice value.

Applies To

clsPartitionLevel

Data Type

String

Access

Read/write

Remarks

This property applies only to levels whose SubClassType is sbclsRegular, sbclsParentChild, or sbclsMining. For all other levels, this property returns an empty string.

Examples

Use the following code to set a level object slice value to the member name May_Sales:

' Assume an object (dsoLevel) of ClassType clsPartitionLevel exists
dsoLevel.SliceValue = "May_Sales"

Use the following code to return the slice value for a level object:

Dim sSliceVal As String
sSliceVal = LevelObject.SliceValue

See Also
clsPartitionLevel

Dimensions

Level Interface
SubClassType (Level Interface)

The SubClassType property of the Level interface contains an enumeration constant that identifies the subclass type of the object.

Applies To

- clsAggregationLevel
- clsDatabaseLevel
- clsCubeLevel
- clsPartitionLevel

Data Type

SubClassTypes

Access

Read-only

Remarks

Objects that implement the Level interface can have a SubClassType property of sbclsRegular, sbclsLinked, sbclsVirtual, sbclsMining, or sbclsParentChild.

Example

Use the following code to check the SubClassType property of a level object.

' Assume an object (dsoLevel) of ClassType clsCubeLevel exists
If dsoLevel.SubClassType = sbclsParentChild Then
  'Code to handle a parent-child level
Else
'Code to handle other types of levels
End If

See Also

Level Interface
Analysis Services Programming

**UnaryOperatorColumn (Level Interface)**

The **UnaryOperatorColumn** property of the **Level** interface contains the name of a column that stores mathematical operators serving as member-specific rollup instructions for a specified level.

**Applies To**

- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>ClassType</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>R</td>
</tr>
<tr>
<td>clsAggregationLevel</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

The **UnaryOperatorColumn** property provides a simple way to control how member values are rolled up to the values of their parents. When the value of this property is assigned to the name of a column, the contents of that column are used as the unary operator for the member. This unary operator is applied to the member when evaluating the value of the member's parent.
This property provides similar but simplified functionality of the 
**CustomRollupColumn** property. In comparison to the **CustomRollupColumn** property, which uses Multidimensional Expressions (MDX) expressions to determine how the member itself is evaluated, the **UnaryOperatorColumn** contains simple math operators to determine how the value of a member affects the parent. This property may be overridden by the values in the column specified in the **CustomRollupColumn** property. However, the **UnaryOperatorColumn** property overrides the **CustomRollupExpression** property.

The following table lists available unary operators and describes how they behave.

<table>
<thead>
<tr>
<th>Unary operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>The value of the member is added to the aggregate value of the preceding sibling members.</td>
</tr>
<tr>
<td>-</td>
<td>The value of the member is subtracted from the aggregate value of the preceding sibling members.</td>
</tr>
<tr>
<td>*</td>
<td>The value of the member is multiplied by the aggregate value of the preceding sibling members.</td>
</tr>
<tr>
<td>/</td>
<td>The value of the member is divided by the aggregate value of the preceding sibling members.</td>
</tr>
<tr>
<td>~</td>
<td>The value of the member is ignored.</td>
</tr>
</tbody>
</table>

Blank values and any other values not found in the table are treated as the plus sign (+) unary operator. There is no operator precedence, so the order of members among their siblings is important.

**See Also**

**CustomRollUpColumn**

**Level Interface**
MDStore Interface

The **MDStore** interface is implemented by objects in Decision Support Objects (DSO) that contain multidimensional data. The following table describes these objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>An object that represents a database on the Analysis server. Databases contain cubes, dimensions, mining models, and roles.</td>
</tr>
<tr>
<td>Cube</td>
<td>An object that represents a cube on the Analysis server. Cubes contain dimensions, measures, and commands.</td>
</tr>
<tr>
<td>Partition</td>
<td>An object that represents the physical storage for the data in a cube. Partitions contain dimensions, measures, and aggregations.</td>
</tr>
<tr>
<td>Aggregation</td>
<td>An object that represents the tables of aggregated (that is, precalculated) data in a cube. Aggregations contain dimensions, measures, and member properties.</td>
</tr>
</tbody>
</table>

Although all of these objects implement their own internal interfaces, the **MDStore** interface is the primary interface to be used when using these objects. To differentiate between the objects implementing the **MDStore** interface, the **ClassType** property is used. The following table lists the objects implementing the **MDStore** interface and associated **ClassType** property values.

<table>
<thead>
<tr>
<th>Object</th>
<th>Class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>clsDatabase</td>
</tr>
<tr>
<td>Cube</td>
<td>clsCube</td>
</tr>
<tr>
<td>Partition</td>
<td>clsPartition</td>
</tr>
<tr>
<td>Aggregation</td>
<td>clsAggregation</td>
</tr>
</tbody>
</table>

The relationships among these objects are maintained through hierarchical linkages using the **MDStores** collections of each of these objects and the server.
The **MDStores** collection of a server object contains database objects. Database objects contain cube objects. Cubes contain partitions, and partitions contain aggregations. Together, the **MDStore** interface and the **MDStores** collections establish and maintain the hierarchy that defines the structure of OLAP data.

The **MDStore** interface provides collections, methods, and properties to manipulate these objects, their contained objects, and data. The four objects that implement the **MDStore** interface do not necessarily implement all of the **MDStores** collections, properties, and methods. For example, only objects of **ClassType clsDatabase** have **MiningModels** collections. Also, some **MDStore** properties and collections may be restricted to read-only access by some objects. For example, an object of **ClassType clsDatabase** allows read/write access to its **DataSources** collection, whereas access to the **DataSources** collection of an object of **ClassType clsAggregation** is read-only.

You create objects that implement the **MDStore** interface by declaring a variable as an **MDStore** data type and then creating an instance of the object and adding it to the **MDStores** collection of another object. The **AddNew** method of the **MDStores** collection creates the instance, sets the object's name to the name you provide, adds the object to the collection, and sets its parent property to reference the owner of the collection. At the same time, the new object's **ClassType** is automatically initialized to the appropriate value depending on the object's parent. For example, if you use the **AddNew** method to create an object in a cube's **MDStores** collection, the new object's **ClassType** will be set to **clsPartition**.

For more information about DSO, see [Introducing Decision Support Objects and Interfaces](#).

### Applies To

- **clsAggregation**
- **clsCube**
- **clsDatabase**
- **clsPartition**
Examples

The following examples walk through the hierarchy of usage for the MDStore object. First, a server object is created, which contains an MDStores collection of databases. Next, a database is created in the server's MDStores database collection. Then, a cube is created in this new database's own collection of MDStores objects. The same process continues by creating a new partition and a new aggregation using the same method. Each time, an interface (or placeholder) is defined to hold an MDStore object. Then the AddNew method of the parent object's MDStores collection is used to create the MDStore object.

A. Creating a Server Object

In this example, a new server object is created and a connection is established to a server named LocalHost:

Dim dsoServer As DSO.Server
' Create a server object and connect to an OLAP server.
Set dsoServer = New DSO.Server
dsoServer.Connect("LocalHost")

B. Adding a Database

The following example declares an MDStore interface (dsoDB) and calls the AddNew method of the server object's MDStores collection. This creates an object whose ClassType property is set to clsDatabase and is interacted with by means of the MDStore interface that was created for it.

IMPORTANT In DSO, MDStore is used in different contexts to indicate different meanings. For example, in the preceding paragraph, MDStore refers to an interface and a collection. The MDStore interface is created first. Because it is an interface, the Microsoft® Visual Basic® keyword new is not used when defining the variable. The MDStores collection is the server object's collection of databases (that is, MDStore objects whose ClassType property has been set to clsDatabase).

' Create and add a database to the server's MDStores collection.
Dim dsoDB As DSO.MDStore
Set dsoDB = dsoServer.MDStores.AddNew("MyDatabase")
C. Adding a Cube

The following example creates an MDStore interface to hold the MDStore object created by the AddNew method of the database's MDStores collection. The resulting object's ClassType property is automatically set to clsCube.

' Create and add a cube to the database's MDStores collection.
Dim dsoCube As DSO.MDStore
Set dsoCube = dsoDB.MDStores.AddNew("MyCube")
'... additional code to set other cube properties

D. Adding a Partition

The following example creates an MDStore interface to hold the MDStore object created by the AddNew method of the cube's MDStores collection. The resulting object's ClassType property is automatically set to clsPartition.

' Create and add a partition to the cube's MDStores collection.
Dim dsoPart As DSO.Partition
Set dsoPart = dsoCube.MDStores.AddNew("MyPartition")
'... additional code to set other partition properties

E. Adding an Aggregation

The following example creates an MDStore interface to hold the MDStore object created by the AddNew method of the partition's MDStores collection. The resulting object's ClassType property is automatically set to clsAggregation.

' Create and add an aggregation to the partition's MDStores collection.
Dim dsoAgg As DSO.MDStore
Set dsoAgg = dsoPart.MDStores.AddNew("MyAggregation")
'... additional code to set other aggregation properties

See Also
Aggregations

Collections, MDStore Interface

Cubes

Databases

Methods, MDStore Interface

Partitions

Properties, MDStore Interface
Collections, MDStore Interface

The **MDStore** interface supports the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>The collection of user-defined commands or sequence of commands</td>
</tr>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>DataSources</td>
<td>The collection of objects specifying a data provider</td>
</tr>
<tr>
<td>Dimensions</td>
<td>The collection that holds the dimension definitions for an object</td>
</tr>
<tr>
<td>MDStores</td>
<td>The collection that holds <strong>MDStore</strong> objects</td>
</tr>
<tr>
<td>Measures</td>
<td>The collection that holds the measures</td>
</tr>
<tr>
<td>MiningModels</td>
<td>The collection of data mining models contained within a database</td>
</tr>
<tr>
<td>Roles</td>
<td>The collection that holds the user role definitions for a database</td>
</tr>
</tbody>
</table>

The following table shows the class types of the objects that each collection can contain.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Class type of contained objects</th>
</tr>
</thead>
</table>
| Commands     | **clsCubeCommand**  
|              |  **clsDatabaseCommand**                                                 |
| CustomProperties | **Property**                                                             |
| DataSources  | **clsDataSource**                                                        |
| Dimensions   | **clsAggregationDimension**  
|              |  **clsCubeDimension**                                                   
|              |  **clsDatabaseDimension**                                                
|              |  **clsPartitionDimension**                                               |
| MDStores     | **clsAggregation**                                                      
|              |  **clsCube**                                                             
|              |  **clsDatabase**                                                         
|              |  **clsPartition**                                                        |
Measures

<table>
<thead>
<tr>
<th>clsAggregationMeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCubeMeasure</td>
</tr>
<tr>
<td>clsPartitionMeasure</td>
</tr>
</tbody>
</table>

MiningModels

| clsMiningModel |

Roles

| clsCubeRole |
| clsDatabaseRole |
| clsMiningModelRole |

Access Cross-Reference

The following table shows whether a collection is read/write (R/W), read-only (R), or not applicable (n/a) for each of the **MDStore** objects.

<table>
<thead>
<tr>
<th>Collection</th>
<th>clsDatabase</th>
<th>clsCube</th>
<th>clsPartition</th>
<th>clsAggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>R/W</td>
<td>R/W</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Custom Properties</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Datasources</td>
<td>R/W</td>
<td>R/W*</td>
<td>R/W</td>
<td>n/a</td>
</tr>
<tr>
<td>Dimensions</td>
<td>R/W</td>
<td>R/W</td>
<td>R/</td>
<td>R</td>
</tr>
<tr>
<td>MDStores</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
</tr>
<tr>
<td>Measures</td>
<td>n/a</td>
<td>R/W</td>
<td>R/</td>
<td>R</td>
</tr>
<tr>
<td>MiningModels</td>
<td>R/W</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Roles</td>
<td>R/W</td>
<td>R/W</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

*This property is not applicable (n/a) for virtual cubes (that is, those of SubClassType sbclsVirtual).

See Also

[MDStore Interface](#)
Methods, MDStore Interface

The **MDStore** interface supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BeginTrans</strong></td>
<td>Begins a transaction on a database.</td>
</tr>
<tr>
<td><strong>Clone</strong></td>
<td>Copies an existing object to a target object of the same class type.</td>
</tr>
<tr>
<td><strong>CommitTrans</strong></td>
<td>Commits a transaction.</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>Locks an object to prevent multiple users from concurrently changing the object. This method is administered through the Command interface.</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>Merges two partitions.</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Processes an <strong>MDStore</strong> object.</td>
</tr>
<tr>
<td><strong>Rollback</strong></td>
<td>Rolls back a transaction.</td>
</tr>
<tr>
<td><strong>UnlockObject</strong></td>
<td>Unlocks a previously locked object.</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td>Updates the definition of an object in the meta data repository.</td>
</tr>
</tbody>
</table>

Method/Class Cross-Reference

The following table shows the implementation of methods by object. X indicates applicable; n/a indicates not applicable.

<table>
<thead>
<tr>
<th>Method</th>
<th>Database</th>
<th>Cube</th>
<th>Partition</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BeginTrans</strong></td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Clone</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>CommitTrans</strong></td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>n/a</td>
<td>n/a</td>
<td>X</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Rollback</strong></td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>UnlockObject</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n/a</td>
</tr>
</tbody>
</table>
See Also

MDStore Interface
BeginTrans (MDStore Interface)

The BeginTrans method of the MDStore interface initiates a transaction on the Analysis server database.

Applies To

clsDatabase

Syntax

object.BeginTrans

object

The Database object to which changes are to be applied.

Remarks

Transactions group the processing of objects on the Analysis server by using the Process method for Database, Cube, Partition, or Dimension objects after executing the BeginTrans method. Processing actions within a transaction are not initiated on the server until you execute the CommitTrans method. You can use the Rollback method to void a transaction and leave the state of the objects on the server in the same condition they were in before the transaction was initiated. The processing of all objects on which you execute the Process method within the same transaction is completed as a single atomic operation. All of the specified processing is completed if the transaction completes successfully; none of it is completed if you roll back the transaction or if it terminates abnormally.

If you invoke a Process method on an object without first explicitly beginning a transaction using the BeginTrans method, Decision Support Objects (DSO) creates a single transaction for you so that the object you are processing is always processed inside a transaction.

Example
The following code example begins a transaction on the **FoodMart 2000** database, processes the Sales and Budget cubes, and commits the transaction:

```vba
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore

' Connect to the local Analysis server.
    dsoServer.Connect "LocalHost"

' Open the FoodMart 2000 database.
    Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Begin a transaction on the database.
    dsoDB.BeginTrans

' Create a reference to the Sales cube.
    Set dsoCube = dsoDB.MDStores("Sales")

' Process the cube, refreshing data.
    dsoCube.Process processRefreshData

' Create a reference to the Budget cube.
    Set dsoCube = dsoDB.MDStores("Budget")

' Process the cube completely.
    dsoCube.Process processFull

' Commit the transaction.
    dsoDB.CommitTrans

See Also

CommitTrans
MDStore Interface
Rollback
Clone (MDStore Interface)

The **Clone** method of the **MDStore** interface copies the property values and optionally the collections of major and minor objects of an existing object to a target object of the same class type.

**Applies To**
- clsAggregation
- clsCube
- clsDatabase
- clsPartition

**Syntax**

```csharp
object.Clone(ByVal TargetObject As MDStore, ByVal Options As CloneOptions = cloneMajorChildren)
```

**object**

The **MDStore** object whose property values and collections of major and minor objects are to be copied.

**TargetObject**

An existing **MDStore** object.

**Options**

One of the values of the **CloneOptions** enumeration. If no value is specified, the **cloneMajorChildren** option is used. For more information, see **CloneOptions**.

**Remarks**

The **Clone** method, depending on the clone option specified in **Options**, copies properties and minor objects to a new **MDStore** object with the same **ClassType**.
property value.

See Also

MDStore Interface
CommitTrans (MDStore Interface)

The CommitTrans method of the MDStore interface commits a transaction previously initiated by the BeginTrans method on a Database object.

Applies To

clsDatabase

Syntax

object.CommitTrans

object

The Database object associated with the transaction.

Remarks

The CommitTrans method commits the transaction started with the BeginTrans method. The Rollback method can be used if the objects involved in the transaction are rolled back to the state prior to the execution of the BeginTrans method. If the CommitTrans method is called before the BeginTrans method is called, an error occurs.

Example

The following code example begins a transaction on the FoodMart 2000 database, processes the Sales and Budget cubes, and commits the transaction.

    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore

    ' Connect to the local Analysis server.
    dsoServer.Connect "LocalHost"
'Open the FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Begin a transaction on the database.
dsoDB.BeginTrans

' Create a reference to the Sales cube.
Set dsoCube = dsoDB.MDStores("Sales")

' Process the cube, refreshing data.
dsoCube.Process processRefreshData

' Create a reference to the Budget cube.
Set dsoCube = dsoDB.MDStores("Budget")

' Process the cube completely.
dsoCube.Process processFull

' Commit the transaction.
dsoDB.CommitTrans

**See Also**

[BeginTrans](#)

[MDStore Interface](#)

[Rollback](#)
LockObject (MDStore Interface)

The LockObject method of the MDStore interface locks an object to prevent multiple users from concurrently changing the object.

Applies To

clsCube
clsDatabase
clsPartition

Syntax

object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String))

object

The object to lock.

LockType

One of the lock types defined in the OlapLockTypes enumeration. For more information, see OlapLockTypes.

LockDescription

A string containing a description of the lock, available to other applications attempting to obtain a lock.

Remarks

It is sometimes possible for an application to request an additional lock on an already locked object. For example, other applications can request and receive an olapLockRead lock on an object already locked using the olapLockProcess lock. For more information on how lock types interact, see OlapLockTypes.
See Also

MDStore Interface
UnlockObject
Merge (MDStore Interface)

The **Merge** method of the **MDStore** interface merges two partitions into a single partition. The partitions must have the same aggregations and storage modes.

**Applies To**

**clsPartition**

**Syntax**

```object.Merge(ByVal SourceName As String)```

*object*

The partition object into which to merge the source partition object.

*SourceName*

A string that contains the name of the source partition object.

**Remarks**

Before merging two partitions that specify data slices, you must first set the slice of the receiving partition to the slice that will apply after the merge has been completed. Otherwise, the partitions will not be successfully merged. The slice for the receiving partition must be the parent of the first level, where the slice values for the two partitions differ.

For example, if you are merging a partition that contains data based on the slice `[AllTime].[1998].[Quarter2]` into a partition that contains `[AllTime].[1998].[Quarter1]`, the target partition's slice must be set to the parent of the two slices that differ, in this case `[AllTime].[1998]`. The target partition's slice must be set to this value before merging the partitions. For more information, see **Managing Partitions** and **Merging Partitions**.

**Note** This adjustment is done automatically when you merge partitions using the Analysis Manager user interface.
Examples

Merging Data Slices

The following code prepares two partitions for a merge by merging the data slice values so they are equal:

Sub MergeDataSlices(SourcePart As DSO.MDStore, _)  
  TargetPart As DSO.MDStore)  
  ' This example code merges the data slices of two partitions.  
  ' This subroutine does not merge the partitions; instead,  
  ' it compares the source and target partitions, changing  
  ' the target partition to match the source partition to  
  ' prepare it for merging.

  Dim dsoDimSource As DSO.Dimension  
  Dim dsoLevelSource As DSO.Level  
  Dim dsoDimTarget As DSO.Dimension  
  Dim dsoLevelTarget As DSO.Level

  Dim nDim As Integer, nLev As Integer, nLev2 As Integer

  ' Search for the first level where the slice differs.  
  ' Then use the parent level just above it.  
  ' Loop through each dimension in the source partition.  
  For nDim = 1 To SourcePart.Dimensions.Count  
    Set dsoDimSource = SourcePart.Dimensions(nDim)  
    Set dsoDimTarget = TargetPart.Dimensions(nDim)

  ' For each source and target dimension, compare the two  
  ' and find the first level where the data slice differs.  
  For nLev = 1 To dsoDimSource.Levels.Count  
    Set dsoLevelSource = dsoDimSource.Levels(nLev)  
    Set dsoLevelTarget = dsoDimTarget.Levels(nLev)
If dsoLevelSource.SliceValue <> dsoLevelTarget.SliceValue Then

' Clear the slice values for all of the levels below
' in the target partition.
For nLev2 = nLev To dsoDimSource.Levels.Count
    Set dsoLevelTarget = dsoDimTarget.Levels(nLev2)
    dsoLevelTarget.SliceValue = ""
Next

' Stop looping through levels.
Exit For

End If
Next
Next

' Now that the target partition is ready for merge,
' update it.
TargetPart.Update
End Sub

**See Also**

[MDStore Interface](#)

[UnlockObject](#)
Process (MDStore Interface)

The Process method of the MDStore interface creates and populates an MDStore object on the Analysis server.

Applies To

clsCube
clsDatabase
clsPartition

Syntax

object.Process([ByVal Options As ProcessTypes])

object

The MDStore object to process.

Options

An optional parameter specifying one of the values enumerated by the ProcessTypes enumeration. For more information, see ProcessTypes.

Remarks

Databases, cubes, and partitions can be processed. Processing each of these objects means that all subordinate objects are processed. For example, invoking the Process method for a database processes all of the associated dimensions, cubes, and data mining models. For more information about processing and the differences between processing and updating, see Maintaining OLAP Data, Cube Processing, and Dimension Processing.

Processing an MDStore object causes the Analysis server to read source data, perform calculations, and store aggregated data. For example, processing an object of ClassType clsCube causes the server to read all source data corresponding to the definition of the cube and to create the resulting
multidimensional cube of data. If you use the **Process** method on a **Database** object, all cubes, dimensions, and mining models in the database are processed. Processing a cube automatically causes the processing of all subordinate partitions. In addition, any of the cube's dimensions whose **State** property is not set to **olapStateCurrent** will also be processed, including shared dimensions.

Processing a cube whose **SubClassType** is **sbclsVirtual** causes cubes used by the virtual cube to be processed only if their **State** property is not set to **olapStateCurrent**.

**See Also**

[MDStore Interface](#)

[UnlockObject](#)
Rollback (MDStore Interface)

The **Rollback** method of the **MDStore** interface rolls back a transaction on a database. All changes made to the object subsequent to the initiation of the transaction with the **BeginTrans** method are voided and the object remains in the state it was in at the time of the beginning of the transaction.

**Applies To**

**clsDatabase**

**Syntax**

```csharp
object.Rollback

object
```

The database object on which to roll back the transaction.

**Remarks**

If the **Rollback** method is called without first calling the **BeginTrans** method, an error occurs.

**Example**

The following code example begins a transaction on the **FoodMart 2000** database, processes the Sales and Budget cubes, and rolls back the transaction. Executing the **Rollback** method for the database restores the Sales and Budget cubes to the state prior to the execution of the **BeginTrans** method.

```csharp
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore

' Connect to the local Analysis server.
```
dsoServer.Connect "LocalHost"

' Open the FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Begin a transaction on the database.
dsoDB.BeginTrans

' Create a reference to the Sales cube.
Set dsoCube = dsoDB.MDStores("Sales")

' Process the cube, refreshing data.
dsoCube.Process processRefreshData

' Create a reference to the Budget cube.
Set dsoCube = dsoDB.MDStores("Budget")

' Process the cube completely.
dsoCube.Process processFull

' Rollback the transaction
dsoDB.Rollback

See Also

BeginTrans
CommitTrans
MDStore Interface
UnlockObject
UnlockObject (MDStore Interface)

The UnlockObject method of the MDStore interface releases a lock on an MDStore object previously established by the LockObject method.

Applies To

clsCube
clsDatabase
clsPartition

Syntax

object.UnlockObject

object

The MDStore object to unlock.

Remarks

If an application that created one or more locks terminates before freeing them with the UnlockObject method, the Analysis server automatically releases the locks when the connection with the application is closed.

See Also

LockObject
MDStore Interface
Analysis Services Programming

Update (MDStore Interface)

The **Update** method of the **MDStore** interface updates the definition of an **MDStore** object in the metadata repository. Changes made to the values of an object's properties are not saved to the repository until the object's **Update** method is executed.

**Applies To**

- `clsCube`
- `clsDatabase`
- `clsPartition`

**Syntax**

```
object.Update
```

*object*

The **MDStore** object to be updated.

**Remarks**

The **Update** method has no effect on an object whose **IsTemporary** property is set to True, which means these objects are not stored in the repository.

**See Also**

- `IsTemporary`
- `LockObject`
- **MDStore Interface**
### Properties, MDStore Interface

The **MDStore** interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AggregationPrefix</strong></td>
<td>Contains the prefix that associates the <strong>MDStore</strong> object with an aggregation in the store.</td>
</tr>
<tr>
<td><strong>AllowDrillThrough</strong></td>
<td>Indicates whether drillthrough is allowed on the cube.</td>
</tr>
<tr>
<td><strong>Analyzer</strong></td>
<td>The analyzer object for the store.</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant identifying the specific object type.</td>
</tr>
<tr>
<td><strong>DefaultMeasure</strong></td>
<td>The name of the default measure for the object.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of the store.</td>
</tr>
<tr>
<td><strong>DrillThroughColumns</strong></td>
<td>The columns that are included in a drillthrough query.</td>
</tr>
<tr>
<td><strong>DrillThroughFilter</strong></td>
<td>The statement restricting rows that are returned by a drillthrough query.</td>
</tr>
<tr>
<td><strong>DrillThroughFrom</strong></td>
<td>An SQL FROM clause with the names of the tables used in drillthrough queries.</td>
</tr>
<tr>
<td><strong>DrillThroughJoins</strong></td>
<td>An SQL JOIN clause with the names of the tables used in drillthrough queries.</td>
</tr>
<tr>
<td><strong>EstimatedRows</strong></td>
<td>The estimated number of rows in the store.</td>
</tr>
<tr>
<td><strong>EstimatedSize</strong></td>
<td>Estimated size of all rows, in bytes, in the store.</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>A comma-separated list of the tables from which the store data is obtained.</td>
</tr>
<tr>
<td><strong>IsDefault</strong></td>
<td>Indicates whether the store is the default store.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>Indicates whether the <strong>MDStore</strong> object is writable.</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>Indicates whether the object is temporary.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the store object is valid.</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether a cube is visible to clients.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>A list of join conditions separated by AND.</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time a store was last processed.</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>A user-defined date. This property is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the store.</td>
</tr>
<tr>
<td>OlapMode</td>
<td>Returns an enumeration constant that identifies the type of OLAP mode of the store.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent <strong>MDStore</strong> object.</td>
</tr>
<tr>
<td>ProcessingKeyErrorLimit</td>
<td>Sets the number of allowable errors that can occur before processing will be stopped.</td>
</tr>
<tr>
<td>ProcessingKeyErrorLogFileName</td>
<td>The UNC path to a file for logging dimension key errors encountered during processing.</td>
</tr>
<tr>
<td>ProcessOptimizationMode</td>
<td>Indicates whether the Analysis server creates indexes and aggregations during or after processing.</td>
</tr>
<tr>
<td>RemoteServer</td>
<td>The name of the remote server where the data for the <strong>MDStore</strong> object is stored.</td>
</tr>
<tr>
<td>Server</td>
<td>Returns a reference to the <strong>DSO.Server</strong> object.</td>
</tr>
<tr>
<td>SourceTable</td>
<td>The name of the source table for the</td>
</tr>
<tr>
<td>Property</td>
<td>Database</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>AggregationPrefix</td>
<td>R</td>
</tr>
<tr>
<td>AllowDrillThrough</td>
<td>n/a</td>
</tr>
<tr>
<td>Analyzer</td>
<td>n/a</td>
</tr>
<tr>
<td>ClassType</td>
<td>R</td>
</tr>
<tr>
<td>DefaultMeasure</td>
<td>n/a</td>
</tr>
<tr>
<td>Description</td>
<td>R/W</td>
</tr>
<tr>
<td>DrillThroughColumns</td>
<td>n/a</td>
</tr>
<tr>
<td>DrillThroughFilter</td>
<td>n/a</td>
</tr>
<tr>
<td>DrillThroughFrom</td>
<td>n/a</td>
</tr>
<tr>
<td>DrillThroughJoins</td>
<td>n/a</td>
</tr>
<tr>
<td>EstimatedRows</td>
<td>n/a</td>
</tr>
<tr>
<td>EstimatedSize</td>
<td>R</td>
</tr>
<tr>
<td>FromClause</td>
<td>n/a</td>
</tr>
<tr>
<td>IsDefault</td>
<td>n/a</td>
</tr>
<tr>
<td>Property</td>
<td>IsReadWrite</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>OlapMode</td>
<td>R/W</td>
</tr>
<tr>
<td>Parent</td>
<td>R</td>
</tr>
<tr>
<td>RemoteServer</td>
<td>n/a</td>
</tr>
<tr>
<td>Server</td>
<td>R</td>
</tr>
<tr>
<td>SourceTable</td>
<td>n/a</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>n/a</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>n/a</td>
</tr>
<tr>
<td>State</td>
<td>R</td>
</tr>
<tr>
<td>SubClassType</td>
<td>R</td>
</tr>
</tbody>
</table>

* This property is not applicable (n/a) for virtual cubes (that is, those of SubClassType sbclsVirtual).

** This property is read-only (R) for virtual cubes (that is, those of SubClassType sbclsVirtual).

** See Also

- MDStore Interface
AggregationPrefix (MDStore Interface)

The AggregationPrefix property of the MDStore interface contains the prefix associated with an aggregation in an MDStore object.

Applies To

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

Data Type

String (maximum length 50 characters, exclusive of any plus signs)

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>R/W</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>R/W</td>
</tr>
<tr>
<td>clsPartition</td>
<td>R/W</td>
</tr>
<tr>
<td>clsAggregation</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

The default value for this property depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabase</td>
<td>None</td>
</tr>
<tr>
<td>clsCube</td>
<td>None.</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>clsPartition</td>
<td>If not provided by user, a unique name is derived from the parent cube name and partition name.</td>
</tr>
<tr>
<td>clsAggregation</td>
<td>The name of the parent partition.</td>
</tr>
</tbody>
</table>

Aggregation prefixes are used to generate aggregation names, and they are used for table names in the relational database.

A fully expanded aggregation name has four parts:

\(<DatabasePrefix><CubePrefix><PartitionPrefix><AggregationID>\)

The first three parts of the name are provided by the user and make up the aggregation prefix; the fourth part of the name is a system-defined ID over which users have no control. The first two prefixes (\(DatabasePrefix\) and \(CubePrefix\)) are optional. \(CubePrefix\) is used only if \(PartitionPrefix\) begins with a plus sign (+), and \(DatabasePrefix\) is used only if \(CubePrefix\) begins with a plus sign. For example, if \(PartitionPrefix\) is +_Partition1, \(CubePrefix\) is +_1995, and \(DatabasePrefix\) is Sales, the aggregation prefix is Sales_1995_Partition1.

**See Also**

MDStore Interface
AllowDrillThrough (MDStore Interface)

The AllowDrillThrough property of the MDStore interface indicates whether drillthrough is enabled on the cube.

Applies To

- clsCube
- clsPartition

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsCube</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsPartition</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

If this property is set to True, a user with drillthrough permissions on the cube can issue a drillthrough query requesting source rows for a cell.

See Also

MDStore Interface
Analyzer (MDStore Interface)

The Analyzer property of the MDStore interface contains a reference to the analyzer object associated with an MDStore object. The system automatically associates one analyzer with an MDStore object.

Applies To

- clsCube (excluding virtual cubes)
- clsPartition

Data Type

Object

The ClassType value of the returned object depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Returned object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsCube</td>
<td>clsCubeAnalyzer</td>
</tr>
<tr>
<td>ClsPartition</td>
<td>clsPartitionAnalyzer</td>
</tr>
</tbody>
</table>

Access

Read-only

Remarks

The analyzer object is used to perform structure and data analysis for the cube or partition to which it is associated. For more information, see clsCubeAnalyzer and clsPartitionAnalyzer.

See Also

- MDStore Interface
ClassType (MDStore Interface)

The ClassType property of the MDStore interface contains an enumeration constant identifying the specific class type.

Applies To

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

Data Type

ClassTypes

Access

Read-only

Remarks

Most objects in Decision Support Objects (DSO) have a ClassType and a SubClassType property. The SubClassType property uses an enumerated value to provide additional information about the object. This property supports four values from the ClassTypes enumeration:

- clsAggregation
- clsCube
- clsDatabase
- clsPartition
See Also

MDStore Interface
SubClassTypes
DefaultMeasure (MDStore Interface)

The DefaultMeasure property of the MDStore interface contains the name of the default measure for the MDStore object.

Applies To

- clsCube
- clsPartition

Data Type

String

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>R/W</td>
</tr>
<tr>
<td>clsPartition</td>
<td>R</td>
</tr>
</tbody>
</table>

See Also

MDStore Interface
Description (MDStore Interface)

The Description property of the MDStore interface contains a user-supplied description of the MDStore object or its contents.

Applies To

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

Data Type

String

Access

Read/write

See Also

MDStore Interface
DrillThroughColumns (MDStore Interface)

The **DrillThroughColumns** property of the **MDStore** interface contains a list of columns that will be included in a drillthrough query.

**Applies To**

- clsCube
- clsPartition

**Data Type**

String

**Access**

Read/write

**Remarks**

The format of the string is identical to the format of the column list contained in an SQL SELECT clause. It includes the ability to define aliases. The format of the string in this property is provider-specific; it must be formatted according to the rules of the data source associated with the cube or partition.

**See Also**

- [MDStore Interface](#)
DrillThroughFilter (MDStore Interface)

The **DrillThroughFilter** property of the **MDStore** interface contains a filter restricting the rows that can be returned by a drillthrough query.

**Applies To**

- clsCube
- clsPartition

**Data Type**

String

**Access**

Read/write

**Remarks**

This property contains a filter restricting the rows that can be returned by a drillthrough query. The format of the string is identical to the format of the Boolean expression contained in an SQL WHERE clause. If a filter is specified, it is logically combined using AND with the tables specified by the **DrillThroughJoins** property. The format of the string in this property is provider-specific; it must be formatted according to the rules of the data source associated with the cube or partition.

**See Also**

**MDStore Interface**
DrillThroughFrom (MDStore Interface)

The **DrillThroughFrom** property of the **MDStore** interface contains an SQL FROM clause with the names of the tables used in drillthrough queries.

**Applies To**

- clsCube
- clsPartition

**Data Type**

String

**Access**

Read/write

**Remarks**

This property contains the names of the tables to be used in the drillthrough query. The format of the string is identical to the format of the expression contained in an SQL FROM clause. The format of the string in this property is provider-specific; it must be formatted according to the rules of the data source associated with the cube or partition.

**See Also**

MDStore Interface
DrillThroughJoins (MDStore Interface)

The DrillThroughJoins property of the MDStore interface contains a series of joins between the tables used in drillthrough queries.

Applies To

- clsCube
- clsPartition

Data Type

String

Access

Read/write

Remarks

This property contains the names of the tables to be used in the query. The format of the string is identical to the format of the Boolean expression contained in an SQL FROM clause. The format of the string in this property is provider-specific; it must be formatted according to the rules of the data source associated with the cube or partition.

See Also

- MDStore Interface
EnableRealTimeUpdates (MDStore Interface)

The `EnableRealTimeUpdates` property of the `MDStore` interface indicates whether or not the object supports real-time updates.

**Applies To**
- `clsAggregation`
- `clsPartition`

**Data Type**
Boolean

**Access**
Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsPartition</td>
<td>R/W</td>
</tr>
<tr>
<td>ClsAggregation</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**
To enable an object of `ClassType clsPartition` object to support real-time updates, which are used by real-time cubes, the object must use a relational OLAP (ROLAP) storage mode (the `StorageMode` property of the object of `ClassType clsPartition` must be `storeasROLAP`) and a Microsoft® SQL Server™ 2000 data source.

**See Also**
- MDStore Interface
- Real-Time Cubes
Analysis Services Programming

EstimatedRows (MDStore Interface)

The EstimatedRows property of the MDStore interface contains the estimated number of rows in the MDStore object. This property value is used in the algorithm that designs aggregations.

Applies To

- clsAggregation
- clsCube
- clsPartition

Data Type

Double

Access

Read/write

Remarks

The interpretation of this property value depends on the value of the ClassType and SubClassType properties of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Subclass type</th>
<th>Interpretation of property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>Any (except sbclsVirtual)</td>
<td>Number of rows in the fact table of the cube</td>
</tr>
<tr>
<td>clsCube</td>
<td>sbclsVirtual</td>
<td>Sum of number of rows in the underlying cubes</td>
</tr>
<tr>
<td>clsPartition</td>
<td>Any</td>
<td>Number of rows in the fact table of the parent cube</td>
</tr>
<tr>
<td>clsAggregation</td>
<td>Any</td>
<td>Number of rows in the aggregation table</td>
</tr>
</tbody>
</table>
See Also

EstimatedSize

MDStore Interface
EstimatedSize (MDStore Interface)

The **EstimatedSize** property of the **MDStore** interface contains the estimated size, in bytes, of the **MDStore** object.

**Applies To**

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

**Data Type**

Double

**Access**

Read-only

**Remarks**

**Note**  The **EstimatedSize** property for a relational OLAP (ROLAP) cube does not include the size of the tables in the relational database. For a hybrid OLAP (HOLAP) cube, the **EstimatedSize** property does not include the size of the fact table. For more information about partition storage modes, see [Partition Storage](#).

The **EstimatedSize** property is valid only after an object is processed. The interpretation of this property value depends on the value of the **ClassType** and **SubClassType** properties of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Subclass type</th>
<th>Interpretation of property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsCube</td>
<td>Any (except sbclsVirtual)</td>
<td>The size of the cube data and aggregations</td>
</tr>
<tr>
<td>Class</td>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>clsCube</td>
<td>sbclsVirtual</td>
<td>The size of the virtual cube</td>
</tr>
<tr>
<td>clsPartition</td>
<td>Any</td>
<td>The size of the partition</td>
</tr>
<tr>
<td>clsAggregation</td>
<td>Any</td>
<td>The size of the aggregation table</td>
</tr>
</tbody>
</table>

**See Also**

[EstimatedRows](#)

[MDStore Interface](#)
FromClause (MDStore Interface)

The **FromClause** property of the **MDStore** interface contains a comma-separated list of the fact table and the dimension tables from which store data is obtained.

### Applies To

- **clsAggregation**
- **clsCube** (excluding virtual cubes)
- **clsPartition**

### Data Type

String

### Access

Read/write

### Remarks

The **FromClause** property contains the string used by the data source provider to construct an SQL FROM clause.

**Note** You must separate the table and column names with the delimiters appropriate to the source database. You can use the **CloseQuoteChar** and **OpenQuoteChar** properties of the **DataSource** object to determine the correct delimiters.

### Example

```
' Assume the existence of a clsCube object, named dsoCube.
 dsoCube.FromClause = """"tblFacts"", """"tblProduct"", """"tblCustomer"
```
The previous code example sets the **FromClause** property to the following string:

"tblFacts", "tblProduct", "tblCustomer"

**See Also**

[EstimatedRows](#)

[MDStore Interface](#)
IsDefault (MDStore Interface)

The IsDefault property of the MDStore interface indicates that an MDStore object is the default partition of a cube or the default aggregation of a partition.

Applies To

clsAggregation
clsPartition

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregation</td>
<td>R/W</td>
</tr>
<tr>
<td>clsPartition</td>
<td>R</td>
</tr>
</tbody>
</table>

Remarks

The default value for this property depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregation</td>
<td>If you generate the aggregation by using an object of ClassType clsPartitionAnalyzer, the system automatically sets IsDefault to True. Generating aggregations in this way has performance benefits and is the recommended method for generating aggregations.</td>
</tr>
<tr>
<td></td>
<td>If you generate the aggregation without using a partition</td>
</tr>
</tbody>
</table>
analyzer object, you should set **IsDefault** to False.

| clsPartition | True if the partition is the only one in the cube, False otherwise. |

**Note** This property does not indicate that an aggregation object is the default within a collection of aggregations. If set to True, it indicates that the aggregation object contains the default dimensions and measures of the partition, which are already stored in the repository. This reduces the size of aggregation meta data, which can become important when a partition contains a large number of aggregations.

**See Also**

[MDStore Interface]
IsReadWrite (MDStore Interface)

The IsReadWrite property of the MDStore interface indicates whether the MDStore object is read-only or write-enabled.

Applies To

- clsCube
- clsDatabase
- clsPartition

Data Type

Boolean

Access

Access depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>R</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>R</td>
</tr>
<tr>
<td>clsPartition</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Remarks

The IsReadWrite property for objects of ClassType clsCube is set to True if the IsReadWrite property is set to True for at least one of the partitions associated with the cube. The IsReadWrite property for objects of ClassType clsDatabase is set to True if the IsReadWrite property is set to True for at least one of the cubes associated with the database.

See Also
MDStore Interface

Write-Enabled Cubes
IsTemporary (MDStore Interface)

The IsTemporary property of the MDStore interface indicates whether an object is temporary. Temporary objects are local to the session in which they are created, cannot be saved, and are not available to other users. To create a temporary object, preface the name with the tilde (~) character.

Applies To

clsAggregation
clsCube
clsPartition

Data Type

Boolean

Access

Read-only

Remarks

A temporary object is not stored in the repository and is not available to other users. Temporary objects persist only during the session in which they are created unless renamed or cloned to another existing object having the same class type. Objects subordinate to a temporary object, such as levels for a dimension, internally inherit the parent object's IsTemporary setting.

Note Only temporary objects can be renamed by changing the Name property. Removing the tilde (~) character from the name of a temporary object means that it is no longer temporary and prevents subsequent renaming of the object. Also, executing the Update method of a temporary object has no effect; the object is not saved to the repository unless the tilde prefix is removed by changing the Name property.
Examples

Creating a Temporary Object

Use the following code to create a temporary dimension object that is renamed and saved to the repository:

'Assume an object (dsoCube) of ClassType clsCube exists.
Dim tmpDim As DSO.Dimension
Set tmpDim = dsoCube.Dimensions.AddNew("~MyDim") 'Temporary
'Add levels, member properties, process, etc.
...
'This is something we want to keep - so drop "~".
tmpDim.Name = "MyDim" 'No longer temporary
tmpCube.Update

See Also

Dimension Interface

MDStore Interface
IsValid (MDStore Interface)

The **IsValid** property of the **MDStore** interface indicates whether the **MDStore** object and its dependent objects are valid.

**Applies To**
- clsAggregation
- clsCube
- clsDatabase
- clsPartition

**Data Type**
Boolean

**Access**
Read-only

**Remarks**
Validation depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregation</td>
<td>The name, parent, and prefix of the aggregation</td>
</tr>
<tr>
<td>clsCube</td>
<td>The measures, dimensions, data source, fact table, and other properties of the cube</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>The cubes, virtual cubes, dimensions, roles, and commands of the database</td>
</tr>
<tr>
<td>clsPartition</td>
<td>The measures, dimensions, fact table, aggregation prefix, and other properties of the partition</td>
</tr>
</tbody>
</table>
See Also

MDStore Interface
IsVisible (MDStore Interface)

The IsVisible property of the MDStore interface indicates whether a cube is visible to client applications.

Applies To

clsCube (excluding virtual cubes)

Data Type

Boolean

Access

Read/write

Remarks

You can create virtual cubes whose source cubes are not visible to client applications. This provides you with greater control over the data available to client users.

See Also

MDStore Interface

Virtual Cubes
JoinClause (MDStore Interface)

The JoinClause property of the MDStore interface contains the list of join conditions currently defined for an MDStore object.

Applies To

- clsAggregation
- clsCube (excluding virtual cubes)
- clsPartion

Data Type

String

Access

Read/write

Remarks

The JoinClause property stores the list of join conditions for the data source in the format used to define an SQL INNER JOIN clause for the data source provider.

Note  You must separate the table and column names with the delimiters that are appropriate to the source database. You can use the CloseQuoteChar and OpenQuoteChar properties of the DataSource object to determine the correct quoting characters.

Example

' Assume the existence of a clsCube object named dsoCube.
dsoCube.JoinClause = '"FactTable"."CustomerId"=" & _ 
"CustTable"."CustomerId" AND " & _
""FactTable"".""ProductId""=""ProductTable"".""SKU"

The previous code example sets the **JoinClause** property to the following string:
"FactTable"."CustomerId"="CustTable"."CustomerId" AND "FactTabl

**See Also**

[MDStore Interface](#)
LastProcessed (MDStore Interface)

The LastProcessed property of the MDStore interface contains the date and time when an MDStore object was last processed.

 Applies To
 clsAggregation
 clsCube
 clsDatabase
 clsPartition

Data Type
Date

Access
Read-only

Remarks
The LastProcessed property for an object is undefined and will raise an error if you attempt to read it when the value of the object’s State property is olapStateNeverProcessed. For more information, see State.

See Also
MDStore Interface
Process
Analysis Services Programming

**LastUpdated (MDStore Interface)**

The **LastUpdated** property of the **MDStore** interface is not used by Microsoft® SQL Server™ 2000 Analysis Services. You can set this to any date/time value you want, for example, to indicate when the source data was last changed.

**Applies To**

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

**Data Type**

Date

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabase</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCube</td>
<td>R/W</td>
</tr>
<tr>
<td>clsPartition</td>
<td>R/W</td>
</tr>
<tr>
<td>clsAggregation</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

Except for aggregation objects, which inherit the value of this property from their parents, the **LastUpdated** property is not automatically set by any method in the Decision Support Objects (DSO) object model. It is provided as a means for client applications to specify a date or time that represents the validity of information. For example, a date of 12/31/97 may mean that the information...
stored in a cube is not valid after December of 1997.

See Also

MDStore Interface
LazyOptimizationProgress (MDStore Interface)

The `LazyOptimizationProgress` property returns the progress of lazy optimization processing for an object of `ClassType clsPartition` object representing a multidimensional OLAP (MOLAP) partition.

**Applies To**

clsPartition

**Data Type**

Integer

**Access**

Read-only

**Remarks**

This property reports lazy processing progress for MOLAP partitions as an integer between 0 and 100, representing the completed percentage of lazy processing. For relational OLAP (ROLAP) and hybrid OLAP (HOLAP) partitions, the returned value is always 100. For unprocessed partitions or for partitions whose lazy processing has not yet started, this property returns 0.

**See Also**

MDStore Interface

State

Dimension Storage Modes
Name (MDStore Interface)

The Name property of the MDStore interface contains the name of the MDStore object.

Applies To

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

Data Type

String (maximum length of 50 characters)

Access

Read/write (read-only after the object has been named)

Remarks

The primary mechanism for identifying an MDStore object is the Name property. You specify the name of an object when you create the object. Unless the object is temporary, you cannot rename it after it has been created.

See Also

MDStore Interface
OlapMode (MDStore Interface)

The `OlapMode` property of the `MDStore` interface contains the OLAP storage mode assigned to the `MDStore` object.

**Applies To**
- `clsAggregation`
- `clsCube` (excluding virtual cubes)
- `clsDatabase`
- `clsPartition`

**Data Type**

`OlapStorageModes`

**Access**

Read/write

**Remarks**

The `OlapMode` property defines the storage mode for each fact table and aggregation in an `MDStore` object. Possible storage modes are relational OLAP (ROLAP) and multidimensional OLAP (MOLAP). Hybrid OLAP (HOLAP) storage combines ROLAP and MOLAP storage modes. Setting this property for a `clsDatabase` object defines the default storage mode for new cubes created within the database, whereas setting this property for a `clsCube` object defines the default storage mode for new partitions created within the cube.

This property is read-only and always `olapmodeROLAP` for a linked cube (that is, a cube of `SubClassType` of `sbclsLinked`).

**See Also**
MDStore Interface

Storage Modes for Partitions (MOLAP, ROLAP, HOLAP)
Analysis Services Programming

**Parent (MDStore Interface)**

The **Parent** property of the **MDStore** interface contains a reference to the parent of the **MDStore** object.

**Applies To**

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

**Data Type**

Object

The **ClassType** value of the returned object depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Returned object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClsDatabase</td>
<td>clsServer</td>
</tr>
<tr>
<td>ClsCube</td>
<td>clsDatabase</td>
</tr>
<tr>
<td>ClsPartition</td>
<td>clsCube</td>
</tr>
<tr>
<td>ClsAggregation</td>
<td>clsPartition</td>
</tr>
</tbody>
</table>

**Access**

Read-only

**Remarks**

The return type of the **Parent** property depends on the **ClassType** property of the object itself. For example, an object of **ClassType clsDatabase** has an **MDStores** collection that contains objects of **ClassType clsCube**. The object of
ClassType clsDatabase is the parent of the clsCube objects.

See Also

MDStore Interface
Analysis Services Programming

**ProcessingKeyErrorLimit (MDStore Interface)**

The **ProcessingKeyErrorLimit** property of the **MDStore** interface sets the number of allowable dimension key errors that cause processing on the Analysis server to cease.

**Applies To**

- clsCube
- clsPartition

**Data Type**

Long

**Access**

Read/write

**Remarks**

If the value of this property is 0 (the default), processing stops and an error description is written to the file specified in the **ProcessingKeyErrorLogFileName** property (if one is specified) the very first time a dimension key error is encountered during processing. By default, this property is set for an entire cube. However, it can be overridden by the value of this property for the individual partitions of the cube.

**See Also**

- Cube Processing
- MDStore Interface
Analysis Services Programming

**ProcessingKeyErrorLogFileName (MDStore Interface)**

The `ProcessingKeyErrorLogFileName` property of `MDStore` interface stores a directory path to a file for logging dimension key errors encountered during processing.

**Applies To**

*.clsCube*

*clsPartition*

**Data Type**

String

**Access**

Read/write

**Remarks**

Dimension key errors occurring during processing will result in a row being appended to the error log file if specified in this property. This file is in comma-separated values format (.csv) with the following fields:

- Date and time of the error

- The name of the database object containing the cube being processed

- The name of the cube being processed

- The name of the partition of the cube being processed
• The name of the dimension with the key error

• The name of the level with the key error (empty for parent-child dimensions)

• The key value from the cube's fact table that failed to match to the dimension

A key value that fails during processing may be written more than once. Thus, it is possible for this value to fail multiple times during an operation.

This error log file can be used to find rows in the fact table that do not correspond to rows in the dimension source table. For example, you can import the log file into a table within Microsoft® SQL Server™ 2000 and construct a query with an inner join between the fact table to the error log table to find the distinct rows that will not match.

By default, this property applies to the entire cube. However, it can be overridden by the value for this property for the individual partitions of the cube.

See Also

Cube Processing

MDStore Interface
ProcessOptimizationMode (MDStore Interface)

The ProcessOptimizationMode property of the MDStore interface indicates whether the Analysis server indexes and aggregates during or after processing.

Applies To

clsCube
clsPartition

Data Type

ProcessOptimizationModes

Access

Read/write

Remarks

This property can be used to expedite the availability of a cube or partition to users for analysis. By default, the processOptimizationModeRegular option specifies that the cube's source data is read, stored, indexed, and aggregated within the processing transaction. The processOptimizationModeLazyOptimizations option reads and stores the source data during the processing transaction and performs lazy processing of indexes and aggregations after processing is complete, when the Analysis server is idle.

By default, this property applies to the entire cube. However, it can be overridden by the value for this property for the individual partitions of the cube.

See Also

MDStore Interface
RemoteServer (MDStore Interface)

The RemoteServer property of the MDStore interface contains the name of the remote server where the data for the partition is stored, for remote partitions.

Applies To

clsPartition

Data Type

String

Access

This property is read-write only for partitions with a SubClassType of sbclsRemote. This property is read-only for all others.

Remarks

When the partition is first created, the value of this property is the empty string. After the property has been changed, it becomes read-only and cannot be changed again. If you want to change the remote server, you must delete and then re-create the partition. Remote partitions are used by distributed partitioned cubes to store partitioned data on Analysis servers other than the one on which the distributed partitioned cube is defined.

See Also

MDStore Interface

Distributed Partitioned Cubes
Server (MDStore Interface)

The Server property of the MDStore interface contains a reference to the DSO.Server object that is the ancestor of the object.

Applies To

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

Data Type

clsServer

Access

Read-only

Remarks

You can use this property to access the methods and properties of the server object.

See Also

MDStore Interface
SourceTable (MDStore Interface)

The **SourceTable** property of the **MDStore** interface contains the name of the fact table associated with the **MDStore** object.

**Applies To**
- **clsAggregation**
- **clsCube** (excluding virtual cubes)
- **clsPartition**

**Data Type**

String

**Access**

Read/write

**See Also**

- [MDStore Interface](#)
- [SourceTableFilter](#)
SourceTableAlias (MDStore Interface)

The **SourceTableAlias** property of the **MDStore** interface contains the alias of the source table for the cube or partition.

**Applies To**

- **clsCube**
- **clsPartition**

**Data Type**

String

**Access**

Read-write

**Remarks**

If the source table has no alias, the contents of this property are identical to those of the **SourceTable** property.

**Examples**

**Setting the SourceTable and SourceTable Alias Properties**

The following example shows how to set the **SourceTable** and **SourceTableAlias** properties:

' Assume that an MDStore object dsoCube exists
' If the FromClause property for the cube is:
' "customer, store, sales_fact_1997 AS Sales"
' Set the SourceTable property to the actual name of the table
dsoCube.SourceTable = "sales_fact_1997"
'And set the SourceTableAlias property to the name of the alias
dsoCube.SourceTableAlias = "Sales"

See Also

MDStore Interface
SourceTableFilter (MDStore Interface)

The `SourceTableFilter` property of the `MDStore` interface contains the WHERE clause of an SQL statement (without the WHERE keyword) used to determine which fact table records are to be included in the `MDStore` object.

**Applies To**

- `clsCube` (excluding virtual cubes)
- `clsPartition`

**Data Type**

String

**Access**

Read/write

**Remarks**

The SQL statement can contain multiple conditions, for example:

"time_by_day"."the_year" = '1997' AND "product"."product_id" = 'soap'

**Note** You must separate the table and column names with the delimiters that are appropriate to the source database. You can use the `CloseQuoteChar` and `OpenQuoteChar` properties of the `DataSource` object to determine the correct quoting characters.

**See Also**

- [MDStore Interface](#)
- [SourceTable](#)
State (MDStore Interface)

The State property of the MDStore interface returns an enumeration constant indicating the processing state of the object on the server represented by the Decision Support Objects (DSO) MDStore object.

Applies To

- clsCube
- clsDatabase
- clsPartition

Data Type

OlapStateTypes

Access

Read-only

Remarks

The State property indicates the current status of an MDStore object. It is used to determine whether processing of the object is required. For more information, see OlapStateTypes.

The supported values of the State property depend on the value of the class type of the associated MDStore object. The default value is olapStateNeverProcessed.

<table>
<thead>
<tr>
<th>Class type</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabase</td>
<td>olapStateNeverProcessed</td>
</tr>
<tr>
<td></td>
<td>olapStateCurrent</td>
</tr>
<tr>
<td>clsCube</td>
<td>olapStateNeverProcessed</td>
</tr>
<tr>
<td></td>
<td>olapStateSourceMappingChanged</td>
</tr>
<tr>
<td>olapStateCurrent</td>
<td>ClsPartition</td>
</tr>
</tbody>
</table>

See Also

MDStore Interface
SubClassType (MDStore Interface)

The SubClassType property of the MDStore interface contains the enumeration constant that identifies the subclass type of the object.

Applies To

- clsAggregation
- clsCube
- clsDatabase
- clsPartition

Data Type

SubClassTypes

Access

Read-only

Remarks

Objects of ClassType clsAggregation and clsDatabase can have a SubClassType property value of sbclsRegular. An object of ClassType clsCube can be of SubClassType sbclsRegular, sbclsVirtual, and sbclsLinked. An object of ClassType clsPartition can be of SubClassType sbclsRegular and sbclsRemote.

Examples

Checking the SubClassType Property of a Cube

Use the following code to check the SubClassType property of a cube:
'Assume an object (dsoCube) of ClassType clsCube exists.
If dsoCube.SubClassType = sbclsVirtual Then
    'Code to handle a virtual cube
Else
    'Code to handle a regular cube
End If

See Also

MDStore Interface
Measure Interface

Measures are the quantitative, numerical columns from the fact table of a cube. When a cube is processed, the data in the measures is aggregated across the dimensions in the cube. The aggregate functions are: Sum, Min, Max, Count, and Distinct Count. For more information, see Aggregate Functions.

In Decision Support Objects (DSO), the objects that implement the Measure interface have a ClassType property value of clsCubeMeasure, clsPartitionMeasure, or clsAggregationMeasure. These objects serve as containers for measure objects within each respective parent object. The Measure interface provides collections and properties that allow you to manipulate these objects. There are no methods associated with this interface. For more information about cube, partition, and aggregation objects, and how they relate to each other, see Introducing Decision Support Objects.

Not all of the objects that implement the Measure interface implement all of the properties of the interface. The properties of some objects may be restricted to read-only access, depending upon their type. For example, a clsCubeMeasure object allows read and write access to its FormatString property. Access to this property for any other measure object is read-only. The collections and properties of the Measure interface also apply to the measures of virtual cubes, although no special class is implemented for virtual cube measures. There are no methods associated with the Measure interface.

To illustrate the place of measures in a fact table, consider the case of a database that contains the following tables:

- Sales_Facts
- Customer
- Product
- Promotion
- **Product_Class**

- **Store**

- **Calendar**

The following diagram illustrates the relationships of these tables.

![Diagram showing relationships between tables]

If you build a cube based upon this database, the **Sales_Facts** table will be the fact table. The related tables will be the dimensions. The **Sales_Quantity**, **Unit_Price**, and **Unit_Cost** rows are measures that can be precalculated across dimensions such as Store, Customer, or Product.

**Applies To**

- `clsAggregationMeasure`
- `clsCubeMeasure`
- `clsPartitionMeasure`

**See Also**

- **Collections, Measure Interface**
- **Properties, Measure Interface**
Collections, Measure Interface

The Measure interface supports the following collection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

Applies To

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

Access

Read/write

See Also

Measure Interface
Properties, Measure Interface

The Measure interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregateFunction</td>
<td>Sets or returns a value that corresponds to the type of aggregate function used for a measure</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td>Description</td>
<td>Sets or returns the measure description</td>
</tr>
<tr>
<td>FormatString</td>
<td>Sets or returns the format used to display the measure values</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the measure object is valid</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the measure is visible to client applications</td>
</tr>
<tr>
<td>Name</td>
<td>Sets or returns the measure name</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the measure in the parent object's Measures collection</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent MDStore object</td>
</tr>
<tr>
<td>SourceColumn</td>
<td>Sets or returns the name of the column that is precalculated</td>
</tr>
<tr>
<td>SourceColumnType</td>
<td>Sets or returns the data type of the measure source column</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

Access Cross-Reference

The following table shows whether the property is read/write (R/W) or read-only (R) for different objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>clsCube Measure</th>
<th>clsPartition Measure</th>
<th>clsAggregation Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Access</td>
<td>Description</td>
<td>Format</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>AggregateFunction</td>
<td>R/W*</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>ClassType</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Description</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>FormatString</td>
<td>R/W*</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsValid</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>IsVisible</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Name</td>
<td>R/W</td>
<td>R (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Parent</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>SourceColumn</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>SourceColumnType</td>
<td>R/W*</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>SubClassType</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

* This property is read-only (R) for virtual cubes (that is, those of SubClassType sbclsVirtual).

**See Also**

*Measure Interface*
Analysis Services Programming

**AggregateFunction (Measure Interface)**

The `AggregateFunction` property of the `Measure` interface contains an enumeration constant that corresponds to the type of aggregate function used to generate the precalculated value of the measure.

**Applies To**

- `clsAggregationMeasure`
- `clsCubeMeasure`
- `clsPartitionMeasure`

**Data Type**

`AggregatesTypes`

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsAggregationMeasure</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsCubeMeasure</code></td>
<td>R/W*</td>
</tr>
<tr>
<td><code>clsPartitionMeasure</code></td>
<td>R</td>
</tr>
</tbody>
</table>

Read only: for measures in measures and measures in fully-created linked cubes.

**Remarks**

To create more sophisticated measures (for example, ratios or statistical functions), use the command object to create a calculated member. For more information, see `clsCubeCommand` and `clsDatabaseCommand`.

**Note**  Because the DISTINCT COUNT aggregation function does not support custom aggregations, the use of this aggregation function in combination with the `CustomRollupExpression` and `CustomRollupColumn` properties is not supported. If a cube uses the DISTINCT COUNT aggregation function and any
of the dimensions in that cube use either the **CustomRollupExpression** property or the **CustomRollupColumn** property, including data mining dimensions, the cube is invalid. Processing such a cube raises a validation error.

**Examples**

**Reading the AggregateFunction Property of a Measure Object**

Use the following code to read the value of the **AggregateFunction** property of a measure object:

```vbscript
' Assume an object (dsoAggMea) of ClassType clsAggregationMeasure
Dim AggType As DSO.AggregatesTypes
AggType = dsoAggMea.AggregateFunction
Select Case AggType
    Case aggSum
        ' Insert code for aggregation summation.
    Case aggCount
        ' Insert code for aggregation counts.
    Case aggMin
        ' Insert code for aggregation min.
    Case aggMax
        ' Insert code for aggregation max.
    Case aggDistinctCount
        ' Insert for aggregation distinct counts.
End Select
```

**See Also**

[AggregatesTypes](#)
[Measure Interface](#)
[CustomRollupColumn](#)
[CustomRollupExpression](#)
**ClassType (Measure Interface)**

The **ClassType** property of the **Measure** interface returns an enumeration constant that identifies the specific object type.

**Applies To**

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

**Data Type**

**ClassTypes**

For measure objects, **ClassType** is set to one of the following values:

- clsCubeMeasure
- clsPartitionMeasure
- clsAggregationMeasure

**Access**

Read-only

**Examples**

**Determining the ClassType Property of a Measure Object**

Use the following code to return the class type of a measure object and determine which object class has been returned:

'Assume an object (dsoCubeMea) of ClassType clsCubeMeasure exists
Select Case dsoCubeMea.ClassType
  Case clsCubeMeasure
    ' Insert code for a cube measure.
  Case clsPartitionMeasure
    ' Insert code for a partition measure.
  Case clsAggregationMeasure
    ' Insert code for an aggregation measure.
  Case Else
    ' other commands
End Select

See Also

ClassTypes
Measure Interface
Description (Measure Interface)

The `Description` property of the `Measure` interface contains the measure description.

**Applies To**

- `clsAggregationMeasure`
- `clsCubeMeasure`
- `clsPartitionMeasure`

**Data Type**

String

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsAggregationMeasure</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsCubeMeasure</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>clsPartitionMeasure</code></td>
<td>R</td>
</tr>
</tbody>
</table>

**Example**

Use the following code to set the measure object's description:

'Assume an object (dsoCubeMea) of ClassType `clsCubeMeasure` exists

dsoCubeMea.Description = "Extended price"

**See Also**

*Measure Interface*
FormatString (Measure Interface)

The **FormatString** property of the **Measure** interface contains the format used to display the measure values. Any format string valid for use with Microsoft® Visual Basic® is acceptable.

**Applies To**

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregationMeasure</td>
<td>R</td>
</tr>
<tr>
<td>clsCubeMeasure</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsPartitionMeasure</td>
<td>R</td>
</tr>
</tbody>
</table>

*Read-only for virtual cube measures.

**Example**

Use the following code to set the format string for the measure object [Sales_Facts].[Price]:

'Assume an object (dsoCubeMea) of ClassType clsCubeMeasure exists
dsoCubeMea.FormatString = ",###.##"

**See Also**

Measure Interface
IsValid (Measure Interface)

The IsValid property of the Measure interface indicates whether the measure structure is valid.

Applies To

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

Data Type

Boolean

Access

Read-only

Remarks

Depending on the value of the ClassType property of the measure object, the IsValid property validates the properties and methods of the measure as indicated in the following table.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregationMeasure</td>
<td>Name, Parent, and SourceField properties</td>
</tr>
<tr>
<td>clsCubeMeasure</td>
<td>Name, Parent, SourceField, and ColumnType properties</td>
</tr>
<tr>
<td>clsPartitionMeasure</td>
<td>Name, Parent, SourceField, and ColumnType properties</td>
</tr>
</tbody>
</table>

Example
Use the following code to determine whether the structure of a measure object is valid:

'Assume an object (dsoCubeMea) of ClassType clsCubeMeasure exists
Dim bValid As Boolean
bValid = dsoCube.IsValid

See Also

Measure Interface
IsVisible (Measure Interface)

The `IsVisible` property of the `Measure` interface determines whether the measure is visible to client applications.

**Applies To**

- `clsAggregationMeasure`
- `clsCubeMeasure`
- `clsPartitionMeasure`

**Data Type**

Boolean

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClsAggregationMeasure</code></td>
<td>R</td>
</tr>
<tr>
<td><code>ClsCubeMeasure</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>ClsPartitionMeasure</code></td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

You can use hidden measures to contain calculations used by other members, thereby keeping intermediate values away from client users.

At least one measure in a cube must be visible. The Analysis server raises an error if all the measures of a cube have their `IsVisible` property set to False.

**See Also**

- `Measure Interface`
**Name (Measure Interface)**

The Name property of the Measure interface contains the name of the measure object.

**Applies To**

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

**Data Type**

String

**Access**

Read/write (Objects can be renamed after their initial creation.)

**Example**

Use the following code to create a cube measure object and name it MyMeasure:

```
'Assume an object (dsoCube) of ClassType clsCube exists
Dim dsoMeasure As DSO.Measure
Set dsoMeasure = dsoCube.Measures.AddNew("MyMeasure")
```

**See Also**

- Description
- Measure Interface
OrdinalPosition (Measure Interface)

The `OrdinalPosition` property of the `Measure` interface contains the ordinal position of the measure in the `Measures` collection of the parent object.

**Applies To**
- `clsAggregationMeasure`
- `clsCubeMeasure`
- `clsPartitionMeasure`

**Data Type**
Integer

**Access**
Read-only

**Remarks**
If no default measure is specified, the first measure is the default measure for the cube. If a Multidimensional Expressions (MDX) expression or query does not contain an explicit reference to a measure, the Analysis server performs the command using the default measure.

**Example**
Use the following code to return the ordinal position of a measure object:

```vba
Dim OrdPos As Integer
OrdPos = MeasureObject.OrdinalPosition
```

**See Also**
Measure Interface
Parent (Measure Interface)

The Parent property of the Measure interface contains a reference to the parent MDStore object.

Applies To

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

Data Type

MDStore

The ClassType value of the returned object depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Returned object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregationMeasure</td>
<td>clsAggregation</td>
</tr>
<tr>
<td>clsCubeMeasure</td>
<td>clsCube</td>
</tr>
<tr>
<td>clsPartitionMeasure</td>
<td>clsPartition</td>
</tr>
</tbody>
</table>

Access

Read-only

See Also

Measure Interface
SourceColumn (Measure Interface)

The `SourceColumn` property of the `Measure` interface contains a reference to the column in the fact table that contains the measure.

**Applies To**

- `clsAggregationMeasure`
- `clsCubeMeasure`
- `clsPartitionMeasure`

**Data Type**

String

**Access**

Access depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsAggregationMeasure</code></td>
<td>R</td>
</tr>
<tr>
<td><code>clsCubeMeasure</code></td>
<td>R/W</td>
</tr>
<tr>
<td><code>clsPartitionMeasure</code></td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Remarks**

To set this property for a measure contained in a regular cube, use the delimiter characters for the data source when naming the table and the column.

For a virtual cube (a cube object with `IsVirtual = True`), the `SourceColumn` property is used to reference a measure within an existing cube rather than a column in a source fact table. When you set this property for a measure object within a virtual cube, you do not need to include the delimiter characters associated with the data source for the underlying cube.
The **SourceColumn** property works in conjunction with the **SourceColumnType** property.

**Examples**

**Setting the SourceColumn Property**

Use the following code to set the **SourceColumn** property for two measure objects:

' Assume two objects (dsoCubeMea, dsoVirtCubeMea)
' of ClassType clsCubeMeasure exist.
' The first object is a measure within a regular cube.
' The measure contains data from the Price column in
' the Sales_Facts table.
  dsoCubeMea.SourceColumn = """"Sales_Facts"""".""""Price""""

...  
' The second measure is for a virtual cube that references
' the Unit_Price measure of a regular cube named Sales
  dsoVirtCubeMea.SourceColumn = "[Sales].[Unit_Price]"

**See Also**

[Measure Interface](#)

[SourceColumnType](#)
SourceColumnType (Measure Interface)

The **SourceColumnType** property of the **Measure** interface identifies the type of data found in the measure object's **SourceColumn** property.

**Applies To**

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

**Data Type**

**ADODB.DataTypeEnum**

For more information about the **ADODB.DataTypeEnum** enumeration, see the Microsoft® ActiveX® Data Objects (ADO) documentation.

**SourceColumnType** is set to one of the following values.

<table>
<thead>
<tr>
<th>Column type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Integer</td>
<td>adBigInt</td>
</tr>
<tr>
<td>Binary</td>
<td>adBinary</td>
</tr>
<tr>
<td>Boolean</td>
<td>adBoolean</td>
</tr>
<tr>
<td>String (Unicode)</td>
<td>adBSTR</td>
</tr>
<tr>
<td>Char</td>
<td>adChar</td>
</tr>
<tr>
<td>Currency</td>
<td>adCurrency</td>
</tr>
<tr>
<td>Date</td>
<td>adDate</td>
</tr>
<tr>
<td>Date</td>
<td>adDBDate</td>
</tr>
<tr>
<td>Time</td>
<td>adDBTime</td>
</tr>
<tr>
<td>Date &amp; Time</td>
<td>adDBTimeStamp</td>
</tr>
<tr>
<td>Decimal</td>
<td>adDecimal</td>
</tr>
<tr>
<td>Double</td>
<td>adDouble</td>
</tr>
<tr>
<td>Integer</td>
<td>adInteger</td>
</tr>
<tr>
<td>Numeric</td>
<td>adNumeric</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Single</td>
<td>adSingle</td>
</tr>
<tr>
<td>Small Integer</td>
<td>adSmallInt</td>
</tr>
<tr>
<td>Tiny Integer</td>
<td>adTinyInt</td>
</tr>
<tr>
<td>Unsigned Big Integer</td>
<td>adUnsignedBigInt</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>adUnsignedInt</td>
</tr>
<tr>
<td>Unsigned Small Integer</td>
<td>adUnsignedSmallInt</td>
</tr>
<tr>
<td>Unsigned Tiny Integer</td>
<td>adUnsignedTinyInt</td>
</tr>
<tr>
<td>Char (Unicode)</td>
<td>adWChar</td>
</tr>
<tr>
<td>Text</td>
<td>adChar</td>
</tr>
</tbody>
</table>

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregationMeasure</td>
<td>R</td>
</tr>
<tr>
<td>clsCubeMeasure</td>
<td>R/W*</td>
</tr>
<tr>
<td>clsPartitionMeasure</td>
<td>R</td>
</tr>
</tbody>
</table>

* Read-only for virtual cube measures and measures in fully created linked cubes.

**Remarks**

The **SourceColumnType** property works in conjunction with the **SourceColumn** property. Be sure to specify a **SourceColumn** and **SourceColumnType** for each measure you create for a cube.

The **SourceColumnType** property for a measure within a virtual cube is inherited from the measure in the underlying regular cube and cannot be changed.

**Note** You must reference the ADO library in your project to use the **ADODB.DataTypeEnum** enumeration.

**Examples**
**Specifying the SourceColumnType Property**

Use the following code to specify and read a value for the **SourceColumnType** property:

'Assume an object (dsoCubeMea) of ClassType clsCubeMeasure exists
dsoCubeMea.ColumnType = adCurrency

```vba
Dim ColType As ADODB.DataTypeEnum
ColType = dsoCubeMea.ColumnType
Select Case ColType
    Case adDouble
        ' commands for adDouble
    Case adSingle
        ' commands for adSingle
    Case Else
        ' other commands
End Select
```

**See Also**

- Measure Interface
- SourceColumn
SubClassType (Measure Interface)

The SubClassType property of the Measure interface contains an enumeration constant identifying the subclass type of the object.

Applies To

- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

Data Type

SubClassTypes

Access

Read-only

Remarks

Objects that implement the Measure interface, (that is, those of ClassType of clsAggregationMeasure, clsCubeMeasure, or clsPartitionMeasure) can have a SubClassType property of sbclsRegular only. For more information, see SubClassTypes.

See Also

Measure Interface
Role Interface

The Role interface supports the maintenance of user groups and security parameters. Users can be grouped according to common access permissions by using the Role interface.

Using Decision Support Objects (DSO), you can use role objects to set permissions on the following areas in Analysis Services:

- Server
- Database
- Cube
- Dimensions and members
- Individual cube cells

Each role object also contains a collection for Command objects. The ability to create role-based commands is important for security reasons, and can also increase cube flexibility. You can customize the content of a cube to match the needs of individual users or entire groups.

In DSO, the objects that implement the Role interface have a ClassType property value of clsCubeRole, clsMiningModelRole, or clsDatabaseRole. The Role interface provides properties and methods to manipulate these objects.

Updating Security Information on the Analysis Server

Any changes you make to role objects are saved when any of the events listed in the following table occur.
<table>
<thead>
<tr>
<th>Saving a cube or mining model</th>
<th>Using the <strong>Update</strong> method of a cube or mining model object sends the updated security information to the Analysis server (assuming the cube or mining model has been processed at least once).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing a cube or mining model</td>
<td>Processing a cube or mining model updates the security information on the server. All role configuration data is saved, regardless of the processing option specified with the <strong>Process</strong> method.</td>
</tr>
<tr>
<td>Saving a database role</td>
<td>Using the <strong>Update</strong> method of an object of <strong>ClassType clsDatabaseRole</strong> sets the default values for the affiliated cube role objects in the database. Any changes you make to this default role are not applied to the affiliated cube role or mining model role objects that have values overriding the default.</td>
</tr>
</tbody>
</table>

When you make a change to default permission settings on a database role and invoke the **Update** method on the role, DSO finds all cubes that still use the default permission settings and sends the new security permissions for these cubes to the server.

For more information about database and cube objects and how they relate to each other, see [Introducing Decision Support Objects](#).

**Applies To**

- **clsCubeRole**
- **clsDatabaseRole**
- **clsMiningModelRole**

**See Also**

- [Collections, Role Interface](#)
Methods, Role Interface
Properties, Role Interface
Collections, Role Interface

The Role interface supports the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>The collection of commands for the role</td>
</tr>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

Access

Read/write

See Also

Role Interface
Methods, Role Interface

The Role interface supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies the properties of a role object to an existing role object</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks a role object</td>
</tr>
<tr>
<td>SetPermissions</td>
<td>Sets role permissions for a given key</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks a previously locked role object</td>
</tr>
<tr>
<td>Update</td>
<td>Saves a role in the repository</td>
</tr>
</tbody>
</table>

Method/Class Cross-Reference

The following table shows the applicability of each method to each object. X indicates applicable; n/a indicates not applicable.

<table>
<thead>
<tr>
<th>Method</th>
<th>clsDatabaseRole</th>
<th>clsCubeRole</th>
<th>clsMiningModelRole</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>LockObject</td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>SetPermissions</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Update</td>
<td>X</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

See Also

Role Interface
Clone (Role Interface)

The **Clone** method of the **Role** interface copies the property values and the collections of a role to a target object of the same class type.

**Applies To**

*clsDatabaseRole*

**Syntax**

```csharp
object.Clone(ByVal TargetObject As Role, [ByVal Options As CloneOptions = cloneMajorChildren])
```

*object*

The **clsDatabaseRole** object to be copied.

*TargetObject*

An existing **clsDatabaseRole** object.

*Options*

One of values of the **CloneOptions** enumeration. If no value is specified, the **cloneMajorChildren** option is used. For more information, see **CloneOptions**.

**Remarks**

Because **Role** objects do not contain major or minor objects, any clone option specified in **Options** is treated as **cloneObjectProperties**.

**See Also**

*CloneOptions*

*Role Interface*
Analysis Services Programming

**LockObject (Role Interface)**

The **LockObject** method of the **Role** interface locks a role object to prevent multiple users from concurrently changing the object.

**Applies To**

*clsDatabaseRole*

**Syntax**

```vbnet
object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String)
```

- **object**
  - The object to lock.

- **LockType**
  - One of the lock types defined in the **OlapLockTypes** enumeration. Because a **Role** object has no dependent objects and cannot be processed, the only valid options for **LockType** are **olapLockRead** and **olapLockWrite**. For more information, see **OlapLockTypes**.

- **LockDescription**
  - A string containing the description of the lock, available to other applications attempting to obtain a lock.

**See Also**

- **OlapLockTypes**
- **UnlockObject**
- **LockObject**
SetPermissions (Role Interface)

The **SetPermissions** method of the **Role** interface sets role permissions for a given key.

**Applies To**

- clsCubeRole
- clsDatabaseRole
- clsMiningModelRole

**Syntax**

```vbnet
bRet = object.SetPermissions(ByVal Key As String, ByVal PermissionExpression As String)
```

**bRet**

A Boolean variable that receives the completion status of the operation: True if it was completed successfully, False otherwise.

**object**

The role object on which to set permissions.

**Key**

String containing the permission key.

**PermissionExpression**

String containing the permission expression for the corresponding key.

**Remarks**

The **Permissions** property contains nine permissions keys. The meaning of each key and its possible **PermissionExpression** string values follow:

**Access**
The **Access** key indicates what type of access the users assigned to the **Role** object have to the entire cube. Valid **PermissionExpression** values for the key are listed in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>The members of this role have read-only access to the cube. (Default)</td>
</tr>
<tr>
<td>RW</td>
<td>The members of this role have read/write access to the cube.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsCubeRole**.

**AllowDrillThrough**

The **AllowDrillThrough** key indicates whether the users assigned to the **Role** object can execute drillthrough queries on the cube.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Drillthrough is allowed on this cube for members of this role. (Default)</td>
</tr>
<tr>
<td>False</td>
<td>Drillthrough is not allowed on this cube for members of this role.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsCubeRole**.

**AllowLinking**

The **AllowLinking** key indicates whether the users assigned to the **Role** object are allowed to link to the cube. Setting this property to False prevents users from creating linked cubes based on the cube.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Linking is allowed to this cube for members of this role. (Default)</td>
</tr>
<tr>
<td>False</td>
<td>Linking is not allowed to this cube for members of this role.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsCubeRole**.

**AllowSQLQueries**

The **AllowSQLQueries** key indicates whether the users assigned to the **Role**
object are allowed to execute SQL SELECT queries against the cube. Setting this property to False prevents users from creating local cubes based on the cube or viewing cube data using an SQL SELECT statement.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>SQL queries are allowed on this cube for members of this role. (Default)</td>
</tr>
<tr>
<td>False</td>
<td>SQL queries are not allowed on this cube for members of this role.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsCubeRole**.

**EnforcementLocation**

The **EnforcementLocation** key indicates whether security for the users assigned to the Role object is enforced on the server or on the client application.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Security is enforced on the client application for members of this role. (Default)</td>
</tr>
<tr>
<td>Server</td>
<td>Security is enforced on the server for members of this role.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsDatabaseRole** and **ClassType clsCubeRole**. If set on a database role object, the **PermissionExpression** value becomes the default value for all cube roles contained in the database.

**CellRead**

The **CellRead** key identifies visible, readable cells for the users assigned to the Role object. The **PermissionExpression** value contains a logical Multidimensional Expressions (MDX) expression, to be evaluated against a cell.

This key is for use only with objects of **ClassType clsCubeRole**.

**CellReadContingent**
The `CellReadContingent` key identifies contingent-readable cells for the users assigned to the `Role` object. The `PermissionExpression` value contains a logical MDX expression, to be evaluated against a cell.

This key is for use only with objects of `ClassType clsCubeRole`.

**CellWrite**

The `CellWrite` key identifies writable cells for the users assigned to the `Role` object. The `PermissionExpression` value contains a logical MDX expression, to be evaluated against a cell. A writable cell is considered readable by default.

This key is for use only with objects of `ClassType clsCubeRole`.

**Dimension:** `<dimension name>`

This key is used to specify dimension security options on a dimension, using a string value containing XML syntax. The syntax for the key includes the name of the dimension that will be secured by the role object. This key can be set for objects of `ClassType clsDatabaseRole` and `ClassType clsCubeRole`. If set on a database role object, the `PermissionExpression` value becomes the default value for all cube roles contained in the database.

The XML syntax for the `PermissionExpression` value is detailed here:

```
<memberSecurity
  [ IsVisible="<Boolean_string>"
  [ DefaultMember="<allowed_member>"
  [ VisualTotalsLowestLevel="<level_expression>"

<permission Access="Read"
  [ UpperLevel="<level_expression>"
  [ LowerLevel="<level_expression>"
  [ AllowedSet="<set_expression>"
  [ DeniedSet="<set_expression>"
  [ Description="<desc>"

<permission Access="Write"
```

The `<Boolean_string>` value can contain either "True" or "False". The `<allowed_member>` value contains the name of a single read-enabled member. The `<level_expression>` contains an MDX expression that returns a single level. The `<set_expression>` value contains an MDX expression that returns a set of members. The `<desc>` value contains a free-form text description of the permission.

**Example**

Use the following code to set permissions on an object of `ClassType clsCubeRole`:

'Assume an object (dsoCubeRole) of ClassType clsCubeRole exists
'Set a read-only permission
dsoCubeRole.SetPermissions "Access", "R"
'Set a read-write permission
dsoCubeRole.SetPermissions "Access", "RW"

**See Also**

- [Role Interface](#)
- [Dimension Security](#)
- [Cell Security](#)
UnlockObject (Role Interface)

The UnlockObject method of the Role interface releases a lock on a role object that has been previously established by the LockObject method.

Applies To

clsDatabaseRole

Syntax

object.UnlockObject

object

The role object from which to remove a lock.

Remarks

If an application that created one or more locks terminates before freeing them with the UnlockObject method, the Analysis server automatically releases the locks when the connection with the application is closed.

See Also

Role Interface
Update (Role Interface)

The **Update** method of the **Role** interface updates the definition of the role object in the meta data repository.

**Applies To**

[clsDatabaseRole](#)

**Syntax**

```csharp
object.Update
```

**object**

The role object to update.

**Remarks**

When you make a change to default permission settings on a database role and invoke the **Update** method on the role, Decision Support Objects (DSO) finds all cubes that did not overwrite the default permission setting and then sends the new security permissions to the server.

**See Also**

[Role Interface](#)
Properties, Role Interface

The **Role** interface supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of a role</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether a role structure is valid</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of a role</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>ParentObject</strong></td>
<td>Returns a reference to the parent object that the current role object is a child of</td>
</tr>
<tr>
<td><strong>Permissions</strong></td>
<td>The role permissions for a given key</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td><strong>UsersList</strong></td>
<td>A semicolon-delimited list of users</td>
</tr>
</tbody>
</table>

Access Cross-Reference

The following table shows whether the property is read/write (R/W), read-only (R), or not applicable (n/a) for different objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>clsDatabaseRole</th>
<th>clsCubeRole</th>
<th>clsMiningModelRole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
<td>R/W (R after the object has been named)</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Permissions</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>UsersList</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

**See Also**

[Role Interface]
**ClassType (Role Interface)**

The **ClassType** property of the **Role** interface contains an enumeration constant that identifies the specific class type.

**Applies To**

- clsCubeRole
- clsDatabaseRole
- clsMiningModelRole

**Data Type**

**ClassTypes**

**ClassType** is set to one of the following values:

- clsDatabaseRole
- clsCubeRole
- clsMiningModelRole

**Access**

Read-only

**Example**

Use the following code to return the class type of a role object and determine which object class has been returned:

```plaintext
' Assume the existence of object RoleObject
Dim ClassTyp As DSO.ClassTypes
ClassTyp = RoleObject.ClassType
```
Select Case ClassTyp
    Case clsDatabaseRole
        ' Insert code for a database role.
    Case clsCubeRole
        ' Insert code for a cube role.
    Case clsMiningModelRole
        ' Insert code for mining model roles.
    Case Else
        ' Insert code for other objects.
End Select

See Also

ClassTypes
Role Interface
SubClassType
Description (Role Interface)

The **Description** property of the **Role** interface contains the description of the role object.

**Applies To**

- clsCubeRole
- clsDatabaseRole
- clsMiningModelRole

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseRole</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeRole</td>
<td>R</td>
</tr>
<tr>
<td>clsMiningModelRole</td>
<td>R</td>
</tr>
</tbody>
</table>

**Example**

Use the following code to set a role object description:

```
RoleObject.Description = "Eastern Region Sales and Marketing"
```

**See Also**

- Role Interface
Analysis Services Programming

**IsValid (Role Interface)**

The **IsValid** property of the **Role** interface indicates whether the role object structure is valid.

**Applies To**

- **clsCubeRole**
- **clsDatabaseRole**
- **clsMiningModelRole**

**Data Type**

Boolean

**Access**

Read-only

**Remarks**

A role object is valid if the **Name** and **UsersList** properties are not empty strings and if its **Parent** property is valid.

**Example**

Use the following code to return a role object validity status:

Dim bRet As Boolean
bRet = RoleObject.IsValid

**See Also**

- **Role Interface**
Name (Role Interface)

The Name property of the Role interface contains the name of the role object.

Applies To

clsCubeRole
clsDatabaseRole
clsMiningModelRole

Data Type
String

Access
Read/write (read-only after the object has been named)

Example
Use the following code to set a role object name:

RoleObject.Name = "Sales and Marketing"

See Also
Role Interface
Parent (Role Interface)

The Parent property of the Role interface contains a reference to the parent MDStore object.

Applies To

clsCubeRole
clsDatabaseRole
clsMiningModelRole

Data Type

MDStore

The ClassType value of the returned object depends on the value of the ClassType property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Returned object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCubeRole</td>
<td>clsCube</td>
</tr>
<tr>
<td>clsDatabaseRole</td>
<td>clsDatabase</td>
</tr>
<tr>
<td>clsMiningModelRole</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

Access

Read-only

Remarks

This property will return Nothing for mining model roles, because data mining models do not support the MDStore interface. To obtain the parent object of a mining model role, use the ParentObject property.

Examples
Using the Parent and ClassType properties

The following example creates a database role and assigns it to the first cube in the database object's collection of cubes. It then prints some of the properties of the parent object by using the role object's Parent property.

Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore
Dim dsoDBRole As DSO.Role
Dim dsoCubeRole As DSO.Role

'Connect to the Analysis Server
    dsoServer.Connect "LocalHost"
    Set dsoDB = dsoServer.MDStores("FoodMart 2000")

'Get the first cube in the database's collection
    Set dsoCube = dsoDB.MDStores(1)
    Debug.Print "Cube.Name = " & dsoCube.Name

'Ensure the existence of TempRole in the database.
    Set dsoDBRole = dsoDB.Roles.AddNew("TempRole")

'Add the command to cube's collection of commands.
    Set dsoCubeRole = dsoCube.Roles.AddNew("TempRole")
    Debug.Print "Cube.Roles("TempRole").Name =" & _
        dsoCube.Roles("TempRole").Name

'Print the roles .ParentObject properties
    Debug.Print "   .Parent properties" & vbCrLf & _
        "   "
    Debug.Print "   TypeName("dsoCubeRole.Parent") = " & _
        TypeName(dsoCubeRole.Parent)
    If dsoCubeRole.Parent.ClassType = clsCube Then
Debug.Print " .ClassType = clsCube"
Else
    Debug.Print "This line should never be executed."
End If
Debug.Print " .Description = " & dsoCubeRole.Parent.Description
Debug.Print " .Name = " & dsoCubeRole.Parent.Name

dsoCube.Roles.Remove ("TempRole")
dsoDB.Roles.Remove ("TempRole")

See Also

Role Interface
ParentObject
ParentObject (Role Interface)

The `ParentObject` property returns a reference to the default interface of the parent object.

**Applies To**

- `clsCubeRole`
- `clsDatabaseRole`
- `clsMiningModelRole`

**Data Type**

Object

The default interface and `ClassType` value of the returned object depends on the value of the `ClassType` property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Returned object interface</th>
<th>Returned object class type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsDatabaseRole</code></td>
<td>MDStore</td>
<td><code>clsDatabase</code></td>
</tr>
<tr>
<td><code>clsCubeRole</code></td>
<td>MDStore</td>
<td><code>clsCube</code></td>
</tr>
<tr>
<td><code>clsMiningModelRole</code></td>
<td>MiningModel</td>
<td><code>clsMiningModel</code></td>
</tr>
</tbody>
</table>

**Access**

Read-only

**Examples**

**Using ParentObject and ClassType properties**

The following example creates a database role and assigns it to the roles collection of the first data mining model in the databases collection of mining
models. It then prints some of the properties of that role's parent object by using the **ParentObject** property.

```vba
Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoDMM As DSO.MiningModel
Dim dsoDBRole As DSO.Role
Dim dsoDMMRole As DSO.Role

'Connect to the Analysis Server
dsoServer.Connect "LocalHost"
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

'Get the first mining model
Set dsoDMM = dsoDB.MiningModels(1)
Debug.Print "MiningModel.Name = " & dsoDMM.Name

'Ensure the existence of TempRole in the database.
Set dsoDBRole = dsoDB.Roles.AddNew("TempRole")

'Ensure the existence of a role.
Set dsoDMMRole = dsoDMM.Roles.AddNew("TempRole")
Debug.Print "MiningModel.Roles("TempRole").Name =" & _
    dsoDMM.Roles("TempRole").Name

'Print the roles .ParentObject properties
Debug.Print " .ParentObject properties" & vbCrLf & _
    " ---------"
Debug.Print " TypeName(""dsoDMMRole.ParentObject"") = " & _
    TypeName(dsoDMMRole.ParentObject)
If dsoDMMRole.ParentObject.ClassType = clsMiningModel Then
    Debug.Print " .ClassType = clsMiningModel"
Else
    Debug.Print "This line should never be executed."
```
End If
Debug.Print " .Description = " & dsoDMMRole.ParentObject.Descr
Debug.Print " .Name = " & dsoDMMRole.ParentObject.Name

dsoDMM.Roles.Remove ("TempRole")
dsoDB.Roles.Remove ("TempRole")

See Also

Role Interface
Permissions (Role Interface)

The Permissions property of the Role interface contains the role permissions for a specified key.

Applies To

- clsCubeRole
- clsDatabaseRole
- clsMiningModelRole

Data Type

String

Access

Read-only

Remarks

The Permissions property contains nine permissions keys. The meaning of each key and its possible return values follow:

Access

The Access key indicates what type of access the users assigned to the Role object have to the entire cube. Valid return values for the key are listed in the following table.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>The members of this role have read-only access to the cube.</td>
</tr>
<tr>
<td>RW</td>
<td>The members of this role have read/write access to the cube.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of ClassType clsCubeRole.
**AllowDrillThrough**

The **AllowDrillThrough** key indicates whether the users assigned to the **Role** object can execute drillthrough queries on the cube. Valid return values for the key are listed in the following table.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Drillthrough is allowed on this cube for members of this role.</td>
</tr>
<tr>
<td>False</td>
<td>Drillthrough is not allowed on this cube for members of this role. (Default)</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsCubeRole**.

**AllowLinking**

The **AllowLinking** key indicates whether the users assigned to the **Role** object are allowed to link to the cube. Setting this property to False prevents users from creating linked cubes based on the cube.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Linking is allowed to this cube for members of this role. (Default)</td>
</tr>
<tr>
<td>False</td>
<td>Linking is not allowed to this cube for members of this role.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsCubeRole**.

**AllowSQLQueries**

The **AllowSQLQueries** key indicates whether the users assigned to the **Role** object are allowed to execute SQL SELECT queries against the cube. Setting this property to False prevents users from creating local cubes based on the cube or viewing cube data using an SQL SELECT statement.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>SQL queries are allowed on this cube for members of this role. (Default)</td>
</tr>
<tr>
<td>False</td>
<td>SQL queries are not allowed on this cube for members of this role.</td>
</tr>
</tbody>
</table>
This key is for use only with objects of **ClassType clsCubeRole**.

**EnforcementLocation**

The **EnforcementLocation** key indicates whether security for the users assigned to the **Role** object is enforced on the server or on the client application. Valid return values for the key are listed in the following table.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Security is enforced on the client application for members of this role. (Default)</td>
</tr>
<tr>
<td>Server</td>
<td>Security is enforced on the server for members of this role.</td>
</tr>
</tbody>
</table>

This key is for use only with objects of **ClassType clsDatabaseRole** and **ClassType clsCubeRole**.

**CellRead**

The **CellRead** key contains a logical Multidimensional Expressions (MDX) expression that identifies visible, readable cells for the users assigned to the **Role** object. If no MDX expression is specified for the **Role** object, an empty string is returned.

This key is for use only with objects of **ClassType clsCubeRole**.

**CellReadContingent**

The **CellReadContingent** key contains a logical MDX expression that identifies contingent-readable cells for the users assigned to the **Role** object. If no MDX expression is specified for the **Role** object, an empty string is returned.

This key is for use only with objects of **ClassType clsCubeRole**.

**CellWrite**

The **CellWrite** key contains a logical MDX expression that identifies writable cells for the users assigned to the **Role** object. If no MDX expression is specified for the **Role** object, an empty string is returned.
This key is for use only with objects of **ClassType clsCubeRole**.

**Dimension:** `<dimension name>`

The Dimension key is used to specify dimension security options on a dimension, using a string value containing XML syntax. The syntax for the key includes the name of the dimension that will be secured by the role object. The Dimension key contains XML syntax that defines the read and write access of the members for the dimension. If no dimension security options are specified for the Role object, an empty string is returned.

The XML syntax for the return value is detailed here:

```xml
<MEMBERSECURITY
    [ IsVisible="<Boolean_string>" ]
    [ DefaultMember="<allowed_member>" ]
    [ VisualTotalsLowestLevel="<level_expression>" ]
/>

<PERMISSION Access="Read"
    [ UpperLevel="<level_expression>" ]
    [ LowerLevel="<level_expression>" ]
    [ AllowedSet="<set_expression>" ]
    [ DeniedSet="<set_expression>" ]
    [ Description="<desc>" ]
/>

<PERMISSION Access="Write"
    [ UpperLevel="<level_expression>" ]
    [ AllowedSet="<set_expression>" ]
    [ Description="<desc>" ]
/>
</MEMBERSECURITY>
```

The `<Boolean_string>` value can contain either "True" or "False". The `<allowed_member>` value contains the name of a single read-enabled member. The `<level_expression>` contains an MDX expression that returns a single level. The `<set_expression>` value contains an MDX expression that returns a set of members. The `<desc>` value contains a free-form text
description of the permission.

This key is for use only with objects of **ClassType clsDatabaseRole** and **ClassType clsCubeRole**.

**Example**

Use the following code to return a role object's permission string:

```vba
Dim strPerms As String
strPerms = RoleObject.Permissions("Access")
```

**See Also**

- Role Interface
- SetPermissions
- Dimension Security
- Cell Security
SubClassType (Role Interface)

The SubClassType property of the Role interface contains an enumeration constant identifying the subclass type of the object.

 Applies To

clsCubeRole
clsDatabaseRole
clsMiningModelRole

Data Type

SubClassTypes

Access

Read-only

Remarks

Objects that implement the Role interface, that is, those of ClassType clsCubeRole, clsDatabaseRole, or clsMiningModelRole, can have a SubClassType property of sbclsRegular only.

See Also

ClassType
Role Interface
SubClassTypes
**UsersList (Role Interface)**

The **UsersList** property of the **Role** interface contains a semicolon-delimited list of users and/or groups assigned to the role.

**Applies To**

- clsCubeRole
- clsDatabaseRole
- clsMiningModelRole

**Data Type**

String

**Access**

Access depends on the value of the **ClassType** property of the object.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseRole</td>
<td>R/W</td>
</tr>
<tr>
<td>clsCubeRole</td>
<td>R</td>
</tr>
<tr>
<td>clsMiningModelRole</td>
<td>R</td>
</tr>
</tbody>
</table>

**Remarks**

The name for a user list must be qualified with the appropriate domain name. The general format is `<domain><user or group>`.

**Example**

Use the following code to set the list of users of a role object:

```csharp
RoleObject.UsersList = "Domain1\Ejones;Domain1\Analysts;Domain2\RGreen"
```
See Also

Role Interface
Analysis Services Programming
Events

The only object in Decision Support Objects (DSO) that directly supports event trapping is the **Database** object. This object fires events for all of its child objects including shared dimensions, cubes, partitions, aggregations, and data mining models.

The following table lists the events that this object supports.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ReportAfter</strong></td>
<td>Called whenever a processing action on an object in the database has finished executing</td>
</tr>
<tr>
<td><strong>ReportBefore</strong></td>
<td>Called before a processing action on an object in the database</td>
</tr>
<tr>
<td><strong>ReportError</strong></td>
<td>Called whenever an error occurs during a processing action</td>
</tr>
<tr>
<td><strong>ReportProgress</strong></td>
<td>Called to report the progress of an action during a processing action</td>
</tr>
</tbody>
</table>

Processing Actions

Each event reports the status of the processing action. This processing action is represented by integer constants. The tense for each processing action depends on the event being trapped. For instance, the merge action (**mdactMerge**) reports that two partitions or aggregations will be merged when trapped in the **ReportBefore** event. In contrast, this same action reports that two partitions or aggregations have been merged when trapped in the **ReportAfter** event.

The following is a list of actions that are supported by the database object events:

<table>
<thead>
<tr>
<th>Action</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td><strong>mdactProcess</strong></td>
<td>Indicates that the object referred to by <em>obj</em> has been processed.</td>
</tr>
<tr>
<td>Merge</td>
<td><strong>mdactMerge</strong></td>
<td>Reports that two</td>
</tr>
<tr>
<td>Method</td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Delete</td>
<td>mdactDelete</td>
<td>Indicates that an object has been deleted.</td>
</tr>
<tr>
<td>Delete Old Aggregations</td>
<td>mdactDeleteOldAggregations</td>
<td>Indicates that the existing relational OLAP (ROLAP) aggregations of a partition have been deleted.</td>
</tr>
<tr>
<td>Rebuild</td>
<td>mdactRebuild</td>
<td>Indicates that the definitions of an object have been rebuilt.</td>
</tr>
<tr>
<td>Commit</td>
<td>mdactCommit</td>
<td>Indicates that a transaction has been committed on the database.</td>
</tr>
<tr>
<td>Rollback</td>
<td>mdactRollback</td>
<td>Reports that a transaction has been rolled back on the database.</td>
</tr>
<tr>
<td>Create Indexes</td>
<td>mdactCreateIndexes</td>
<td>Indicates that indexes for a ROLAP aggregation have been created.</td>
</tr>
<tr>
<td>Create Table</td>
<td>mdactCreateTable</td>
<td>Reports that the aggregation table for the ROLAP aggregation has been created.</td>
</tr>
<tr>
<td>Insert Into</td>
<td>mdactInsertInto</td>
<td>Indicates that the aggregation table for the ROLAP partition has been populated.</td>
</tr>
<tr>
<td>Transaction</td>
<td>mdactTransaction</td>
<td>Reports that a transaction has been started, completed, or has encountered an exception.</td>
</tr>
<tr>
<td>Initialize</td>
<td>mdactInitialize</td>
<td>Indicates that the object referred to by the obj parameter has been initialized.</td>
</tr>
<tr>
<td>Create View</td>
<td>mdactCreateView</td>
<td>Reports that an aggregation view has been created for the partitions/aggregations have been merged.</td>
</tr>
</tbody>
</table>
ROLAP aggregation. This action is only valid when processing a ROLAP cube with Microsoft® SQL Server™ 2000 using indexed views.

<table>
<thead>
<tr>
<th>Action</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Data</td>
<td>mdactWriteData</td>
<td>Data has been written to the disk.</td>
</tr>
<tr>
<td>Read Data</td>
<td>mdactReadData</td>
<td>Data has been read from the disk.</td>
</tr>
<tr>
<td>Aggregate</td>
<td>mdactAggregate</td>
<td>Aggregations are being built.</td>
</tr>
<tr>
<td>Execute SQL</td>
<td>mdactExecuteSQL</td>
<td>An SQL statement has been executed.</td>
</tr>
<tr>
<td>Now Executing SQL</td>
<td>mdactNowExecutingSQL</td>
<td>An SQL statement is executing that can be canceled.</td>
</tr>
<tr>
<td>Executing Modified SQL</td>
<td>mdactExecuteModifiedSQL</td>
<td>A modified SQL statement has been executed.</td>
</tr>
<tr>
<td>Rows Affected</td>
<td>mdactRowsAffected</td>
<td>Reports number of rows affected by an SQL statement.</td>
</tr>
<tr>
<td>Error</td>
<td>mdactError</td>
<td>Indicates that an error has occurred during processing.</td>
</tr>
<tr>
<td>Write Aggregations and Indexes</td>
<td>mdactWriteAggsAndIndexes</td>
<td>Indexes and aggregations will be written to the disk.</td>
</tr>
<tr>
<td>Write Segment</td>
<td>mdactWriteSegment</td>
<td>Segments will be written to the disk.</td>
</tr>
<tr>
<td>Data Mining Model Processed Percentage</td>
<td>mdactDataMiningProgress</td>
<td>The status of the completion of processing for a data mining model in percentage terms.</td>
</tr>
</tbody>
</table>
For more information about the Database object, see clsDatabase.

**Tutorial - Trapping Database Events**

The following tutorial demonstrates trapping processing events. In examples A through C, a Microsoft Visual Basic® project file is set up that contains all of the information needed to use the rest of the examples. Examples D through G demonstrate trapping each of the events that are available from the database object.

A. Setting up the Project File  
B. Adding the Form_Load Event and Button Click Events  
C. Adding the ProcessDatabase Subroutine  
D. Adding the ReportBefore Event Handler  
E. Adding the ReportAfter Event Handler  
F. Adding the ReportProgress Event Handler  
G. Adding the ReportError Event Handler
ReportAfter (clsDatabase)

This event is called after a processing action for the Database object has finished executing.

Applies To

clsDatabase

Syntax

ReportAfter(obj As Object, ByVal Action As Integer, ByVal success As Boolean)

obj

Refers to the object being processed or the target object of the action.

Action

Refers to the processing action that has been completed.

success

Indicates whether the action succeeded.

Remarks

This event is called whenever a processing action for a given Database object (referenced by obj) or any of its subordinate major or minor objects has finished executing. The type of action can be determined from the value of Action. Whether or not the action was successful can be determined by the value of success.

For more information about using this event, see Events.
Analysis Services Programming

**ReportBefore (clsDatabase)**

This event is called before a processing action for a given **Database** object (referenced by `obj`) starts to run.

**Applies To**

`clsDatabase`

**Syntax**

```vba
ReportBefore(obj As Object, ByVal Action As Integer, Cancel As Boolean, Skip As Boolean)
```

- `obj`  
  Refers to the object being processed or the target object of the action.

- `Action`  
  Refers to the processing action that has been completed.

- `Cancel`  
  Allows the application to cancel an action by setting this parameter to True.

- `Skip`  
  Reserved for future use.

**Remarks**

This event is called before a processing action for a given **Database** object (referenced by `obj`) or any of its subordinate major or minor objects starts to run. The action can be determined from the value of `Action`.

For more information about using this event, see Events.
ReportError (clsDatabase)
This event is called whenever a processing error occurs.

Applies To
clsDatabase

Syntax
ReportError(obj As Object, ByVal Action As Integer, ByVal ErrorCode As Long, ByVal Message As String, Cancel As Boolean)

obj
Refers to the object being processed or the target object of the action.

Action
Refers to the processing action that has been completed.

ErrorCode
A value in the ErrorCodes enumeration.

Message
A user friendly message describing the error.

Cancel
Allows the application to cancel an action by setting this parameter to True.

Remarks
This event is called whenever an error occurs during processing.
For more information about using this event, see Events.
Analysis Services Programming

**ReportProgress (clsDatabase)**

This event is called to report progress during a processing action.

**Applies To**

*clsDatabase*

**Syntax**

```plaintext
ReportProgress(obj As Object, ByVal Action As Integer, Counter As Long, Message As String, Cancel as Boolean)
```

- **obj**
  - Refers to the object being processed or the target object of the action.

- **Action**
  - Refers to the processing action that has been completed.

- **Counter**
  - Indicates the numerical progress of the operation referred to by `Action`. For cubes, this argument refers to the number of rows that have been processed. For data mining models, this argument contains a number between 0 and 100 indicating the percentage of the processing task that has been completed.

- **Message**
  - A user-friendly message describing the progress made.

- **Cancel**
  - Allows the application to cancel an action by setting this parameter to True.

**Remarks**

This event can be used to update a progress bar or counter in a user interface. When using a progress bar to track the progress of a cube that is being
processed, the maximum value of the progress bar is determined by the cube's
**EstimatedRows** property. For tracking the progress of a data mining model, set
the maximum value of the progress bar to 100. When processing a cube this
event will be fired every 1000 rows. For data mining models this interval is
inconsistent and cannot be determined in advance.

For more information about using this event, see [Events](#).

**See Also**

[EstimatedRows](#)
Analysis Services Programming
Objects

In Decision Support Objects (DSO) there are two ways to classify objects: objects that can be accessed and managed directly, and objects that implement a DSO interface.

Objects that can be accessed and managed directly have their own collections, methods and properties. They include the following classes:

- **clsColumn**
- **clsCubeAnalyzer**
- **clsDataSource**
- **clsMemberProperty**
- **clsMiningModel**
- **clsPartitionAnalyzer**
- **clsServer**

**Note**  Class type designations that use the format `clsClassType`, such as `clsServer`, are used internally by the DSO `ClassType` property and do not necessarily correspond to a particular class definition within Microsoft® Visual Basic®.

Objects that implement an interface use a subset of the collections, methods, and properties associated with the interface. The DSO `ClassType` and `SubClassType` properties determine which features of an interface are implemented by a particular object. Information about these features appears throughout this document.

Each collection, method, and property description contains the names of the
objects in which it appears. Conversely, each object description contains the names of the collections, methods, and properties that the object implements.

The following table lists the six DSO interfaces and the objects that implement them.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implemented by</th>
</tr>
</thead>
</table>
| Command   | clsDatabaseCommand  
|           | clsCubeCommand     |
| Dimension | clsDatabaseDimension 
|           | clsCubeDimension    
|           | clsPartitionDimension 
|           | clsAggregationDimension |
| Level     | clsDatabaseLevel   
|           | clsCubeLevel       
|           | clsPartitionLevel  
|           | clsAggregationLevel|
| MDStore   | clsDatabase       
|           | clsCube           
|           | clsPartition      
|           | clsAggregation    |
| Measure   | clsCubeMeasure    
|           | clsPartitionMeasure 
|           | clsAggregationMeasure |
| Role      | clsDatabaseRole   
|           | clsCubeRole       
|           | clsMiningModelRole |

For more information, see Interfaces.

**Objects That Are Accessed Directly**

The following objects do not implement a shared interface and are accessed directly.

<table>
<thead>
<tr>
<th>Object</th>
<th>ClassType</th>
</tr>
</thead>
</table>

The DSO object model uses interfaces to simplify your interaction with groups of related objects, while maintaining parent-child inheritance throughout the object model hierarchy. For more information about the complete hierarchy, see **Introducing Decision Support Objects**.

Each DSO object that implements a DSO interface belongs to one of the following categories:

- Aggregations
- Commands
- Cubes
- Databases
- Dimensions
- Levels
- Measures
• Partitions

• Roles

The uniqueness of multiple DSO objects within the same category is determined by where each is contained within the overall DSO object model hierarchy. For example, a cube can contain several dimensions. These dimension objects are contained in the Dimensions collection of the cube. Each of these dimension objects is a DSO object of ClassType clsCubeDimension.

Each cube also contains a collection of partition objects. Each of these partition objects also contains a collection of dimension objects in its Dimensions collection. Each of these dimension objects is a DSO object of ClassType clsPartitionDimension.

Although the DSO objects of ClassType clsCubeDimension and clsPartitionDimension are both dimension objects, their methods and properties are unique because of the parent objects in which their collections are contained. The DSO object model groups such objects together and manages them by the implementation of a common interface.

**Major and Minor Objects**

In DSO, most child objects cannot commit their own changes to the Analysis server, but instead must rely on their parent object to commit the changes of their child objects. Any object that can commit itself and its children is referred to in DSO terminology as a major object. Any object that cannot commit itself, but must rely on a major object to perform such an action, is referred to as a minor object.

Objects with the following ClassType property values are considered major objects:

• clsCube

• clsDatabase

• clsDatabaseCommand
- clsDatabaseDimension

- clsDatabaseRole

- clsDataSource

- clsMiningModel

- clsPartition

- clsServer

All objects not included in the previous list are considered minor objects. To commit changes to major and minor objects, all major objects in DSO support the **Update** method. Any change to a DSO minor object must be committed through the parent DSO major object in order to be committed. For example, a change to a **clsCubeRole** object is committed only when the **Update** method of its parent **clsCube** object is executed. Although most interfaces in the DSO hierarchy have an **Update** method, attempting to use the **Update** method on a minor object in DSO will result in an error.
An object of ClassType `clsAggregation` provides a specific implementation of the Decision Support Objects (DSO) `MDStore` interface. Each instance of a `clsAggregation` object represents a unique DSO aggregation. This object provides collections, methods, and properties through the `MDStore` interface.

**Example**

Use the following code to list the aggregations contained within a partition:

```vbnet
'Assume an object (dsoServer) of ClassType clsServer exists
Dim dsoDB As MDStore
Dim dsoCube As MDStore
Dim dsoPart As MDStore
Dim dsoAgg As MDStore

Set dsoDB = dsoServer.MDStores(1)   'Database
Set dsoCube = dsoDB.MDStores(1)    'Cube
Set dsoPart = dsoCube.MDStores(1)  'Partition

'MDStores collection of a partition object
'contains objects of ClassType clsAggregation
Debug.Print " # Aggregations = " & dsoPart.MDStores.Count
```

**See Also**

- [Aggregations](#)
- [Collections, clsAggregation](#)
- [MDStore Interface](#)
- [Methods, clsAggregation](#)
Properties, clsAggregation
Collections, clsAggregation

An object of ClassType clsAggregation implements the following collections of the MDStore interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>Dimensions</td>
<td>The collection of dimension objects associated with the aggregation</td>
</tr>
<tr>
<td>Measures</td>
<td>The collection of objects associated with the aggregation</td>
</tr>
</tbody>
</table>

See Also

clsAggregation

MDStore Interface
Methods, clsAggregation

An object of `ClassType clsAggregation` implements the following methods of the `MDStore` interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an aggregation object to an existing target object of the same class type</td>
</tr>
</tbody>
</table>

See Also

- `clsAggregation`
- `MDStore Interface`
Properties, clsAggregation

An object of **ClassType clsAggregation** implements the following properties of the **MDStore** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AggregationPrefix</strong></td>
<td>Contains the prefix associated with an aggregation in an <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of the aggregation</td>
</tr>
<tr>
<td><strong>EnableRealTimeUpdates</strong></td>
<td>Indicates whether real-time updates are enabled for the aggregation</td>
</tr>
<tr>
<td><strong>EstimatedRows</strong></td>
<td>The estimated number of rows in the aggregation</td>
</tr>
<tr>
<td><strong>EstimatedSize</strong></td>
<td>The estimated size (in bytes) of all rows in the aggregation</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>The comma-separated list of source tables in the aggregation</td>
</tr>
<tr>
<td><strong>IsDefault</strong></td>
<td>Sets or returns True if the aggregation is the default aggregation for the partition, False if otherwise</td>
</tr>
<tr>
<td><strong>IsTemporary</strong></td>
<td>Indicates whether the aggregation should be persisted in the repository</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Returns True if the aggregation structure is valid, False if otherwise</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>The list of join conditions, separated by AND</td>
</tr>
<tr>
<td><strong>LastProcessed</strong></td>
<td>The date and time when the partition containing the aggregation was last processed</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the aggregation</td>
</tr>
<tr>
<td><strong>OlapMode</strong></td>
<td>Returns an enumeration constant that identifies the type of OLAP mode of the data store</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Returns a reference to the <strong>DSO.Server</strong> object</td>
</tr>
<tr>
<td><strong>SourceTable</strong></td>
<td>The name of the fact table for the aggregation</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

**See Also**

*clsAggregation*

*MDStore Interface*
An object of ClassType clsAggregationDimension allows you to specify the level of granularity an aggregation will have. This object provides collections and properties through a specific implementation of the Decision Support Objects (DSO) Dimension interface. There are no methods associated with an object of ClassType clsAggregationDimension.

Remarks

By default, an aggregation for a partition precalculates values based on the top-most levels within the partition. To specify a different granularity, that is, the degree to which an aggregation is precalculated, add additional levels to the aggregation's dimensions. For example, in a default scenario a cube (and consequently its partition) may contain a Time dimension that has the levels (All) (default), Year, Quarter, and Month. An aggregation for this partition inherits all of the dimensions of the partition, but only the top-most level or the default (All) level is precalculated. To precalculate a greater detail of data over the Time dimension, add one or more of the levels Year, Quarter, and Month.

Example

The following example causes the aggregation for the Time dimension to include data for the Year, Quarter, and Month levels, in addition to the default level (All):

'Assume an object (dsoAgg) of ClassType clsAggregation exists
Dim dsoAggDim as DSO.Dimension
Set dsoAggDim = dsoAgg.Dimensions("Time")
dsoAggDim.Levels.AddNew("Year")
dsoAggDim.Levels.AddNew("Quarter")
dsoAggDim.Levels.AddNew("Month")

See Also
Collections, clsAggregationDimension
Dimension Interface
Properties, clsAggregationDimension
Collections, clsAggregationDimension

An object of ClassType clsAggregationDimension implements the following collections of the Dimension interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>Levels</td>
<td>The collection of Level objects in an aggregation dimension</td>
</tr>
</tbody>
</table>

See Also

clsAggregationDimension
Dimension Interface
Properties, clsAggregationDimension

An object of ClassType `clsAggregationDimension` implements the following properties of the `Dimension` interface:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AllowSiblingsWithSameName</code></td>
<td>Specifies whether sibling members of the same parent within a dimension can have the same name.</td>
</tr>
<tr>
<td><code>AreMemberKeysUnique</code></td>
<td>Indicates whether member keys are unique within a particular level for the dimension.</td>
</tr>
<tr>
<td><code>AreMemberNamesUnique</code></td>
<td>Indicates whether member names are unique within a particular level for the dimension.</td>
</tr>
<tr>
<td><code>ClassType</code></td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
</tr>
<tr>
<td><code>DataMemberCaptionTemplate</code></td>
<td>Contains a template string that is used to create captions for system-generated data members.</td>
</tr>
<tr>
<td><code>DataSource</code></td>
<td>The name of the data source object.</td>
</tr>
<tr>
<td><code>DefaultMember</code></td>
<td>Defines the default member of the dimension.</td>
</tr>
<tr>
<td><code>DependsOnDimension</code></td>
<td>Names a dimension to which the current dimension is related.</td>
</tr>
<tr>
<td><code>DimensionType</code></td>
<td>Returns an enumeration constant that identifies the specific type of dimension.</td>
</tr>
<tr>
<td><code>EnableRealTimeUpdates</code></td>
<td>Indicates whether real-time updates are enabled for the dimension.</td>
</tr>
<tr>
<td><code>FromClause</code></td>
<td>A comma-separated list of the tables from which the store data is obtained.</td>
</tr>
<tr>
<td><code>IsChanging</code></td>
<td>Indicates whether members and/or levels are expected to change on a regular basis.</td>
</tr>
<tr>
<td><code>IsReadWrite</code></td>
<td>Indicates whether dimension writebacks are available to clients with appropriate</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsShared</td>
<td>Indicates whether the dimension is shared among cubes.</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>Indicates whether the dimension is temporary.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the dimension structure is valid.</td>
</tr>
<tr>
<td>IsVirtual</td>
<td>Indicates whether the dimension is virtual.</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the dimension is visible to the client.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>Contains the SQL JOIN clause for the dimension.</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time when the dimension was last processed.</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>User-specified date. It is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td>MembersWithData</td>
<td>Determines which members in a dimension can have associated data in the fact table.</td>
</tr>
<tr>
<td>Name</td>
<td>The dimension name.</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the dimension object within its parent object's Dimensions collection.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent MDStore object.</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>Restricts members included in a dimension.</td>
</tr>
<tr>
<td>StorageMode</td>
<td>Determines the method of storing dimension contents.</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type of the object.</td>
</tr>
</tbody>
</table>

**See Also**

[clsAggregationDimension](#)
Dimension Interface
**clsAggregationLevel**

An object of `ClassType clsAggregationLevel` provides a specific implementation of the Decision Support Objects (DSO) `Level` interface. It is used to maintain the level objects associated with an `MDStore` object that has a `ClassType` of `clsAggregation`. This object provides collections and properties through the `Level` interface. There are no methods associated with an object of `ClassType clsAggregationLevel`.

**Remarks**

Levels describe the dimension hierarchy from the highest (most aggregated) level to the lowest (most detailed) level of data. The (All) level of a dimension is the top level of a dimension; it includes all the members of subordinate levels.

**Example**

Use the following code to reference a level of an existing aggregation:

```
' Assume the existence of an object (myAgg) of
' ClassType clsAggregationDimension
Dim myLev As DSO.Level
Set myLev = myAgg.Levels("Brand Name")
```

**See Also**

- Collections, `clsAggregationLevel`
- Level Interface
- Properties, `clsAggregationLevel`
Collections, clsAggregationLevel

An object of **ClassType clsAggregationLevel** implements the following collection of the **Level** interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

**See Also**

- clsAggregationLevel
- Level Interface
Properties, clsAggregationLevel

An object of `ClassType clsAggregationLevel` implements the following properties of the `Level` interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AreMemberKeysUnique</code></td>
<td>Indicates whether the members of a level are uniquely identified by their member key column</td>
</tr>
<tr>
<td><code>AreMemberNamesUnique</code></td>
<td>Indicates whether the members of a level are uniquely identified by their member name column</td>
</tr>
<tr>
<td><code>ClassType</code></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><code>ColumnSize</code></td>
<td>The size (in bytes) of the data in the <code>MemberKeyColumn</code> property of the level</td>
</tr>
<tr>
<td><code>ColumnType</code></td>
<td>The data type of the <code>MemberKeyColumn</code> property of the level</td>
</tr>
<tr>
<td><code>CustomRollUpColumn</code></td>
<td>Contains the name of the column that contains member-specific rollup instructions</td>
</tr>
<tr>
<td><code>CustomRollUpExpression</code></td>
<td>Contains a Multidimensional Expressions (MDX) expression used to override the default rollup mode</td>
</tr>
<tr>
<td><code>CustomRollUpPropertiesColumn</code></td>
<td>Contains the name of the column that supplies cell properties for member-specific rollup instructions</td>
</tr>
<tr>
<td><code>Description</code></td>
<td>The description of the level</td>
</tr>
<tr>
<td><code>EstimatedSize</code></td>
<td>The estimated number of members in the level</td>
</tr>
<tr>
<td><code>FromClause</code></td>
<td>Contains the SQL FROM clause for the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Grouping</strong></td>
<td>Indicates the type of grouping used by the OLAP server</td>
</tr>
<tr>
<td><strong>HideMemberIf</strong></td>
<td>Indicates whether a member should be hidden from client applications</td>
</tr>
<tr>
<td><strong>IsDisabled</strong></td>
<td>Indicates whether the level is disabled</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the level structure is valid</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>Indicates whether the level is visible to client applications</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>Contains the SQL JOIN clause for the level</td>
</tr>
<tr>
<td><strong>LevelNamingTemplate</strong></td>
<td>Defines how levels in a parent-child hierarchy are named</td>
</tr>
<tr>
<td><strong>LevelType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of level</td>
</tr>
<tr>
<td><strong>MemberKeyColumn</strong></td>
<td>The name of the column that contains the member key of the aggregation level</td>
</tr>
<tr>
<td><strong>MemberNameColumn</strong></td>
<td>The name of the column that contains member names.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the level</td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>Specifies the method to use when ordering the members of a level</td>
</tr>
<tr>
<td><strong>OrderingMemberProperty</strong></td>
<td>Specifies a member property used to determine the ordering of members</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the level in the <strong>Levels</strong> collection of the parent object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent dimension object</td>
</tr>
<tr>
<td><strong>ParentKeyColumn</strong></td>
<td>Identifies the parent of a member in a parent-child hierarchy</td>
</tr>
<tr>
<td><strong>RootMemberIf</strong></td>
<td>Determines how the root member or members of a parent-child hierarchy are identified</td>
</tr>
<tr>
<td><strong>SkippedLevelsColumn</strong></td>
<td>Identifies the column that holds the number of empty levels between a member and its parent</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td><strong>UnaryOperatorColumn</strong></td>
<td>Contains the name of a column that stores mathematical operators serving as member-specific rollup instructions for the level</td>
</tr>
</tbody>
</table>

**See Also**

- clsAggregationLevel
- Level Interface
clsAggregationMeasure

An object of ClassType clsAggregationMeasure provides a specific implementation of the Decision Support Objects (DSO) Measure interface. It is used to maintain the measure objects contained within an aggregation object. This object provides collections and properties through the Measure interface. There are no methods associated with an object of ClassType clsAggregationMeasure.

Remarks

When a cube is processed, measures are aggregated across the dimensions in the cube.

Example

Use the following code to reference a measure of an aggregation:

'Assume an object (dsoAggregation) of ClassType clsAggregation exis
Dim dsoAggMeasure As DSO.Measure
Set dsoAggMeasure = dsoAggregation.Measures("Unit Sales")

See Also

Collections, clsAggregationMeasure
Measure Interface
Object Architecture
Properties, clsAggregationMeasure
Collections, clsAggregationMeasure

An object of **ClassType clsAggregationMeasure** implements the following collection of the **Measure** interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

See Also

- [clsAggregationMeasure](#)
- [Measure Interface](#)
Properties, clsAggregationMeasure

An object of `ClassType clsAggregationMeasure` implements the following properties of the `Measure` interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AggregateFunction</code></td>
<td>A value corresponding to the type of aggregation function used for the measure</td>
</tr>
<tr>
<td><code>ClassType</code></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><code>Description</code></td>
<td>Contains the description of the measure</td>
</tr>
<tr>
<td><code>FormatString</code></td>
<td>Contains the format used to display the measure values</td>
</tr>
<tr>
<td><code>IsValid</code></td>
<td>Indicates whether the measure structure is valid</td>
</tr>
<tr>
<td><code>IsVisible</code></td>
<td>Indicates whether the measure is visible to the client</td>
</tr>
<tr>
<td><code>Name</code></td>
<td>Contains the measure name</td>
</tr>
<tr>
<td><code>OrdinalPosition</code></td>
<td>Returns the ordinal position of the measure in the <code>Measures</code> collection of the parent object</td>
</tr>
<tr>
<td><code>Parent</code></td>
<td>Returns a reference to the parent aggregation object</td>
</tr>
<tr>
<td><code>SourceColumn</code></td>
<td>Contains the name of the measure column in the aggregated fact table</td>
</tr>
<tr>
<td><code>SourceColumnType</code></td>
<td>Returns a Microsoft® ActiveX® Data Objects (ADO) DB enumeration constant identifying the data type of the column specified by the <code>SourceColumn</code> property</td>
</tr>
<tr>
<td><code>SubClassType</code></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

See Also

`clsAggregationMeasure`
Measure Interface
clsCollection

A Decision Support Objects (DSO) collection is an object similar to a standard Microsoft® Visual Basic® Collection object. Unlike typical Visual Basic collections, however, DSO collections can contain only objects of the same type, determined by the ClassType property of the contained DSO objects. For example, the Dimensions collection can contain only objects of the object classes that apply to dimensions, such as clsDatabaseDimension, clsCubeDimension, clsPartitionDimension, and clsAggregationDimension. Collections are provided with methods and properties through their default interface, the OlapCollection interface, for interacting with them.

Remarks

The ContainedClassType property of the clsCollection object can be used to determine the objects allowed in a collection.

See Also

Collections
ClassType
ContainedClassType
Methods, clsCollection

The following methods apply to Decision Support Objects (DSO) collections.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds an existing object to a collection</td>
</tr>
<tr>
<td>AddNew</td>
<td>Creates and adds a new object to a collection</td>
</tr>
<tr>
<td>Find</td>
<td>Determines whether a specified object is in a collection</td>
</tr>
<tr>
<td>Item</td>
<td>Retrieves an object from a collection</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes an object from a collection</td>
</tr>
</tbody>
</table>

**Note**  These methods do not apply to CustomProperties collections. For more information, see CustomProperties.

**See Also**

Collections
Add (clsCollection)

The **Add** method of a Decision Support Objects (DSO) collection adds an object to the collection. This method does not apply to **CustomProperties** collections.

**Note** It is recommended that you use the **AddNew** method when adding new objects. You should use the **Add** method only when adding existing objects: for example, when you are adding aggregations to a partition after you have designed them. For more information, see [clsPartitionAnalyzer](#).

**Syntax**

```
object.Add(obj, [sKey As String], [Before])
```

**object**

An instance of a DSO collection object.

**obj**

An instance of a DSO object.

**sKey**

(Reserved) Defaults to `Item.Name`. If specified, it must contain the value of the item's **Name** property.

**Before**

(Optional) An expression that specifies a relative position in the collection. The member to be added is placed in the collection before the member identified by the **Before** argument. The value of **Before** must be a number from 1 to the value of the colCollection.**Count** property. If you omit this parameter, the item is appended at the last position in the collection. This argument is ignored if the collection is sorted; the member to be added is placed in the position indicated by the sort order of the collection.

**Note** All collections in the DSO object model are one-based. That is, the first item in the collection has an index of 1 and the last item has an index equal to the value of the **Count** property.
See Also

Collections
AddNew (clsCollection)

The **AddNew** method of a Decision Support Objects (DSO) collection creates and adds an object to a collection.

**Syntax**

```vba
Set vnt = object.AddNew(Name As String, [SubClassType As SubClassTypes])
```

* vnt

  A Variant variable that receives the instance of the new member. Instead of a variant, you can use a variable that has been declared to match the object being retrieved from the collection. For example, a variable declared as type **MDStore**, with its **ClassType** property value set to **clsCube**, can be used to retrieve an object from an **MDStores** collection of **clsCube** objects.

* object

  An instance of a DSO collection object.

* Name

  A string that specifies the name of the new object to add to the collection.

* SubClassType

  (Optional) One of the values enumerated by the **SubClassTypes** enumeration. For more information, see [SubClassTypes](https://docs.microsoft.com/en-us/dotnet/api/subclasstypes).

**Remarks**

The **ClassType** property of the new object is set automatically and depends on the parent of the collection to which the object is being added. For example, objects added to the **MDStores** collection of an object of **ClassType clsDatabase** automatically receive a **ClassType** value of **clsCube**.

The **AddNew** method maintains hierarchical relationships and ordering within the collection.
**Note** The **AddNew** method should be used when adding new objects to a collection. You should use the **Add** method only when adding existing objects: for example, when you are adding aggregations to a partition after you have designed them. For more information, see [clsPartitionAnalyzer](#).

**Example**

Use the following code to create a new cube and add it to the **MDStores** collection of cubes:

' Assume the existence of an object objDB
' of ClassType clsDatabase.
Dim objNewCube As MDStore
Set objNewCube = objDB.MDStores.AddNew("NewCube")

**See Also**

[Collections](#)

[SubClassTypes](#)
Analysis Services Programming

**Find (clsCollection)**

The **Find** method of a Decision Support Objects (DSO) collection locates an item in a collection. This method does not apply to **CustomProperties** collections.

**Syntax**

\[ bPresent = object.Find(vKey) \]

- **bPresent**
  A Boolean variable that receives the returned value: True if the item was found, False otherwise.

- **object**
  An instance of a DSO collection object.

- **vKey**
  The key or index of the item to be found.

**Example**

Use the following code to check for the existence of a partition named EastCoast in the **MDStores** collection of partitions for a cube:

```vql
' Assume the existence of an object cubCube of ClassType clsCube.
Dim bPresent As Boolean
bPresent = cubCube.MDStores.Find("EastCoast")
```

**See Also**

Collections
Item (clsCollection)

The Item method of a Decision Support Objects (DSO) collection returns an instance of an item in the collection. This method does not apply to CustomProperties collections.

Syntax

Set vnt = object.Item(vntIndexKey)

vnt

A Variant variable that receives the instance of the member. Instead of a variant, you can use a variable that has been declared to match the object being retrieved from the collection. For example, a variable declared as type MDStore, with its ClassType property value set to clsCube, can be used to retrieve an object from an MDStores collection of clsCube objects.

object

An instance of a DSO collection object.

vntIndexKey

Can be either the index (integer) or key (string) to the collection.

Note All collections in the DSO object model are one-based. That is, the first item in the collection has an index of 1 and the last item has an index of Count.

Example

Use the following code to return the partition named EastCoast from the MDStores collection of partitions for a cube:

' Assume the existence of an object cubCube
' of ClassType clsCube.
Dim Temp_Partition As MDStore
' Retrieve using the key

Analysis Services Programming
Set Temp_Partition = cubCube.MDStores.Item("EastCoast")
' OR Retrieve using the Index
Set Temp_Partition = cubCube.MDStores.Item(2)

See Also

Collections
Remove (clsCollection)

The **Remove** method of a Decision Support Objects (DSO) collection removes an item from the collection. This method does not apply to **CustomProperties** collections.

**Syntax**

```plaintext
object.Remove(vntIndexKey)
```

- **object**
  - An instance of a DSO collection object.

- **vntIndexKey**
  - Either the index (integer) or key (string) to the collection.

**Note** All collections in the DSO object model are one-based. That is, the first item in the collection has an index of 1 and the last item has an index of Count.

**Remarks**

The **Remove** method, by removing the selected member from the collection, removes the selected member from both the Analysis server and the repository.

**Example**

Use the following code to remove the partition named EastCoast from the **MDStores** collection of partitions for a cube:

```plaintext
' Assume the existence of an object dsoCube
' of ClassType clsCube.
' Remove using the key
dsoCube.MDStores.Remove "EastCoast"
' OR Remove using the Index
dsoCube.MDStores.Remove 2
```
See Also

Collections
Properties, clsCollection

The following properties apply to Decision Support Objects (DSO) collections.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassType</td>
<td>The class type of a collection</td>
</tr>
<tr>
<td>ContainedClassType</td>
<td>The class type of the items contained in a collection</td>
</tr>
<tr>
<td>Count</td>
<td>The number of items in a collection</td>
</tr>
</tbody>
</table>

Note  These properties do not apply to CustomProperties collections. For more information, see CustomProperties.

See Also

Collections
ClassType (clsCollection)

The **ClassType** property of a Decision Support Objects (DSO) collection returns the class type of the collection object. This property does not apply to **CustomProperties** collections.

Data Type

**ClassTypes**

This property returns **clsCollection** for all collections, regardless of the value of the **ClassType** property for the objects contained by the collection.

Access

Read-only

Remarks

Use the **ContainedClassType** property to determine the value of the ClassType objects accepted by the collection.

See Also

[ContainedClassType (clsCollection)]

[ClassTypes]

[clsDataSource]

[clsDataStore]

[Collections]

[Command Interface]

[Dimension Interface]

[Level Interface]

[MDStore Interface]
Role Interface
**ContainedClassType (clsCollection)**

The **ContainedClassType** property of a Decision Support Objects (DSO) collection returns the class type of the items contained within the collection. This property does not apply to **CustomProperties** collections.

**Data Type**

ClassTypes

**Access**

Read-only

**Remarks**

The DSO object model uses the properties **ClassType** and **SubClassType** to identify the object. All DSO objects, with the exception of those of **ClassType clsCube, clsLevel, and clsMiningModel**, return a **SubClassType** of **sbclsRegular**. In addition to **sbclsRegular**, an object of **ClassType clsCube** or **clsLevel** can have a **SubClassType** of **sbclsVirtual**, which identifies the object as a virtual cube or a virtual (calculated) level. Objects of **ClassType clsMiningModel** return a **SubClassType** of **sbclsOLAP** or **sbclsRelational**, depending on the type of mining model defined by the object.

**Example**

Use the following code to return a collection object's **ContainedClassType** and determine which class type has been returned:

```vba
Dim ctVar As ClassTypes
ctVar = CollectionObject.ContainsClassType
Select Case ctVar
    Case clsCubeMeasure
        ' Insert code for a cube measure.
```
Case clsCubeDimension
   ' Insert code for a cube dimension.
Case clsCubeLevel
   ' Insert code for a cube level.
Case clsCubeCommand
   ' Insert code for a cube command.
Case clsCubeRole
   ' Insert code for a cube role.
Case Else
   ' Insert code for other objects.
End Select

See Also

ClassTypes
clsDataSource
Collections
Command Interface
Dimension Interface
Level Interface
MDStore Interface
Role Interface
**Count (clsCollection)**

The **Count** property of a Decision Support Objects (DSO) collection returns the number of items in the collection. This property does not apply to **CustomProperties** collections.

**Data Type**

Integer

**Access**

Read-only

**Note** All collections in the DSO object model are one-based. That is, the first item in the collection has an index of 1 and the last item has an index of Count.

**Example**

Use the following code to return the number of cubes in an **MDStores** collection of cubes:

```vbscript
'Assume an object (dsoDB) of ClassType clsDatabase exists
Dim dsoCube As DSO.MDStore
Dim cubeCounter As Integer
For cubeCounter = 1 to dsoDB.MDStores.Count
    Set dsoCube = dsoDB.MDStores(cubeCounter)
    Debug.Print "  Cube Name: " & dsoCube.Name
Next cubeCounter
```

**See Also**

[Collections](#)
Analysis Services Programming

**clsColumn**

Data mining column objects (that is, objects of ClassType clsColumn), along with data mining model objects, provide a programmatic interface to data mining capabilities. Data mining automates data analysis by applying algorithms to reveal historical and predictive patterns within large databases. The class type clsMiningModel is provided by Decision Support Objects (DSO) to represent data mining models. Data mining models are the primary objects for predictive analysis, just as a cube is the primary object for OLAP analysis. Objects of ClassType clsColumn are used to define the structure of mining model objects through the Columns collection of the model. Columns are provided with collections and properties through their default interface, the Column interface, for interacting with them. There are no methods associated with clsColumn objects.

**Remarks**

An object of ClassType clsColumn can have a SubClassType of sbclsRegular or sbclsNested. A column of SubClassType sbclsRegular is an individual data column, whereas a column of SubClassType sbclsNested represents a nested table composed of multiple individual data columns.

You create column objects by declaring a variable as a clsColumn data type and then creating an instance of the object and adding it to the Columns collection of either a mining model object or another column object. The AddNew method of the Columns collection creates the instance, sets the name of the object to the name you provide, adds the object to the collection, establishes the SubClassType of the column, and sets its Parent property to reference the owner of the collection.

**Examples**

**Adding a New Column to a Data Mining Model**

The following example demonstrates how to add a new column to a data mining
model:
'------------------------------------------------------------------------
' Add a new column to the mining model called Gender and relate this
column to the Gender member property of the Name level of the
Customers dimension. Declare that the data in this column is
'statistically discrete.
' Assume the existence of a DSO Level object, dsoLvl.
'------------------------------------------------------------------------

'Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Gender")
'Identify the member property of the Customers dimension
'that this column is based on.
Set dsoColumn.SourceOlapObject = dsoLvl.MemberProperties("Ge")
'Identify its type.
dsoColumn.DataType = adWChar
'Make this column related to the Customer Id column.
dsoColumn.RelatedColumn = "Customer Id"
'Identify this column as one containing discrete data.
dsoColumn.ContentType = "DISCRETE"

See Also

AddNew
clsMiningModel
Collections, clsColumn
Data Mining Models
Properties, clsColumn
## Collections, clsColumn

An object of `ClassType clsColumn` supports the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns</strong></td>
<td>The collection of column objects that defines a nested table in the structure of a data mining model object. This collection applies only to columns of <code>SubClassType sbclsNested</code>.</td>
</tr>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties for the data mining model.</td>
</tr>
</tbody>
</table>
Properties, clsColumn

An object of `ClassType clsColumn` supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AreKeysUnique</code></td>
<td>Indicates whether key columns defined in the <strong>Columns</strong> collection uniquely identify members in the case table.</td>
<td>R/W†</td>
</tr>
<tr>
<td><code>ClassType</code></td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
<td>R</td>
</tr>
<tr>
<td><code>ContentType</code></td>
<td>Describes the content type of a column's data.</td>
<td>R/W*</td>
</tr>
<tr>
<td><code>DataType</code></td>
<td>The data type of the column.</td>
<td>R/W*</td>
</tr>
<tr>
<td><code>Description</code></td>
<td>The description of the column.</td>
<td>R/W*</td>
</tr>
<tr>
<td><code>Distribution</code></td>
<td>Identifies the statistical distribution of a column's data.</td>
<td>R/W</td>
</tr>
<tr>
<td><code>Filter</code></td>
<td>Filters the rows used in the nested table.</td>
<td>R/W*</td>
</tr>
<tr>
<td><code>FromClause</code></td>
<td>Specifies the FROM clause of the SQL query that returns a nested table for a column.</td>
<td>R/W*</td>
</tr>
<tr>
<td><code>IsDisabled</code></td>
<td>Specifies whether a column is disabled for training purposes.</td>
<td>R/W</td>
</tr>
<tr>
<td><code>IsInput</code></td>
<td>Indicates whether a column can accept input values for training a mining model object. For more information, see <code>IsPredictable</code>.</td>
<td>R/W</td>
</tr>
<tr>
<td><code>IsKey</code></td>
<td>Indicates whether or not the column is a key column in a case table or a nested table.</td>
<td>R/W*</td>
</tr>
<tr>
<td><code>IsParentKey</code></td>
<td>Indicates whether the column is a foreign key that relates to the case table.</td>
<td>R or R/W*</td>
</tr>
<tr>
<td><code>IsPredictable</code></td>
<td>Indicates whether this column can be predicted based on other input columns.</td>
<td>R/W</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Access</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>JoinClause</td>
<td>Specifies the JOIN clause of the SQL query that returns a nested table for a column.</td>
<td>R/W*</td>
</tr>
<tr>
<td>ModelingFlags</td>
<td>Specifies modeling options for a column.</td>
<td>R/W</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the column.</td>
<td>R/W</td>
</tr>
<tr>
<td>Num</td>
<td>The ordinal position of the column.</td>
<td>R/W</td>
</tr>
<tr>
<td>Parent</td>
<td>The parent mining model or column object.</td>
<td>R</td>
</tr>
<tr>
<td>RelatedColumn</td>
<td>The column to which a column is related.</td>
<td>R/W*</td>
</tr>
<tr>
<td>SourceColumn</td>
<td>The name of the column's source column in a relational table.</td>
<td>R/W**†</td>
</tr>
<tr>
<td>SourceOlapObject</td>
<td>The name of a column's source Decision Support Objects (DSO) object.</td>
<td>R/W††</td>
</tr>
<tr>
<td>SpecialFlag</td>
<td>Identifies the statistical nature of a column's data.</td>
<td>R/W</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type.</td>
<td>R</td>
</tr>
</tbody>
</table>

* This property applies only to columns of SubClassType sbclsRegular.
† This property applies only to columns belonging to ClassType clsMiningModel objects of SubClassType sbclsRegular.
†† This property applies only to columns belonging to ClassType clsMiningModel objects of SubClassType sbclsOlap.
AreKeysUnique (clsColumn)

The AreKeysUnique property of a clsColumn object indicates whether key columns (that is, a clsColumn object with an IsKey property set to True) defined in the Columns collection uniquely identify members in the case table.

Note This property applies only to columns that belong to mining model objects of SubClassType sbclsRelational.

Data Type

Boolean

Access

Read/write for columns with a SubClassType of sbclsNested, read-only for all others.

Remarks

The AreKeysUnique property determines whether the relational mining model adds the DISTINCT keyword to the SQL SELECT query used to retrieve the training data set from the case tables. If the values for the key columns identified in the data mining model are unique in the case tables, setting this property to True can improve performance when the relational data mining model is trained.

For columns with a SubClassType of sbclsRegular, this property returns the AreKeysUnique property value of the parent, either an object of ClassType clsColumn with a SubClassType of sbclsNested or an object of ClassType clsMiningModel.

See Also

clsColumn
ClassType (clsColumn)

The ClassType property of a clsColumn object returns an enumeration constant that identifies the specific class type.

Data Type

ClassTypes

Access

Read-only

Remarks

The ClassType property always returns clsColumn for column objects.

See Also

clsColumn
ContentType (clsColumn)

The **ContentType** property of an object of **ClassType clsColumn** describes the content type of a column's data.

**Data Type**

String

**Access**

Read/write for columns with a **SubClassType** of **sbclsRegular** whose **IsKey** and **IsParentKey** properties are False, read-only for all others.

**Remarks**

For columns whose **IsKey** or **IsParentKey** properties are set to True and columns with a **SubClassType** of **sbclsNested**, this property returns an empty string.

This property suggests the column contents to the mining model. This suggestion is used to optimize the mining model's **MiningAlgorithm** property and must be specified for each column.

Supported values for this property are listed in the MINING_SERVICES schema rowset in the SUPPORTED_TYPE_FLAGS column.

**Examples**

**Identifying the Content Type of a Mining Model Column**

The following example demonstrates how to create a new column and set its **ContentType** property to CONTINUOUS:

'------------------------------------------------------------------------
' Add a new column to the mining model called Unit Sales and relate
'this column to the Sales cube measure of the same name. Set the
columns data type to Integer, and identify the data content in it as
being continous and logarithmically normalized. Finally, identify this
column as being predictable.
Assume the existence of a DSO Cube object, dsoCb.

'------------------------------------------------------------------------

'Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Unit Sales")
'Identify this column as being based on the Unit Sales measure.
Set dsoColumn.SourceOlapObject = dsoCb.Measures("Unit Sales")
'Identify the column type.
dsoColumn.DataType = adInteger
'Identify this column's content as being continuous.
dsoColumn.ContentType = "CONTINUOUS"
'Identify the statistical distribution of this data.
dsoColumn.Distribution = "LOG_NORMAL"
'Identify the column as being predictable.
dsoColumn.IsPredictable = True

See Also

clsColumn
Data Mining Schema Rowsets
MINING_SERVICES
MiningAlgorithm
Analysis Services Programming

**DataType (clsColumn)**

The `DataType` property identifies the data type of an object of `ClassType clsColumn`. This property applies only to columns of `SubClassType sbclsRegular`.

**Data Type**

`ADODB.DataTypeEnum`

**Access**

Read/write

**Remarks**

Values for the `DataType` property are supplied by the Microsoft® ActiveX® Data Objects (ADO) `DataTypeEnum` enumeration constants. For more information, see the ADO documentation.

**Examples**

**Setting the DataType Property**

The following example adds a new column to a data mining model object. It then sets various properties, including the `DataType` property.

'------------------------------------------------------------------------
' Add a new column to the mining model called Unit Sales and relate
' this column to the Sales cube measure of the same name. Set the
' column's data type to Integer, and identify the data content in it as
' being continuous and logarithmically normalized. Finally, identify thi
' column as being predictable.
' Assume the existence of a DSO Cube object, dsoCb.
'------------------------------------------------------------------------
'Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Unit Sales")
'Identify this column as being based on the Unit Sales measure.
Set dsoColumn.SourceOlapObject = dsoCb.Measures("Unit Sales")
'Identify the column type.
dsoColumn.DataType = adInteger
'Identify this column's content as being continuous.
dsoColumn.ContentType = "CONTINUOUS"
'Identify the statistical distribution of this data.
dsoColumn.Distribution = "LOG_NORMAL"
'Identify the column as being predictable.
dsoColumn.IsPredictable = True

See Also

clsColumn
**Description (clsColumn)**

The **Description** property of an object of **ClassType clsColumn** sets or returns the description of the column. This property is reserved for future reference in Decision Support Objects (DSO) and is not available to client applications.

**Data Type**

String

**Access**

Read/write

**Examples**

**A. Setting the Description Property**

Use the following code to set the **Description** property for a **clsColumn** object:

' Assume an object (dsoColumn) of ClassType clsColumn exists
dsoColumn.Description = "Number Sold"

**B. Adding a Column to a Mining Model and Setting the Column D**

The following example creates a new column in the mining model and sets a number of properties, including the **Description** property:

' Assume the existence of a DSO Level object, dsoLv.
Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Gender")

Identify the member property of the Customers dimension
that this column is based on.
Set the column's description for browsers of the schema.
Set dsoColumn.Description = "Based on the Gender member proper
"of the Name level of the Customers dimension."
Set dsoColumn.SourceOlapObject = dsoLvl.MemberProperties("Ge")

Identify its type.
dsoColumn.DataType = adWChar

Make this column related to the Customer Id column.
dsoColumn.RelatedColumn = "Customer Id"

Identify this column as containing discrete data.
dsoColumn.ContentType = "DISCRETE"

See Also

clsColumn
Distribution (clsColumn)

The Distribution property of an object of ClassType clsColumn identifies the statistical distribution of the column's data.

Data Type

String

Access

Read/write for columns with a SubClassType of sbclsRegular whose IsKey and IsParentKey properties are False, read-only for all others.

Remarks

For columns whose IsKey or IsParentKey properties are set to True and columns with a SubClassType of sbclsNested, this property returns an empty string.

Access

Read/write

Remarks

This property specifies the column's statistical distribution. This is used to optimize performance by the mining model's mining algorithm and can be left unspecified.

This property applies only to columns of SubClassType sbclsRegular, if they are not used as key or parent key columns. (That is, it applies only to regular columns whose IsKey and IsParentKey properties are False.) Supported values for this property are listed in the MINING_SERVICES schema rowset in the SUPPORTED_DISTRIBUTION_FLAGS column.
Examples

**Setting the Distribution Property**

The following example creates a new column and sets its `Distribution` property, among others:

'------------------------------------------------------------------------
' Add a new column to the mining model called Unit Sales and relate
' this column to the Sales cube measure of the same name. Set the
' column's data type to Integer, and identify the data content in it as
' being continous and logarithmically normalized. Finally, identify this
' column as being predictable.
' Assume the existence of a DSO Cube object, dsoCb.
'------------------------------------------------------------------------

'Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Unit Sales")
'Identify this column as being based on the Unit Sales measure.
Set dsoColumn.SourceOlapObject = dsoCb.Measures("Unit Sales")
'Identify the column type.
dsoColumn.DataType = adInteger
'Identify this column's content as being continuous.
dsoColumn.ContentType = "CONTINUOUS"
'Identify the statistical distribution of this data.
dsoColumn.Distribution = "LOG_NORMAL"
'Identify the column as being predictable.
dsoColumn.IsPredictable = True

See Also

- clsColumn
- Data Mining Schema Rowsets
- MiningAlgorithm
Filter (clsColumn)

The Filter property of an object of ClassType clsColumn specifies a filter condition that is applied to the SQL query that returns the cases for the mining model.

**Note** This property applies only to columns that belong to mining model objects of SubClassType sbclsRegular.

**Data Type**

String

**Access**

Read/write for columns with a SubClassType of sbclsNested, read-only for all others.

**Remarks**

For columns with a SubClassType of sbclsRegular, this property returns the Filter property of the parent object. Columns can be nested, so the parent object can be either a clsMiningModel object or a clsColumn object.

**See Also**

clsColumn
FromClause (clsColumn)

The FromClause property of an object of ClassType clsColumn specifies the FROM clause of the SQL query that returns a nested table.

Note This property applies only to columns that belong to mining model objects of SubClassType sbclsRegular.

Data Type

String

Access

Read/write for columns with a SubClassType of sbclsNested, read-only for all others.

Remarks

For columns with a SubClassType of sbclsRegular, this property returns the Filter property of the parent object. Columns can be nested, so the parent object can be either a clsMiningModel object or a clsColumn object.

Examples

Creating a New Nested Column

The following code creates a new nested column called Products. It uses the FromClause and JoinClause properties to establish the SQL joins to the parent table. It then creates a new column called CustomerID and establishes that this column contains key values from the parent table by setting the IsParentKey property to TRUE. The clsColumn object that contains the keys in the parent table is referred to by the value of the RelatedColumn property: KeyColumn.

' Create a new nested column.
Set dsoNestedCol = dsoDmm.Columns.AddNew("Products", sbclsNes
dsoNestedCol.FromClause = """"Sales"""", """"SalesReps"""", """"Products"""

dsoNestedCol.JoinClause = """"Sales"".""SalesRep"" = """"SalesReps"".""Name"" AND """"Sales"".""Product"" = """"Products"".""Product""

dsoNestedCol.Filter = ""

' Create a new column that contains key values from the parent table. Set dsoColumn = dsoNestedCol.Columns.AddNew("CustomerID")

dsoColumn.SourceColumn = """"Products"".""CustId""

dsoColumn.DataType = adInteger

dsoColumn.IsParentKey = True

' The RelatedColumn property is set to the clsColumn object used as the key column for the data mining model.

dsoColumn.RelatedColumn = "KeyColumn"

See Also

clsColumn
IsDisabled (clsColumn)

The IsDisabled property of an object of ClassType clsColumn specifies whether the column is included in the mining model or is only used during training to specify joins between tables.

Data Type
Boolean

Access
Read/write

Remarks
Columns where the IsDisabled property has been set to True are ignored when a mining model is being created or trained on the Analysis server. This property setting is useful for OLAP data mining models (that is, objects of ClassType clsMiningModel and SubClassType sbclsOlap) when the Columns collection is created automatically by calling the Update method. After the Update method is called, columns to be used by the data mining model can be enabled by setting the IsDisabled property to False.

For columns of SubClassType sbclsNested, setting the IsDisabled property to True automatically disables descendant columns. Setting the IsDisabled property to False automatically enables ancestor columns.

See Also

clsColumn
Update
IsInput (clsColumn)

The IsInput property of an object of ClassType clsColumn indicates whether the column can accept input values when carrying out predictions.

Data Type

Boolean

Access

Read/write

Remarks

A column can have both the IsInput and the IsPredictable properties set to True.

Note  All columns are considered as input columns when training a mining model unless they are disabled. It is only when predictions are carried out against a mining model that the notions of IsInput or IsPredictable have any meaning.

The value of the IsInput property can be related to other properties of the object, as well as properties of the parent object. Changing the property can also affect the properties of related objects, including parent objects.

For columns with a SubClassType of sbclsRegular, if the column is related to a column that is not a key column, the value of this property is equal to the value of the IsInput property of the related column. If the parent of the column is a clsColumn object (that is, the column is a child of a nested column) and the IsKey property is True, the value of this property is equal to the value of the IsInput property of the parent column. If the parent of the column is a clsMiningModel object and the IsKey property is True, the value of this property is False. If the IsParentKey property of this column is True, this property is False.
Changing the **IsInput** property to True for a column (other than a key column) whose parent is a **clsColumn** object (that is, the column is a child of a nested column) changes the **IsInput** property of the parent column to True.

For columns with a **SubClassType** of **sbclsNested**, changing the **IsInput** property to False changes the **IsInput** property for all child columns whose **IsKey**, **IsParentKey**, and **IsRelated** properties are all False.

See Also

- **clsColumn**
- **IsDisabled**
- **IsPredictable**
IsKey (clsColumn)

The IsKey property of an object of ClassType clsColumn indicates whether the column is a key column in the case table or in a nested table.

Data Type

Boolean

Access

Read/write for columns with a SubClassType of sbclsRegular that belong to a clsMiningModel object with a SubClassType of sbclsRegular, read-only for all others.

Remarks

A key column is a column that uniquely identifies each row in the case table. There can be more than one key column in a row. For example, to uniquely identify a customer it may be necessary to use both the name column and address column of a customer record as the keys. In a nested table, the key column with a parent key column (using the IsParentKey property) is used to uniquely identify the rows of the nested table and relate them to the case table.

The value of IsKey can vary based on the SubClassType property of the column and the properties of the parent object.

The IsKey property is always False for columns with a SubClassType of sbclsNested. If the parent object is an OLAP mining model (a clsMiningModel object with a SubClassType of sbclsOlap), the IsKey property returns True only if the column is associated with the lowest enabled level of the case dimension (that is, the SourceColumn property of the column matches the CaseLevel property of the parent clsMiningModel object).

Examples
Adding a New Column

The following example adds a new column, Customer Id, to the `Columns` collection of a mining model object. It then sets the `IsKey` property and other important properties.

```
' Add a new column to the mining model called Customer Id and relate
' this column to the Name level of the Customers dimension.
' Describe the level's type and make it a key for the model.
' Assume that a DSO level object already exists, called dsoLvl.
'------------------------------------------------------------------------

'Add Customer Id as a new column in the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Customer Id")
'Identify the level in Sales that this column is based on.
Set dsoColumn.SourceOlapObject = dsoLvl
'Identify the type of column this is.
dsoColumn.DataType = adInteger ' This enumeration is from ADO.
'Identify this column as a key.
dsoColumn.IsKey = True
```

See Also

clsColumn
IsParentKey (clsColumn)

The IsParentKey property of an object of ClassType clsColumn indicates whether the column is a foreign key that relates to a column in the case table. This property, when used with the IsKey property, serves to uniquely identify the rows in a nested table.

Data Type

Boolean

Access

Read/write for columns with a SubClassType of sbclsRegular that belong to a clsMiningModel object with a SubClassType of sbclsRegular, read-only for all others.

Remarks

The IsParentKey property always returns False for columns with a SubClassType of sbclsNested and for columns belonging to a mining model with a SubClassType of sbclsOlap.

The IsParentKey property indicates which column in the nested table contains the foreign key to the case table, and it assists in creating the SQL JOIN clause used for the training query. For example, if a data mining model is constructed from the store table, with a key column named Store ID based on the store_id field, and a nested column based on the sales_fact_1998 table is added, a column related to the Store ID column (that is, a clsColumn object with a RelatedColumn property set to "Store ID") is created with the source column set to the store_id field in the sales_fact_1998 table and the IsParentKey set to True.

Examples
Creating a Data Mining Model With a Nested Column

The following code example creates a new relational data mining model named Test Model in the **FoodMart 2000** database. Test Model is based on the **store** table in the **FoodMart** data source. The nested column **Sales Fact 1998**, based on the **sales_fact_1998** table, contains a parent key column named **Parent Store ID**. The **Parent Store ID** column, defined from the **store_id** column in the **sales_fact_1998** table, is related to the **Store ID** column in the data mining model defined from the **store_id** column in the **store** table.

```vba
Public Sub CreateDMMWithNestedColumn()
    Dim dsoServer As DSO.Server
    Dim dsoDB As DSO.Database
    Dim dsoDMM As DSO.MiningModel
    Dim dsoColumn As DSO.Column
    Dim dsoNestedColumn As DSO.Column

    ' Initialize server.
    Set dsoServer = New DSO.Server

    ' Connect to the local Analysis server.
    ' If a connection cannot be made, an error is raised.
    dsoServer.Connect "LocalHost"

    ' Connect to the FoodMart 2000 database.
    Set dsoDB = dsoServer.MDStores("FoodMart 2000")

    ' Create a new relational data mining model.
    Set dsoDMM = dsoDB.MiningModels.AddNew("Test Model", sbclsRelational)

    ' Set the properties for the data mining model.
    With dsoDMM
        .FromClause = """"store"""
        .MiningAlgorithm = "Microsoft_Decision_Trees"
        .DataSources.Add dsoDB.DataSources("FoodMart")
    End With

End Sub
```
End With

' Create the key and predictable columns for the mining model.
Set dsoColumn = dsoDMM.Columns.AddNew("Store ID", sbclsRegular)

With dsoColumn
    .SourceColumn = """store"""".""store_id"""
    .DataType = adInteger
    .IsKey = True
End With

Set dsoColumn = dsoDMM.Columns.AddNew("Store Type", sbclsRegular)

With dsoColumn
    .SourceColumn = """store"""".""store_type"""
    .DataType = adWChar
    .IsKey = False
    .IsInput = True
    .IsPredictable = True
    .ContentType = "DISCRETE"
End With

' Create the nested column.
Set dsoColumn = dsoDMM.Columns.AddNew("Sales Fact 1998", sbclsNested)

With dsoColumn
    .FromClause = """sales_fact_1998"""
    .IsInput = True
    .IsPredictable = False
End With

' Create the parent key column for the nested column.
Set dsoNestedColumn = dsoColumn.Columns.AddNew("Store ID",
' Set the properties for the parent key column.
With dsoNestedColumn
    .SourceColumn = """sales_fact_1998"".""store_id""
    .DataType = adInteger
    .IsKey = False
    .IsInput = False
    .IsPredictable = False
    .IsParentKey = True
    .RelatedColumn = "Store ID"
End With

' Create the key and predictable columns for the nested column.
Set dsoNestedColumn = dsoColumn.Columns.AddNew("Product ID")

With dsoNestedColumn
    .SourceColumn = """sales_fact_1998"".""product_id""
    .DataType = adInteger
    .IsKey = True
    .IsInput = True
    .IsPredictable = False
    .IsParentKey = False
End With

Set dsoNestedColumn = dsoColumn.Columns.AddNew("Store Sales")

With dsoNestedColumn
    .SourceColumn = """sales_fact_1998"".""store_sales""
    .DataType = adInteger
    .ContentType = "CONTINUOUS"
    .IsKey = False
    .IsInput = True
    .IsPredictable = False
.IsParentKey = False
End With

' Save the new data mining model.
dsoDMM.Update

' Process the data mining model.
dsoDMM.Process

End Sub

See Also

clsColumn
IsPredictable (clsColumn)

The `IsPredictable` property of an object of `ClassType clsColumn` indicates whether the column's parent mining model object can predict the column's value based on other input columns.

**Data Type**

Boolean

**Access**

Read/write

**Remarks**

A column can have both the `IsPredictable` and the `IsInput` properties set to True.

**Note** All columns are considered as input columns when training a mining model unless they are disabled. It is only when predictions are carried out against a mining model that the notions of `IsInput` or `IsPredictable` have any meaning.

The value of the `IsPredictable` property can be related to other properties of the object, as well as properties of the parent object. Changing the property can also affect the properties of related objects, including parent objects.

For columns with a `SubClassType` of `sbclsRegular`, if the column is related to a column that is not a key column, the value of this property is equal to the value of the `IsPredictable` property of the related column. If the parent of the column is a `clsColumn` object (that is, the column is a child of a nested column) and the `IsKey` property is True, the value of this property is equal to the value of the `IsPredictable` property of the parent column. If the parent of the column is a `clsMiningModel` object and the `IsKey` property is True, the value of this property is False. If the `IsParentKey` property of this column is True, this property is False.
Changing the **IsPredictable** property to True for a column (other than a key column) whose parent is a **clsColumn** object (that is, the column is a child of a nested column) changes the **IsPredictable** property of the parent column to True.

For columns with a **SubClassType** of **sbclsNested**, changing the **IsPredictable** property to False changes the **IsPredictable** property for all child columns whose **IsKey**, **IsParentKey**, and **IsRelated** properties are all False.

**Examples**

**Adding a Column to the Columns Collection**

The following example adds a column called **Unit** to a data mining model's **Columns** collection **Sales**. It then enables the column by setting its **IsDisabled** property to False and makes the column predictable by setting its **IsPredictable** property to True.

'Make the Unit Sales measure predictable.
Set dsoColumn = dsoDmm.Columns("Unit Sales")
'Enable the column.
dsoColumn.IsDisabled = False
'Make the column predictable.
dsoColumn.IsPredictable = True

**See Also**

**clsColumn**
JoinClause (clsColumn)

The JoinClause property of an object of ClassType clsColumn specifies the JOIN clause of the SQL query that returns a nested table for the column. This property applies to columns that belong to mining model objects of SubClassType sbclsRegular.

Data Type

String

Access

Read/write

Remarks

This property is read/write only for nested columns (columns of SubClassType sbclsNested). For regular columns (columns of SubClassType sbclsRegular), this property is read-only and returns the JoinClause property of the column's parent object.

Examples

Creating a Nested Column

The following example creates a nested column and establishes two joins to the parent columns based on the SalesRep column and the Product column:

Set dsoNestedCol = dsoDmm.Columns.AddNew("Products", sbclsNes
dsoNestedCol.FromClause = "Sales, SalesReps, Products"
dsoNestedCol.JoinClause = "Sales.SalesRep = SalesReps.Name AND
dsoNestedCol.Filter = ""

See Also
clsColumn
ModelingFlags (clsColumn)

The **ModelingFlags** property of an object of **ClassType clsColumn** specifies options for modeling a column's data in a mining model.

**Data Type**

String

**Access**

Read/write for columns with a **SubClassType** of **sbclsRegular** whose **IsKey** and **IsParentKey** properties are False, read-only for all others.

**Remarks**

For columns whose **IsKey** or **IsParentKey** properties are set to True and columns with a **SubClassType** of **sbclsNested**, this property returns an empty string.

**Access**

Read/write for columns with a **SubClassType** of **sbclsRegular** whose **IsKey** and **IsParentKey** properties are False, read-only for all others.

**Remarks**

This property is a comma-delimited list of modeling option values for the column, used to optimize the mining model algorithm (specified by the **MiningAlgorithm** property of the **clsMiningModel** object) and can be left unspecified.

Supported values for this property are listed in the MINING SERVICES schema rowset in the SUPPORTED_MODELING_FLAGS column.

**See Also**
clsColumn

Data Mining Schema Rowsets

MiningAlgorithm
Analysis Services Programming

Name (clsColumn)

The Name property of an object of ClassType clsColumn contains the name of the column as it will appear in the mining model.

Data Type

String

Access

Read/write (read-only after object is named)

Example

Use the following code to return a level object name:

' Assume an object (dsoColumn) of ClassType clsColumn exists
Dim strName As String
strName = dsoColumn.Name

See Also

clsColumn
Num (clsColumn)

The Num property of a clsColumn object returns the ordinal position of the column in relation to its parent.

Data Type

Integer

Access

Read/write

Remarks

If a column belongs to the nested table of another column, the Num property indicates the ordinal position of the column within the nested table of the parent column. For a column that is used to define a mining model, the Num property indicates the ordinal position of the column within the mining model.

See Also

clsColumn
clsMiningModel
Parent (clsColumn)

The Parent property of a clsColumn returns a reference to the parent of the column.

Data Type
Object

Access
Read-only

Remarks
If a column belongs to another column's nested table, the Parent property returns a reference to a clsColumn object. For a column that is used to define a mining model, the Parent property returns a reference to a clsMiningModel object.

See Also

clsColumn
clsMiningModel
Analysis Services Programming

**RelatedColumn (clsColumn)**

The **RelatedColumn** property of a **clsColumn** identifies a column to which the column is related.

**Data Type**

String

**Access**

Read/write for columns with a **SubClassType** of **sbclsRegular** whose **IsKey** property is False, read-only for all others.

**Remarks**

For columns with a **SubClassType** of **sbclsRegular** whose **IsKey** property is set to True and for columns with a **SubClassType** of **sbclsNested**, this property returns an empty string.

The functionality of the **RelatedColumn** property differs depending on the context of its usage:

- The **RelatedColumn** property is used to relate a column in a nested table to a column in the case table (that is, the parent table) of the data mining model. In this case, the column's **IsParentKey** property is set to True.

- The **RelatedColumn** property is used to define hierarchical relationships between columns. For example, you can use it to define that the **Region** column is related to the **State** column, the **State** column is related to the **City** column, and so on. For another example, consider a case set involving customer purchases. If **ProductName** is a column defined in the model, a column called **ProductType** can have its **RelatedColumn** property set to the **ProductName** column to indicate that its information is related to the **ProductName** column.
The **SpecialFlag** property is used with the **RelatedColumn** property. Consider the example in which a column is defined using the **SpecialFlag** property to contain a probability. In this case, the **RelatedColumn** property is used to determine which column the probability is based on. If a column is defined that is related to the **CreditRisk** column and contains a probability, the column would contain the numeric probability of a given credit for a given case.

**Examples**

**A. Creating a Key Column and Relating it to a Key in the Case Table**

The following example creates a key column in the case table for a mining model. It then creates a nested table based on three different tables and establishes the relationships between them (that is, their joins). Finally, it establishes a key column within this nested table and relates it to the key column in the case table.

'Define the key column for the case table.
Set dsoColumn = dsoDmm.Columns.AddNew("KeyColumn")
dsoColumn.SourceColumn = "Key"
dsoColumn.DataType = adInteger
dsoColumn.IsKey = True

'Define a nested table and relate the tables it is based on in a join.
Set dsoNestedCol = dsoDmm.Columns.AddNew("Products", sbclsNes)
dsoNestedCol.FromClause = "Sales, SalesReps, Products"
dsoNestedCol.Filter = ""

'Create a parent key column for the nested table and relate it to a column.
Set dsoColumn = dsoNestedCol.Columns.AddNew("CustomerID")
dsoColumn.SourceColumn = "CustId"
dsoColumn.DataType = adInteger
dsoColumn.IsParentKey = True
dsoColumn.RelatedColumn = "KeyColumn"

**B. Establishing a Hierarchical Relationship Between Columns in a**

The following example builds a hierarchical relationship between the columns as they are added to a nested table. The following diagram shows their structure.

Set dsoColumn = dsoNestedCol.Columns.AddNew("Product Name")
dsoColumn.SourceColumn = "Sales.Product"
dsoColumn.DataType = adWChar
dsoColumn.IsKey = True

Set dsoColumn = dsoNestedCol.Columns.AddNew("Product Type")
dsoColumn.SourceColumn = "Products.Type"
dsoColumn.DataType = adWChar
dsoColumn.RelatedColumn = "Product Name"

Set dsoColumn = dsoNestedCol.Columns.AddNew("Product Category")
dsoColumn.SourceColumn = "Products.Category"
dsoColumn.DataType = adWChar
dsoColumn.RelatedColumn = "Product Type"

Set dsoColumn = dsoNestedCol.Columns.AddNew("Aisle")
dsoColumn.SourceColumn = "Products.Aisle"
dsoColumn.DataType = adWChar
dsoColumn.RelatedColumn = "Product Name"

**C. Establishing a Probabilistic Relationship**

The following example adds a column to a nested table. It then adds a second column whose contents will contain a probability based upon the first column.
Set dsoColumn = dsoNestedCol.Columns.AddNew("Quantity")
dsoColumn.SourceColumn = "Sales.Quantity"
dsoColumn.DataType = adDouble
dsoColumn.ContentType = "CONTINUOUS"

Set dsoColumn = dsoNestedCol.Columns.AddNew("pQuantity")
dsoColumn.SourceColumn = "Sales.pQuantity"
dsoColumn.DataType = adDouble
dsoColumn.RelatedColumn = "Quantity"
dsoColumn.SpecialFlag = "PROBABILITY"

See Also

clsColumn
SourceColumn (clsColumn)

The SourceColumn property of an object of ClassType clsColumn identifies the name of its source column in a relational table. This property applies only to columns belonging to mining model objects of SubClassType sbclsRegular.

Data Type

String

Access

Read/write for columns with a SubClassType of sbclsRegular, read-only for all others.

Remarks

For columns with a SubClassType of sbclsNested that belong to a mining model object of SubClassType of sbclsRegular, this property returns an empty string.

To understand the function of this property, consider the relationships of columns in a model to an SQL query. If you use a SELECT query to define the structure of a table when you create a mining model, the contents of this property for each column in the model correspond to a column designation within the SELECT query. For example, consider the following query:

   SELECT "Key" AS "CustId", "Age" AS "Age" FROM "People"

If a mining model were to be created using this SELECT statement, the SourceColumn properties for each column would be "Key" and "Age" respectively.

Examples

Creating a Data Mining Model
The following example creates a data mining model based upon the **People** table of a relational database. This table is specified by the **FromClause** property. Because the model is based upon a single table, no joins are needed. It then creates and adds two columns to the model's **Columns** collection. Each column is related to a field in the original relational table (that is to say, the **People** table) by setting the **SourceColumn** property of each column to the appropriate value.

```vba
dsoDmm.Description = "Analyzes the purchasing behavior of customers"
dsoDmm.MiningAlgorithm = "Microsoft_Decision_Trees"
dsoDmm.FromClause = "People"
dsoDmm.JoinClause = "" 'None is needed because there is only a single table.'
dsoDmm.Filter = ""
dsoDmm.TrainingQuery = "" 'Let DSO figure out the training query.'
```

Set `dsoColumn = dsoDmm.Columns.AddNew("CustId")`
`dsoColumn.SourceColumn = "People.Key"
dsoColumn.DataType = adInteger
dsoColumn.IsKey = True

Set `dsoColumn = dsoDmm.Columns.AddNew("Age")`
`dsoColumn.SourceColumn = "People.Age"
dsoColumn.DataType = adDouble
dsoColumn.ContentType = "CONTINUOUS"
```

**See Also**

[clsColumn](#)
SourceOlapObject (clsColumn)

The SourceOlapObject property of an object of ClassType clsColumn identifies the source Decision Support Objects (DSO) object for the column. This property only applies to columns that belong to mining model objects of SubClassType sbclsOlap.

Data Type
Object

Access
Read/write

Remarks
The SourceOlapObject property of a column represents the source object in DSO from which the Column object draws information. The SourceOlapObject property can be set to an object with a ClassType property of:

- clsCubeDimension
- clsCubeLevel
- clsCubeMeasure
- clsMemberProperty

Any object specified in the SourceOlapObject property must be visible (that is, the isVisible property of the object must be True). If the isVisible property of the object is False, an error is raised.

Examples
Adding a New Column to a Data Mining Model

The following example adds a new column to a data mining model and sets its source to a level in an OLAP cube.

'Add Customer Id as a new column in the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Customer Id")
'Identify the level in Sales that this column is based on.
Set dsoColumn.SourceOlapObject = dsoLvl
'Identify the type of column this is.
dsoColumn.DataType = adInteger ' This enumeration is from ADO.
'Identify this column as a key.
dsoColumn.IsKey = True

See Also

clsColumn
SpecialFlag (clsColumn)

The SpecialFlag property assists the Distribution property in identifying the statistical nature of the column's data values for model optimization purposes.

Data Type
String

Access
Read/write for columns with a SubClassType of sbclsRegular whose IsKey and IsParentKey properties are False, read-only for all others.

Remarks
For columns whose IsKey or IsParentKey properties are set to True and columns with a SubClassType of sbclsNested, this property returns an empty string.

This property suggests the column contents to the mining model. This suggestion is used to optimize the mining model's MiningAlgorithm and can be left unspecified.

Supported values for this property are listed in the MINING_SERVICES schema rowset in the SUPPORTED_SPECIAL_FLAGS column.

Examples

Building a New Column

The following example builds a new column and sets its SpecialFlag property to PROBABILITY:

Set dsoColumn = dsoNestedCol.Columns.AddNew("pOn Sale")
dsoColumn.SourceColumn = "Sales.pOnSale"
dsoColumn.DataType = adDouble
dsoColumn.RelatedColumn = "On Sale"
dsoColumn.SpecialFlag = "PROBABILITY"

See Also

clsColumn

Data Mining Schema Rowsets

MiningAlgorithm
SubClassType (clsColumn)

The SubClassType property of an object of ClassType clsColumn returns an enumeration constant identifying the specific subclass type.

Data Type

SubClassTypes

Access

Read-only

Remarks

Objects of ClassType clsColumn can have a SubClassType property value of sbclsRegular or sbclsNested. A column has a SubClassType value of sbclsRegular if it is an individual column. If a column contains a nested table, it has a SubClassType value of sbclsNested.

See Also

clsColumn
An object of ClassType clsCube provides an implementation of the MDStore interface of the Decision Support Objects (DSO) library specific to cubes. Each instance of clsCube provides collections, methods, and properties through the MDStore interface.

**Example**

Use the following code to create a cube object (that is, an object of ClassType clsCube):

' Assume an object (dsoServer) of ClassType clsServer exists
' and contains a database in its MDStores collection
Dim dsoDB As DSO.MDStore ' Create an interface for the database.
Dim dsoCube As DSO.MDStore ' Create an interface for the cube.

' Assign the database interface to the first database
' in the server's collection of databases.
Set dsoDB = dsoServer.MDStores(1)
' Next, create the new cube by using the AddNew method
' of the database object's MDStores collection of cubes.
Set myCube = dsoDB.MDStores.AddNew("MyCube")
' Set properties and add dimensions, levels, and measures
'...

' Next, create a virtual cube.
Dim dsoVCube As DSO.MDStore ' Create an interface for the virtual cube.
' Use the AddNew method of the MDStores collection,
' just as before, but specify that the cube is virtual
' using the SubClassType argument sbclsVirtual.
Set dsoVCube = dsoDB.MDStores.AddNew("MyVCube", sbclsVirtual)
' Add measures, set properties, and add dimensions
Virtual Cubes

A cube object with a SubClassType of sbclsVirtual is a virtual cube. A virtual cube is used to encapsulate a subset of the measures, dimensions, and levels contained in a group of cubes. A virtual cube, like a view in a relational database, is a logical construct that contains no data. Just as a view joins multiple relations, a virtual cube joins multiple cubes.

The basic method for managing virtual cubes is to add them to a database with the SubClassType parameter set to sbclsVirtual. Then, you can add dimensions and measures to them as needed. However, the dimensions and measures are derived from previously defined cubes within the database, rather than from a dimension table. Any levels associated with a dimension that has been added to a virtual cube automatically apply to the dimension in the virtual cube. Partitions and aggregations do not apply to virtual cubes.

If you add or remove a dimension or a measure from a virtual cube, you must reprocess the virtual cube so that the change will affect the Analysis server operations. The same is true if you remove a dimension or a measure from a cube after assigning it to a virtual cube, or if you remove a dimension or a measure from a database after assigning it to a cube or virtual cube.

Linked Cubes

A cube object with a SubClassType of sbclsLinked is a linked cube. The contents of a linked cube are based on another cube that is defined and stored on a different Analysis server. Unlike a virtual cube, which can contain portions of one or more cubes, a linked cube references the entire contents of a single cube.

Example

Use the following code to create a linked cube. This procedure must involve two different servers. Attempting to create a link to a cube on the same server results in an error.

Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.MDStore ' Create an interface for the database.
Dim dsoLCube As DSO.MDStore ' Create an interface for the linked c
Dim dsoLDS As DSO.DataSource
' Connect to the server
dsoServer.Connect "localhost"

' Get a reference for the database that
' will contain the linked cube.
Set dsoDB = dsoServer.MDStores("FoodMart")

' Create a new data source for the linked cube.
Set dsoLDS = dsoDB.DataSources.AddNew("Linked Cube")

' Set the connection string, so that the data source points
' to an Analysis server running SQL Server 2000 Analysis Services.
dsoLDS.ConnectionString = "Provider=MSOLAP;Data Source=servername;
Initial Catalog=Foodmart;"

' Save this data source in the repository.
dsoLDS.Update

' Create a new cube on the local server, mark it as linked.
Set dsoLCube = dsoDB.MDStores.AddNew("Linked Sales", sbclsLinked)

' Add dsoLDS to the DataSources collection of the linked cube.
dsoLCube.DataSources.Add dsoLDS

' Use the name of the published cube as the
' source table for the subscribed cube.
dsoLCube.SourceTable = "" & "Sales" & ""

' Update the cube.
dsoLCube.Update

' Completely process the linked cube.
dsoLCube.Process processFull

**See Also**

[Cubes](#)

[MDStore Interface](#)

[Collections, clsCube](#)

[Methods, clsCube](#)

[Properties, clsCube](#)
Collections, clsCube

An object of `ClassType clsCube` implements the following collections of the `MDStore` interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commands</strong></td>
<td>The collection of command objects defined in the cube</td>
</tr>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td><strong>DataSources</strong></td>
<td>The collection of data source objects used by the cube</td>
</tr>
<tr>
<td><strong>Dimensions</strong></td>
<td>The collection of dimension objects defined in the cube</td>
</tr>
<tr>
<td><strong>MDStores</strong></td>
<td>The collection of <code>MDStore</code> objects defined for the cube</td>
</tr>
<tr>
<td><strong>Measures</strong></td>
<td>The collection of measure objects defined in the cube</td>
</tr>
<tr>
<td><strong>Roles</strong></td>
<td>The collection of role objects defined for the cube</td>
</tr>
</tbody>
</table>

See Also

- clsCube
- MDStore Interface
Methods, clsCube

An object of ClassType clsCube implements the following methods of the MDStore interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type. This method also creates a copy of the property value and has the option of creating collections of major and minor objects.</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks the cube to prevent multiple users from concurrently changing the object.</td>
</tr>
<tr>
<td>Process</td>
<td>Processes the cube.</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Releases a lock previously established by the LockObject method.</td>
</tr>
<tr>
<td>Update</td>
<td>Updates the cube's definition in the meta data repository.</td>
</tr>
</tbody>
</table>

See Also

clsCube

MDStore Interface
Properties, clsCube

An object of ClassType clsCube implements the following properties of the MDStore interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowDrillThrough</td>
<td>Indicates whether drillthrough is allowed on the cube.</td>
</tr>
<tr>
<td>AggregationPrefix</td>
<td>The prefix associated with an aggregation in a cube.</td>
</tr>
<tr>
<td>Analyzer</td>
<td>The cube analyzer object for this cube.</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
</tr>
<tr>
<td>DefaultMeasure</td>
<td>The name of the default measure for the cube.</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the cube.</td>
</tr>
<tr>
<td>DrillThroughColumns</td>
<td>List of columns that are included in a drillthrough query.</td>
</tr>
<tr>
<td>DrillThroughFilter</td>
<td>Statement restricting rows that are returned by a drillthrough query.</td>
</tr>
<tr>
<td>DrillThroughFrom</td>
<td>An SQL FROM clause with the names of the tables used in drillthrough queries.</td>
</tr>
<tr>
<td>DrillThroughJoins</td>
<td>An SQL JOIN clause with the names of the tables used in drillthrough queries.</td>
</tr>
<tr>
<td>EnableRealTimeUpdates</td>
<td>Indicates whether real-time updates are allowed on the cube.</td>
</tr>
<tr>
<td>EstimatedRows</td>
<td>The estimated number of rows in the cube.</td>
</tr>
<tr>
<td>EstimatedSize</td>
<td>The estimated size of the cube (estimated total size of all rows, in bytes).</td>
</tr>
<tr>
<td>FromClause</td>
<td>Contains the SQL FROM clause defining the list of tables used to define</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>Indicates whether the cube is read/write.</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>Indicates whether the cube should be stored in the repository.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the cube structure is valid.</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the cube is visible to a client.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>The JOIN clause (list of join conditions, separated by AND) for the cube.</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time when the cube was last processed.</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>User-specified date. It is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the cube.</td>
</tr>
<tr>
<td>OlapMode</td>
<td>Returns an enumeration constant identifying the type of OLAP storage mode.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent MDStore object.</td>
</tr>
<tr>
<td>ProcessingKeyErrorLimit</td>
<td>Sets the number of allowable errors that cause processing to cease.</td>
</tr>
<tr>
<td>ProcessingKeyErrorLogFileName</td>
<td>The universal naming convention (UNC) path to a file for logging dimension key errors encountered during processing.</td>
</tr>
<tr>
<td>ProcessingMode</td>
<td>Indicates whether the Analysis server should index and aggregate during processing or afterward.</td>
</tr>
<tr>
<td>Server</td>
<td>Returns a reference to the DSO.Server object.</td>
</tr>
<tr>
<td>SourceTable</td>
<td>The name of the fact table of the cube.</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>The alias of the source table for the cube.</td>
</tr>
</tbody>
</table>
| SourceTableFilter        | The SQL clause used to determine which
<table>
<thead>
<tr>
<th><strong>State</strong></th>
<th>Returns an enumeration constant that indicates the difference between the \texttt{MDStore} object referenced by the client application and the corresponding \texttt{MDStore} object on the Analysis server.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**clsCubeAnalyzer**

A Decision Support Objects (DSO) object of *ClassType clsCubeAnalyzer* contains a single method used to extract information from the query log. The query log stores the descriptions of queries executed on the Analysis server. This object provides a method through its own internal interface.

There are no collections or properties associated with an object of *ClassType clsCubeAnalyzer*.

**Examples**

**Retrieving the Cube Query Log**

The following code example retrieves the entire contents of a cube's query log from the Analysis server and prints the number of records in the immediate window:

Option Explicit

```vbnet
Public dsoServer As DSO.Server
Public dsoDB As DSO.MDStore
Public dsoCube As DSO.MDStore
Public dsoCubeAnalyzer As DSO.CubeAnalyzer
Public ADODBRecSet As ADODB.Recordset

Public Sub AnalyzeCube()
    If dsoServer Is Nothing Then
        Set dsoServer = New DSO.Server
        'MyServer is the name of the Analysis server.
        dsoServer.Connect ("MyServer")
    End If

    'Get first database from server.
```
Set dsoDB = dsoServer.MDStores(1)

'Get first cube from database.
Set dsoCube = dsoDB.MDStores(1)

'Get analyzer object from cube.
Set dsoCubeAnalyzer = dsoCube.Analyzer

'Get recordset from log.
Set ADODBRecSet = dsoCubeAnalyzer.OpenQueryLogRecordset _
("SELECT * FROM QueryLog")

If ADODBRecSet.BOF And ADODBRecSet.EOF Then
    Debug.Print "<<No records in query log>>"
Else
    ADODBRecSet.MoveLast
    Debug.Print " Record count: " & ADODBRecSet.RecordCount
End If
End Sub

See Also

Methods, clsCubeAnalyzer
Methods, clsCubeAnalyzer

An object of ClassType clsCubeAnalyzer implements the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenQueryLogRecordset</td>
<td>Opens a query log recordset</td>
</tr>
</tbody>
</table>

See Also

clsCubeAnalyzer
OpenQueryLogRecordset (clsCubeAnalyzer)

The OpenQueryLogRecordset method of an object of ClassType clsCubeAnalyzer returns a Microsoft® ActiveX® Data Objects (ADO) recordset containing a record for each analysis query run on the Analysis server that satisfies the given SQL query.

Syntax

Set ADODBRecSet = object.OpenQueryLogRecordset(SQLString As String)

ADODBRecSet

An ADODB recordset.

object

The object of ClassType clsCubeAnalyzer used.

SQLString

The SQL query that returns the query log recordset. You can create an SQL statement using any of the fields in the query log. For example:

"SELECT * FROM QueryLog WHERE Duration > 5"

The following columns are returned in ADODBRecSet. The ADO data types specified for each column can be found in the ADODB.DataTypeEnum enumeration. For more information about the ADO data types, see the ADO documentation.

<table>
<thead>
<tr>
<th>Column</th>
<th>ADO data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSOLAP_Database</td>
<td>adVarWChar</td>
<td>The name of the database used in the query</td>
</tr>
<tr>
<td>MSOLAP_Cube</td>
<td>adVarWChar</td>
<td>The name of the cube used in the query</td>
</tr>
<tr>
<td>MSOLAP_User</td>
<td>adVarWChar</td>
<td>The name of the user that ran the</td>
</tr>
<tr>
<td>Dataset</td>
<td>adVarWChar</td>
<td>A numeric string indicating the level from each dimension used to satisfy the query</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Slice</td>
<td>adVarWChar</td>
<td>A string indicating the data slice for the query.</td>
</tr>
<tr>
<td>StartTime</td>
<td>adDate</td>
<td>The time the query began</td>
</tr>
<tr>
<td>Duration</td>
<td>adInteger</td>
<td>The length of time (in seconds) of the query execution</td>
</tr>
<tr>
<td>MOLAPPartitions</td>
<td>adSmallInt</td>
<td>The number of different multidimensional OLAP (MOLAP) partitions used to satisfy the query</td>
</tr>
<tr>
<td>ROLAPPartitions</td>
<td>adSmallInt</td>
<td>The number of different relational OLAP (ROLAP) partitions used to satisfy the query</td>
</tr>
<tr>
<td>SamplingRate</td>
<td>adInteger</td>
<td>The sampling rate at the time the query was executed</td>
</tr>
</tbody>
</table>

**Remarks**

In order to create an instance of the ADODB Recordset object, you must add the Microsoft ActiveX Data Objects reference to your Microsoft Visual Basic® project.

The Dataset column is of particular interest for designing aggregations. The values in the Dataset column can be used when calling the AddGoalQuery method of the clsPartitionAnalyzer object to construct goal queries. Goal queries are used to fine-tune the process of aggregation design for a partition.

**See Also**

clsCubeAnalyzer

clsPartitionAnalyzer

Using Decision Support Objects
clsCubeCommand

An object of ClassType clsCubeCommand provides a specific implementation of the Decision Support Objects (DSO) Command interface. This object provides collections and properties through the Command interface. There are no methods associated with an object of ClassType clsCubeCommand.

Remarks

An object of ClassType clsCubeCommand encapsulates a user-defined command automatically executed on the Microsoft® SQL Server™ 2000 Analysis Services client when the cube containing the command is accessed. You add a command to a cube by adding it to the cube's Commands collection. Such commands include calculated members, named sets, library references, and others.

For more information, see Introducing Decision Support Objects.

Examples

Creating an Object of ClassType clsCubeCommand

Use the following code to create an object of ClassType clsCubeCommand:

'Assume an object (dsoServer) of ClassType clsServer exists 'with existing database and cube
Dim dsoDB As DSO.MDStore   'Database
Dim dsoCube As DSO.MDStore  'Cube
Dim dsoCmd As DSO.Command  'Command

Set dsoDB = dsoServer.MDStores(1)
Set dsoCube = dsoDB.MDStores(1)
Set dsoCmd = dsoCube.Commands.AddNew("CubeCmd1")
See Also

Collections, clsCubeCommand
Command Interface
Properties, clsCubeCommand
Analysis Services Programming

**Collections, clsCubeCommand**

An object of **ClassType clsCubeCommand** implements the following collection of the **Command** interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

**See Also**

- clsCubeCommand
- Command Interface
Properties, clsCubeCommand

An object of **ClassType clsCubeCommand** implements the following properties of the **Command** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>CommandType</strong></td>
<td>Returns an enumeration constant that identifies the specific command option</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of the cube command</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the <strong>Name</strong> and <strong>Statement</strong> properties are empty and that the command object belongs to a collection</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the cube command</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the command object in <strong>Commands</strong> collection of the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent object, using the <strong>MDStore</strong> interface of the parent object</td>
</tr>
<tr>
<td><strong>ParentObject</strong></td>
<td>Returns a reference to the parent object, using the default interface of the parent object</td>
</tr>
<tr>
<td><strong>Statement</strong></td>
<td>The text of the cube command statement, in Multidimensional Expressions (MDX)</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

**See Also**

- clsCubeCommand
- Command Interface
**clsCubeDimension**

An object of **ClassType clsCubeDimension** provides an implementation of the Decision Support Objects (DSO) **Dimension** interface that is specific to dimensions within a cube. This object provides collections and properties through the **Dimension** interface. There are no methods associated with an object of **ClassType clsCubeDimension**.

**Remarks**

The primary difference between a database dimension and a cube dimension is that in a cube dimension, certain properties that are inherited from the database dimension can be overridden by changing their values. For example, the **IsVisible** property can be overridden on a cube dimension, but the **StorageType** property cannot.

To define a cube dimension, you add a reference to a dimension that exists within a database to the **Dimensions** collection of the cube. A shared database dimension can be associated with multiple cube dimensions; a private database dimension can be associated with only one cube dimension. In both cases, the database dimension is automatically associated with the cube's partitions and aggregations, if there are any.

**Example**

Use the following code to create a **clsCubeDimension** object:

'Assume an object (dsoServer) of ClassType clsServer exists 'with an existing database and cube
Dim dsoDB As MDStore
Dim dsoCube As MDStore
Dim dsoCubeDim As DSO.Dimension
Set dsoDB = dsoServer.MDStores("FoodMart")
Set dsoCube = dsoDB.MDStores("Sales")
""Employees" is an existing database dimension
Set dsoCubeDim = dsoCube.Dimensions.AddNew("Employees")

See Also

Collections, clsCubeDimension
Dimension Interface
Properties, clsCubeDimension
Analysis Services Programming

Collections, clsCubeDimension

An object of `ClassType clsCubeDimension` implements the following collections of the `Dimension` interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CustomProperties</code></td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td><code>Levels</code></td>
<td>The collection of level objects associated with the cube dimension</td>
</tr>
</tbody>
</table>

See Also

- `clsCubeDimension`
- `Dimension Interface`
Properties, clsCubeDimension

An object of **ClassType clsCubeDimension** implements the following properties of the **Dimension** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AggregationUsage</strong></td>
<td>Specifies how aggregations are to be designed for the dimension.</td>
</tr>
<tr>
<td><strong>AllowSiblingsWithSameName</strong></td>
<td>Indicates whether a parent-child dimension can contain members with identical names.</td>
</tr>
<tr>
<td><strong>AreMemberKeysUnique</strong></td>
<td>Indicates whether member keys are unique for the dimension.</td>
</tr>
<tr>
<td><strong>AreMemberNamesUnique</strong></td>
<td>Indicates whether member names are unique for the dimension.</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type, which in this case is <strong>clsCubeDimension</strong>.</td>
</tr>
<tr>
<td><strong>DataMemberCaptionTemplate</strong></td>
<td>Contains a template string that is used to create captions for system-generated data members.</td>
</tr>
<tr>
<td><strong>DataSource</strong></td>
<td>A reference to the data source object used by the cube dimension.</td>
</tr>
<tr>
<td><strong>DefaultMember</strong></td>
<td>Defines the default member of the dimension.</td>
</tr>
<tr>
<td><strong>DependsOnDimension</strong></td>
<td>Names a dimension to which the current dimension is related.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of the cube dimension.</td>
</tr>
<tr>
<td><strong>DimensionType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of dimension.</td>
</tr>
<tr>
<td><strong>EnableRealTimeUpdates</strong></td>
<td>Indicates whether real-time updates are enabled for the dimension.</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>The SQL FROM clause for the cube dimension.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsChanging</td>
<td>Indicates whether members and/or levels are expected to change on a regular basis.</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>Indicates whether dimension writebacks are available to clients with appropriate permissions.</td>
</tr>
<tr>
<td>IsShared</td>
<td>Indicates whether the cube dimension is shared.</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>Indicates whether the cube dimension is temporary.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the structure of the cube dimension is valid.</td>
</tr>
<tr>
<td>IsVirtual</td>
<td>Indicates whether a dimension is virtual.</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the dimension is visible to the client.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>The SQL JOIN clause for a cube dimension.</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time when the cube dimension was last processed.</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>User-specified date. This is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td>MembersWithData</td>
<td>Determines which members in a dimension can have associated data in the fact table.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the cube dimension.</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the dimension object within the Dimensions collection of its parent object.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent MDStore object.</td>
</tr>
<tr>
<td>SourceTable</td>
<td>The name of the cube dimension's primary data source table.</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>Returns the alias of the source table for the dimension.</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>Restricts members included in a dimension.</td>
</tr>
<tr>
<td>StorageMode</td>
<td>Determines the method for storing</td>
</tr>
</tbody>
</table>
**SubClassType**

Returns an enumeration constant that identifies the subclass type of the object.

---

**See Also**

- *clsCubeDimension*
- *Dimension Interface*
clsCubeLevel

An object of ClassType **clsCubeLevel** provides a specific implementation of the Decision Support Objects (DSO) **Level** interface. This object provides collections and properties through the **Level** interface. There are no methods associated with an object of **ClassType clsCubeLevel**.

**Remarks**

When a dimension within a database is assigned to a cube, the cube inherits all levels of the dimension. An object of **ClassType clsCubeLevel** allows access to these levels. Because not all database dimensions necessarily apply to a given cube, one advantage to accessing the levels of a cube directly is that you avoid traversing the dimensions and levels of the entire database to determine which levels are used in a cube.

**Example**

Use the following code to create a dimension and levels for a database and apply them to a cube:

'Assume an object (dsoDB) of ClassType clsDatabase exists 'with an existing data source
Dim dsoDim As DSO.Dimension
Dim dsoLevel As DSO.Level
Dim dsoDS As DSO.Datasource

'Add a dimension and levels to the database
Set dsoDS = dsoDB.Datasources(1)
Set dsoDim = dsoDB.Dimensions.AddNew("Products")
Set dsoDim.DataSource = dsoDS 'Dimension DataSource
dsoDim.FromClause = "product" 'Source Table

'Add a Product Brand Name level
Set dsoLev = dsoDim.Levels.AddNew("Brand Name")
dsoLev.MemberKeyColumn = """"product"".""brand_name"""
dsoLev.ColumnSize = 255
dsoLev.ColumnType = adWChar
dsoLev.EstimatedSize = 100

'Add a Product Name level
Set dsoLev = dsoDim.Levels.AddNew("Product Name")
dsoLev.MemberKeyColumn = """"product"".""product_name"""
dsoLev.ColumnSize = 255
dsoLev.ColumnType = adWChar
dsoLev.EstimatedSize = 1560
dsoDim.Update
'Add additional dimensions and levels as required
...
'Add cube to database
Dim dsoCube As MDStore
Set dsoCube = dsoDB.MDStores.AddNew(strCubeName)

'Create and configure a DataSource object for the cube
Set dsoDS = dsoDB.DataSources(1)
dsoCube.DataSources.AddNew (dsoDS.Name)

'Set source fact table and estimated rows in fact table
dsoCube.SourceTable = """"sales_fact_1998"""
dsoCube.EstimatedRows = 1000

'Add shared database dimensions
'Cube inherits dimension levels
dsoCube.Dimensions.AddNew ("Products")
'Add other shared or private dimensions
See Also

Collections, clsCubeLevel
IsDisabled
Level Interface
Properties, clsCubeLevel
### Collections, clsCubeLevel

An object of `ClassType clsCubeLevel` implements the following collection of the `Level` interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CustomProperties</code></td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td><code>MemberProperties</code></td>
<td>The collection of objects of <code>ClassType clsMemberProperty</code></td>
</tr>
</tbody>
</table>

### See Also

- [clsCubeLevel](#)
- [Level Interface](#)
Properties, clsCubeLevel

An object of **ClassType clsCubeLevel** implements the following properties of the **Level** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreMemberKeysUnique</td>
<td>Indicates whether the members of a level are uniquely identified by their member key column within the level itself</td>
</tr>
<tr>
<td>AreMemberNamesUnique</td>
<td>Indicates whether the members of a level are uniquely identified by their member name column within the level itself</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant identifying the specific object type which, in this case, is <strong>clsCubeLevel</strong></td>
</tr>
<tr>
<td>ColumnSize</td>
<td>The size (in bytes) of the data in the <strong>MemberKeyColumn</strong> property of the level</td>
</tr>
<tr>
<td>ColumnType</td>
<td>The data type of the <strong>MemberKeyColumn</strong> property of the level</td>
</tr>
<tr>
<td>CustomRollUpColumn</td>
<td>Contains the name of the column that contains member-specific rollup instructions</td>
</tr>
<tr>
<td>CustomRollUpExpression</td>
<td>Contains a Multidimensional Expressions (MDX) expression used to override the default rollup mode</td>
</tr>
<tr>
<td>CustomRollUpPropertiesColumn</td>
<td>Contains the name of the column that supplies cell properties for member-specific rollup instructions</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the cube level</td>
</tr>
<tr>
<td>EnableAggregations</td>
<td>Specifies whether aggregations are to be enabled for the level object</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EstimatedSize</strong></td>
<td>The estimated number of rows of unique members in the level</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>Contains the SQL FROM clause for the cube level</td>
</tr>
<tr>
<td><strong>Grouping</strong></td>
<td>Indicates the type of grouping used by the Analysis server</td>
</tr>
<tr>
<td><strong>HideMemberIf</strong></td>
<td>Indicates whether a member should be hidden from client applications</td>
</tr>
<tr>
<td><strong>IsDisabled</strong></td>
<td>Indicates whether the cube level is disabled</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the structure of the cube level is valid</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>Indicates whether the level is visible to client applications</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>The SQL JOIN clause of the cube level</td>
</tr>
<tr>
<td><strong>LevelNamingTemplate</strong></td>
<td>Defines how levels in a parent-child hierarchy are named</td>
</tr>
<tr>
<td><strong>LevelType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of level</td>
</tr>
<tr>
<td><strong>MemberKeyColumn</strong></td>
<td>The name of the column that contains the member key of the cube level</td>
</tr>
<tr>
<td><strong>MemberNameColumn</strong></td>
<td>The name of the column that contains member names</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the cube level</td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>Specifies the method to use when ordering the members of a level</td>
</tr>
<tr>
<td><strong>OrderingMemberProperty</strong></td>
<td>Specifies a member property used to determine the ordering of members</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the level in the parent object's <strong>Levels</strong> collection</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>Dimension</strong> object</td>
</tr>
<tr>
<td><strong>ParentKeyColumn</strong></td>
<td>Identifies the parent of a member in a parent-child hierarchy</td>
</tr>
<tr>
<td><strong>RootMemberIf</strong></td>
<td>Determines how the root member or</td>
</tr>
</tbody>
</table>
members of a parent-child hierarchy are identified

**SkippedLevelsColumn**
Identifies the column that holds the number of empty levels between a member and its parent

**SubClassType**
Returns an enumeration constant that identifies the subclass type of the object

**UnaryOperatorColumn**
Contains the name of a column that stores mathematical operators serving as member-specific rollup instructions for the level

**See Also**

clsCubeLevel

Level Interface
**clsCubeMeasure**

An object of the **ClassType clsCubeMeasure** provides a specific implementation of the Decision Support Objects (DSO) **Measure** interface. This object provides collections and properties through the **Measure** interface. There are no methods associated with an object of **ClassType clsCubeMeasure**.

**Remarks**

A cube measure corresponds to a numeric column in a cube's fact table. When a cube is processed, its measures can be precalculated across its dimensions. For example, the number of items sold is a measure that can be precalculated across the dimensions of product, time, and geography.

For more information about the object model hierarchy, see **Object Architecture**.

**Example**

Use the following code to create an object of **ClassType clsCubeMeasure**:

```vba
'Assume an object (dsoCube) of ClassType clsCube exists
Dim dsoMeasure As DSO.Measure
Set dsoMeasure = dsoCube.Measures.AddNew("Salaries")
dsoMeasure.Description = "Employee salaries"
dsoMeasure.SourceColumn = ""''Employees''.'''Salaries''"
dsoMeasure.SourceColumnType = adInteger
dsoMeasure.AggregateFunction = aggSum
dsoMeasure.FormatString = ",,##0"
```

**See Also**

- [Working with Cubes](#)
- [Collections, clsCubeMeasure](#)
- [Measure Interface](#)
Properties, clsCubeMeasure
Collections, clsCubeMeasure

An object of ClassType clsCubeMeasure implements the following collection of the Measure interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

See Also

clsCubeMeasure

Measure Interface
Properties, clsCubeMeasure

An object of `ClassType clsCubeMeasure` implements the following properties of the `Measure` interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AggregateFunction</code></td>
<td>A value corresponding to the type of aggregate function used by the cube measure</td>
</tr>
<tr>
<td><code>ClassType</code></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><code>Description</code></td>
<td>The description of the cube measure</td>
</tr>
<tr>
<td><code>FormatString</code></td>
<td>The format used to display the values of the cube measure</td>
</tr>
<tr>
<td><code>IsValid</code></td>
<td>Indicates whether the measure structure is valid</td>
</tr>
<tr>
<td><code>IsVisible</code></td>
<td>Indicates whether the measure is visible to the client</td>
</tr>
<tr>
<td><code>Name</code></td>
<td>The name of the cube measure</td>
</tr>
<tr>
<td><code>OrdinalPosition</code></td>
<td>Returns the ordinal position of the measure in the parent object's <code>Measures</code> collection</td>
</tr>
<tr>
<td><code>Parent</code></td>
<td>Returns a reference to the parent cube object</td>
</tr>
<tr>
<td><code>SourceColumn</code></td>
<td>The name of the source column (in the fact table) for the cube measure</td>
</tr>
<tr>
<td><code>SourceColumnType</code></td>
<td>Returns a Microsoft® ActiveX® (ADO) DB enumeration constant that identifies the <code>SourceColumn</code> (in the fact table) data type</td>
</tr>
<tr>
<td><code>SubClassType</code></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

See Also

- `clsCubeMeasure`
- `Measure Interface`
Analysis Services Programming

**clsCubeRole**

An object of the `ClassType clsCubeRole` provides a specific implementation of the Decision Support Objects (DSO) Role interface. This object provides collections, methods, and properties through the Role interface.

**Remarks**

You use objects of `ClassType clsCubeRole` to manage the set of users who can access a cube (of any type) and the manner in which they can access it. A cube role has a name, a description, a parent object, a class type, a list of users, and a set of permissions. Each permission has a key and a corresponding permission expression.

You create roles at the database level (database roles) and then assign them to cubes (cube roles) by adding them to the collection of roles associated with the cube. The roles assigned to a cube automatically apply to its partitions and aggregations.

You can remove a database role by removing it from the database's collection of role objects. When you do so, the system automatically removes the corresponding cube roles from the cube's collection of role objects.

You can remove a cube role by removing it from the cube's collection of role objects. When you do so, the corresponding database role is not affected. However, the definition of the cube role remains in effect until you update or process the cube.

**Example**

Suppose you want to define roles named FinanceManagers, ProductionManagers, and SalesManagers at the database level. Suppose also that you want to assign the appropriate vice-president to the list of users for each role and the company president to the list of users for all three roles. Finally, suppose that you want to create a cube for each year's financial, production, and sales data for the years 1995, 1996, and 1997.
Use the following code to define the appropriate database and cube roles for this situation.

**Note** User lists defined for database roles are automatically associated with the corresponding cube roles and cannot be changed at the cube role level.

'Assume an object (dsoDB) of ClassType clsDatabase exists.
'Create database roles.
Dim DbRole_FinanceMgrs As DSO.Role
Dim DbRole_ProductionMgrs As DSO.Role
Dim DbRole_SalesMgrs As DSO.Role
Set DbRole_FinanceMgrs = dsoDB.Roles.AddNew("FinanceManagers")
Set DbRole_ProductionMgrs = dsoDB.Roles.AddNew("ProductionManagers")
Set SbRole_SalesMgrs = dsoDB.Roles.AddNew("SalesManagers")

'Define user lists for database roles.
'(In a real-world situation, actual user names would be used in place of titles like "President".)
DbRole_FinanceMgrs.UsersList = "President;VP_Finance"
DbRole_ProductionMgrs.UsersList = "President;VP_Production"
DbRole_SalesMgrs.UsersList = "President;VP_Sales"

'Update the repository for the database roles.
DbRole_FinanceMgrs.Update
DbRole_ProductionMgrs.Update
DbRole_SalesMgrs.Update

'Assume objects (Cube95, Cube96 and Cube97) of ClassType clsCube exist
'Create cube roles. Cube role names must be identical to the corresponding database role names.
Dim CubeRole_FinanceMgrs As DSO.Role
Dim CubeRole_ProductionMgrs As DSO.Role
Dim CubeRole_SalesMgrs As DSO.Role
'Add roles to Cube95.
Set CubeRole_FinanceMgrs = Cube95.Roles.AddNew("FinanceManagers")
Set CubeRole_ProductionMgrs = Cube95.Roles.AddNew("ProductionManagers")
Set CubeRole_SalesMgrs = Cube95.Roles.AddNew("SalesManagers")

'Add roles to Cube96.
Set CubeRole_FinanceMgrs = Cube96.Roles.AddNew("FinanceManagers")
Set CubeRole_ProductionMgrs = Cube96.Roles.AddNew("ProductionManagers")
Set CubeRole_SalesMgrs = Cube96.Roles.AddNew("SalesManagers")

'Add roles to Cube97.
Set CubeRole_FinanceMgrs = Cube97.Roles.AddNew("FinanceManagers")
Set CubeRole_ProductionMgrs = Cube97.Roles.AddNew("ProductionManagers")
Set CubeRole_SalesMgrs = Cube97.Roles.AddNew("SalesManagers")

'Update the repository for the cubes.
Cube95.Update
Cube96.Update
Cube97.Update

See Also

- clsDatabaseRole
- Collections, clsCubeRole
- Methods, clsCubeRole
- Properties, clsCubeRole
- Role Interface
- Security and Authentication
Collections, clsCubeRole

An object of ClassType clsCubeRole implements the following collection of the Role interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>The collection of commands for the role</td>
</tr>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

See Also

clsCubeRole
Role Interface
Methods, clsCubeRole

An object of ClassType clsCubeRole implements the following method of the Role interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPermissions</td>
<td>Sets the permissions for the cube role for a given key</td>
</tr>
</tbody>
</table>

See Also

clsCubeRole

Role Interface
Analysis Services Programming

Properties, clsCubeRole

An object of ClassType clsCubeRole implements the following properties of the Role interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the cube role</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the role structure is valid</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the cube role</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent object, using the MDStore interface of the parent object</td>
</tr>
<tr>
<td>ParentObject</td>
<td>Returns a reference to the parent object, using the default interface of the parent object</td>
</tr>
<tr>
<td>Permissions</td>
<td>The permissions for the cube role for a given key</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td>UsersList</td>
<td>A semicolon-separated list of users of the cube role</td>
</tr>
</tbody>
</table>

See Also

clsCubeRole

Role Interface
clsDatabase

An object of the ClassType clsDatabase provides a specific implementation of the Decision Support Objects (DSO) MDStore interface. This object provides collections, methods, and properties through the MDStore interface.

Example

Use the following code to create an object of ClassType clsDatabase:

' Assume an object (dsoServer) of ClassType clsServer exists
Dim dsoDatabase As DSO.MDStore ' Create an interface for the database
' Use the AddNew method of the server's MDStores collection to create
' the new database:
Set dsoDatabase = dsoServer.MDStores.AddNew("MyDatabase")

See Also

Collections, clsDatabase
Databases
MDStore Interface
Methods, clsDatabase
Properties, clsDatabase
**Collections, clsDatabase**

An object of `ClassType clsDatabase` implements the following collections of the `MDStore` interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commands</strong></td>
<td>The collection of command objects defined in the database</td>
</tr>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td><strong>DataSources</strong></td>
<td>The collection of data source objects used by the database</td>
</tr>
<tr>
<td><strong>Dimensions</strong></td>
<td>The collection of dimension objects defined in the database</td>
</tr>
<tr>
<td><strong>MDStores</strong></td>
<td>The collection of <code>MDStore</code> objects defined for the database</td>
</tr>
<tr>
<td><strong>MiningModels</strong></td>
<td>The collection of mining model objects defined for the database</td>
</tr>
<tr>
<td><strong>Roles</strong></td>
<td>The collection of role objects defined for the database</td>
</tr>
</tbody>
</table>

**See Also**

- clsDatabase
- MDStore Interface
Events, clsDatabase

An object of ClassType clsDatabase implements the following methods of the Database interface.

**IMPORTANT** In order to access these events, you must use the Database interface, not the MDStore interface.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportAfter</td>
<td>Called whenever a processing action on an object in the database has finished executing</td>
</tr>
<tr>
<td>ReportBefore</td>
<td>Called before a processing action on an object in the database</td>
</tr>
<tr>
<td>ReportError</td>
<td>Called whenever an error occurs during a processing action</td>
</tr>
<tr>
<td>ReportProgress</td>
<td>Called to report the progress of an action during processing</td>
</tr>
</tbody>
</table>

See Also

clsDatabase
Methods, clsDatabase

An object of `ClassType clsDatabase` implements the following methods of the `MDStore` interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BeginTrans</strong></td>
<td>Begins the transaction on the database</td>
</tr>
<tr>
<td><strong>Clone</strong></td>
<td>Copies the property values and (optionally) the collections of major and minor objects from one database object to another</td>
</tr>
<tr>
<td><strong>CommitTrans</strong></td>
<td>Commits the transaction on the database</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>The <strong>LockObject</strong> method of the <strong>Database</strong> interface locks an object to prevent multiple users from concurrently changing the object</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Processes the database</td>
</tr>
<tr>
<td><strong>Rollback</strong></td>
<td>Rolls back the transaction on the database</td>
</tr>
<tr>
<td><strong>UnlockObject</strong></td>
<td>Releases a lock previously established by the <strong>LockObject</strong> method</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td>Updates the database definition in the meta data repository</td>
</tr>
</tbody>
</table>

See Also

- clsDatabase
- MDStore Interface
Properties, clsDatabase

An object of **ClassType clsDatabase** implements the following properties of the **MDStore** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregationPrefix</td>
<td>The common prefix that can be used for aggregation names for all of the partitions in a database</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the database</td>
</tr>
<tr>
<td>EstimatedSize</td>
<td>The estimated size of the database</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>Indicates the read/write access status of the database</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the structure of the database is valid</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the database is visible to other client applications</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time when the database was last processed</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>A user-specified date. Not used by Microsoft® SQL Server™ 2000 Analysis Services</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the database</td>
</tr>
<tr>
<td>OlapMode</td>
<td>Returns an enumeration constant that identifies the type of OLAP storage mode</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent server object</td>
</tr>
<tr>
<td>Server</td>
<td>Returns a reference to the <strong>DSO.Server</strong> object</td>
</tr>
<tr>
<td>State</td>
<td>Returns an enumeration constant that indicates the difference between the database object referenced by the client application and corresponding database on the Analysis server</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type</td>
</tr>
</tbody>
</table>
See Also

clsDatabase

MDStore Interface
An object of **ClassType clsDatabaseCommand** provides a specific implementation of the Decision Support Objects (DSO) **Command** interface. This object provides collections, methods, and properties through the **Command** interface.

**Remarks**

An object of **ClassType clsDatabaseCommand** encapsulates a user-defined command that is automatically executed on the Microsoft® SQL Server™ 2000 Analysis Services client when the database containing the command is accessed. You add a command to a database by adding it to the database's **Commands** collection. Such commands include calculated members, named sets, library references, and others.

**Example**

Use the following code to create an object of **ClassType clsDatabaseCommand**:

```vba
'Assume an object (dsoServer) of ClassType clsServer exists 'with an existing database
Dim dsoDB As DSO.MDStore    'Database
Dim dsoCmd As DSO.Command   'Command

Set dsoDB = dsoServer.MDStores(1)
Set dsoCmd = dsoDB.Commands.AddNew("DBCmd1")
```

**See Also**

- [Collections, clsDatabaseCommand](#)
- [Command Interface](#)
- [Commands](#)
Methods, clsDatabaseCommand
Properties, clsDatabaseCommand
Collections, clsDatabaseCommand

An object of ClassType clsDatabaseCommand implements the following collection of the Command interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

See Also

Command Interface
Methods, clsDatabaseCommand

An object of ClassType clsDatabaseCommand implements the following methods of the Command interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks an object</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks a previously locked object</td>
</tr>
<tr>
<td>Update</td>
<td>Saves the definition of the command object in the metadata repository</td>
</tr>
</tbody>
</table>

See Also

Command Interface
Properties, clsDatabaseCommand

An object of **ClassType clsDatabaseCommand** implements the following properties of the **Command** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific class type</td>
</tr>
<tr>
<td><strong>CommandType</strong></td>
<td>Returns an enumeration constant that identifies the command option</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of the database command</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the structure of the <strong>Command</strong> object is valid</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the database command</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the command object in the <strong>Commands</strong> collection of the parent <strong>MDStore</strong> object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent object, using the <strong>MDStore</strong> interface of the parent object</td>
</tr>
<tr>
<td><strong>ParentObject</strong></td>
<td>Returns a reference to the parent object, using the default interface of the parent object</td>
</tr>
<tr>
<td><strong>Statement</strong></td>
<td>The text of the database command statement, in Multidimensional Expressions (MDX)</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

**See Also**

**Command Interface**
clsDatabaseDimension

An object of ClassType clsDatabaseDimension provides a specific implementation of the Decision Support Objects (DSO) Dimension interface. This object provides collections, methods, and properties through the Dimension interface.

Remarks

Database dimensions can be shared or private. A shared database dimension can be associated with any number of cubes, but a private database dimension can be associated with only a single cube. When a database dimension is associated with a cube, it is automatically associated with the cube's partitions and aggregations, if there are any.

All dimensions, shared and private, are created in a database object and stored in the database object's Dimensions collection. Private dimensions are identified by incorporating the names of the cubes to which they are private into the names of the dimensions. For example, a dimension named Cube1^PrivateDimension is private to Cube1 because its name begins with Cube1 followed by the caret (^) character.

Example

Use the following code to create an object of ClassType clsDatabaseDimension:

' Assume an object (dsoDB) of ClassType clsDatabase exists
Dim dsoDim As DSO.Dimension
Set dsoDim = dsoDB.Dimensions.AddNew("MyDim")

See Also

Collections, clsDatabaseDimension

Dimension Interface
Dimensions

Methods, clsDatabaseDimension
Properties, clsDatabaseDimension
Collections, clsDatabaseDimension

An object of ClassType clsDatabaseDimension implements the following collections of the Dimension interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>Levels</td>
<td>The collection of objects of level objects associated with the database dimension</td>
</tr>
</tbody>
</table>

See Also

Dimension Interface
Methods, clsDatabaseDimension

An object of ClassType clsDatabaseDimension implements the following methods of the Dimension interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks the database dimension</td>
</tr>
<tr>
<td>Process</td>
<td>Processes the database dimension</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks the previously locked database dimension</td>
</tr>
<tr>
<td>Update</td>
<td>Updates the definition of the database dimension in the metadata repository</td>
</tr>
</tbody>
</table>

See Also

Dimension Interface
Properties, clsDatabaseDimension

An object of **ClassType clsDatabaseDimension** implements the following properties of the **Dimension** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AllowSiblingsWithSameName</strong></td>
<td>Indicates whether a parent-child dimension can contain members with identical names.</td>
</tr>
<tr>
<td><strong>AreMemberKeysUnique</strong></td>
<td>Indicates whether member keys are unique within the dimension.</td>
</tr>
<tr>
<td><strong>AreMemberNamesUnique</strong></td>
<td>Indicates whether member names are unique within the dimension.</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
</tr>
<tr>
<td><strong>DataMemberCaptionTemplate</strong></td>
<td>Contains a template string that is used to create captions for system-generated data members.</td>
</tr>
<tr>
<td><strong>DataSource</strong></td>
<td>The name of the object of <strong>ClassType clsDataSource</strong> used by the database dimension.</td>
</tr>
<tr>
<td><strong>DefaultMember</strong></td>
<td>Defines the default member of the dimension.</td>
</tr>
<tr>
<td><strong>DependsOnDimension</strong></td>
<td>Names a dimension to which the current dimension is related.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of the database dimension.</td>
</tr>
<tr>
<td><strong>DimensionType</strong></td>
<td>Returns an enumeration constant identifying the specific type of dimension.</td>
</tr>
<tr>
<td><strong>EnableRealTimeUpdates</strong></td>
<td>Indicates whether real-time updates are enabled for the dimension.</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>The SQL FROM clause for a database dimension.</td>
</tr>
<tr>
<td><strong>IsChanging</strong></td>
<td>Indicates whether members and/or levels are expected to change on a regular basis.</td>
</tr>
<tr>
<td><strong>IsReadWrite</strong></td>
<td>Indicates whether end users that have appropriate permissions can write back to dimensions.</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IsShared</strong></td>
<td>Indicates whether the database dimension is shared.</td>
</tr>
<tr>
<td><strong>IsTemporary</strong></td>
<td>Indicates whether the database dimension should be permanently stored in the repository.</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the structure of the database dimension is valid.</td>
</tr>
<tr>
<td><strong>IsVirtual</strong></td>
<td>Indicates whether the database dimension is virtual.</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>The SQL JOIN clause for the dimension.</td>
</tr>
<tr>
<td><strong>LastProcessed</strong></td>
<td>The date and time when the database dimension was last processed.</td>
</tr>
<tr>
<td><strong>LastUpdated</strong></td>
<td>A user-specified date. This property is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td><strong>MembersWithData</strong></td>
<td>Determines which members in a dimension can have associated data in the fact table.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the database dimension.</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the dimension object within its parent object's <strong>Dimensions</strong> collection.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object.</td>
</tr>
<tr>
<td><strong>SourceTable</strong></td>
<td>The name of the primary table of the database dimension.</td>
</tr>
<tr>
<td><strong>SourceTableAlias</strong></td>
<td>Returns the alias of the source table for the database dimension.</td>
</tr>
<tr>
<td><strong>SourceTableFilter</strong></td>
<td>Restricts members that are included in a dimension.</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Indicates the difference between the dimension object referenced by the client application and the corresponding</td>
</tr>
</tbody>
</table>
dimension on the Analysis server.

<table>
<thead>
<tr>
<th><strong>StorageMode</strong></th>
<th>Determines how the contents of a cube's dimensions are stored.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object.</td>
</tr>
</tbody>
</table>

**See Also**

[Dimension Interface]
Analysis Services Programming

**clsDatabaseLevel**

An object of `ClassType clsDatabaseLevel` provides a specific implementation of the Decision Support Objects (DSO) `Level` interface. This object provides collections and properties through the `Level` interface. There are no methods associated with an object of `ClassType clsDatabaseLevel`.

**Remarks**

When you add a [dimension](#) to a [cube](#), the cube inherits whatever [levels](#) you defined for the database dimension; that is, the database levels you defined become cube levels, as well. Similarly, database levels are automatically inherited by the [partitions](#) and [aggregations](#) you add to a cube.

**Example**

Use the following code to create an object of `ClassType clsDatabaseLevel`:

```vbnet
' Assume the existence of an object (myDim) of ClassType clsDimensi
Dim myLev As DSO.Level
Set myLev = myDim.Levels.AddNew('Brand Name')
```

**See Also**

- [Collections, clsDatabaseLevel](#)
- [Level Interface](#)
- [Properties, clsDatabaseLevel](#)
Collections, clsDatabaseLevel

An object of ClassType clsDatabaseLevel implements the following collection of the Level interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>MemberProperties</td>
<td>The collection of objects of ClassType clsMemberProperty</td>
</tr>
</tbody>
</table>

See Also

clsDatabaseLevel

Level Interface
Properties, clsDatabaseLevel

An object of ClassType clsDatabaseLevel implements the following properties of the Level interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreMemberKeysUnique</td>
<td>Indicates whether the members of a level are uniquely identified by their member key column</td>
</tr>
<tr>
<td>AreMemberNamesUnique</td>
<td>Indicates whether the members of a level are uniquely identified by their member name column</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td>ColumnSize</td>
<td>The size (in bytes) of the data in the member key column of the level</td>
</tr>
<tr>
<td>ColumnType</td>
<td>The data type of the member key column of the level</td>
</tr>
<tr>
<td>CustomRollUpColumn</td>
<td>Contains the name of the column that contains member-specific rollup instructions</td>
</tr>
<tr>
<td>CustomRollUpExpression</td>
<td>Contains a Multidimensional Expressions (MDX) expression used to override the default rollup mode</td>
</tr>
<tr>
<td>CustomRollUpPropertiesColumn</td>
<td>Contains the name of the column that supplies cell properties for member-specific rollup instructions</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the database level</td>
</tr>
<tr>
<td>EstimatedSize</td>
<td>The estimated number of rows in the database level</td>
</tr>
<tr>
<td>FromClause</td>
<td>The SQL FROM clause for the database level</td>
</tr>
<tr>
<td>Grouping</td>
<td>Indicates the type of grouping used by the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>HideMemberIf</strong></td>
<td>Indicates whether a member should be hidden from client applications</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the structure of the database level is valid</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>Indicates whether the level is visible to client applications</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>The SQL JOIN clause for the database level</td>
</tr>
<tr>
<td><strong>LevelNamingTemplate</strong></td>
<td>Defines how levels in a parent-child hierarchy are named</td>
</tr>
<tr>
<td><strong>LevelType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of level</td>
</tr>
<tr>
<td><strong>MemberKeyColumn</strong></td>
<td>The name of the column that contains the member keys of the database level</td>
</tr>
<tr>
<td><strong>MemberNameColumn</strong></td>
<td>The name of the column that contains member names</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the database level</td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>Specifies the method to use when ordering the members of a level</td>
</tr>
<tr>
<td><strong>OrderingMemberProperty</strong></td>
<td>Specifies a member property used to determine the ordering of members</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the level in the <strong>Levels</strong> collection of the parent object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>Dimension</strong> object</td>
</tr>
<tr>
<td><strong>ParentKeyColumn</strong></td>
<td>Identifies the parent of a member in a parent-child hierarchy</td>
</tr>
<tr>
<td><strong>RootMemberIf</strong></td>
<td>Determines how the root member or members of a parent-child hierarchy are identified</td>
</tr>
<tr>
<td><strong>SkippedLevelsColumn</strong></td>
<td>Identifies the column that holds the number of empty levels between a member and its parent</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>UnaryOperatorColumn</strong></td>
<td>Contains the name of a column that stores mathematical operators serving as member-specific rollup instructions for the level</td>
</tr>
</tbody>
</table>

**See Also**

clsDatabaseLevel

Level Interface
Analysis Services Programming

**clsDatabaseRole**

An object of **ClassType clsDatabaseRole** provides a specific implementation of the Decision Support Objects (DSO) **Role** interface. This object provides collections, methods, and properties through the **Role** interface.

**Remarks**

Objects of **ClassType clsDatabaseRole** are used to manage the set of users who can access a database and the manner in which they can access it. As with cube roles and mining model roles, a database role has a name, a description, a parent object, a class type, a list of users, and a set of permissions. Each permission has a key and a corresponding permission expression.

Unlike cube roles and mining model roles, however, a database role serves as the basis for cube and mining model roles. To create roles for cubes and data mining models, create roles at the database level (database roles) and then assign them to cubes or mining models by adding them to the collection of roles associated with the cube or mining model. The act of assigning database roles to cubes or mining models creates corresponding cube roles and mining model roles based on the database role. The roles assigned to a cube apply automatically to its partitions and aggregations.

You can remove a database role by removing it from the database's collection of role objects. When you do so, the system automatically removes the corresponding cube and mining model roles from the cube's collection of role objects.

**Example**

Suppose you define roles named FinanceManagers, ProductionManagers, and SalesManagers at the database level. Suppose also that you assign the appropriate vice president to the list of users for each role and the company president to the list of users for all three roles.

Use the following code to define the appropriate database and cube roles for this situation. After the code has been executed, you can assign the database roles to
cubes (or virtual cubes) as needed. For more information, see clsCubeRole.

' Assume an object (dsoDB) of ClassType clsDatabase exists.
' Create database roles.
Dim DbRole_FinanceMgrs As DSO.Role
Dim DbRole_ProductionMgrs As DSO.Role
Dim DbRole_SalesMgrs As DSO.Role
Set DbRole_FinanceMgrs = dsoDB.Roles.AddNew("FinanceManagers")
Set DbRole_ProductionMgrs = dsoDB.Roles.AddNew("ProductionManagers")
Set DbRole_SalesMgrs = dsoDB.Roles.AddNew("SalesManagers")

' Define user lists for database roles.
' (In a real-world situation, actual user names would be
' used in place of titles like "President").
DbRole_FinanceMgrs.UsersList = "President;VP_Finance"
DbRole_ProductionMgrs.UsersList = "President;VP_Production"
DbRole_SalesMgrs.UsersList = "President;VP_Sales"

' Update the repository for the database roles.
DbRole_FinanceMgrs.Update
DbRole_ProductionMgrs.Update
DbRole_SalesMgrs.Update

' Assume objects (Cube95, Cube96 and Cube97) of ClassType clsCube
' Create cube roles. Cube role names must be identical
' to the corresponding database role names.
Dim CubeRole_FinanceMgrs As DSO.Role
Dim CubeRole_ProductionMgrs As DSO.Role
Dim CubeRole_SalesMgrs As DSO.Role

' Add roles to Cube95.
Set CubeRole_FinanceMgrs = Cube95.Roles.AddNew("FinanceManagers")
Set CubeRole_ProductionMgrs = Cube95.Roles.AddNew("ProductionManagers")
Set CubeRole_SalesMgrs = Cube95.Roles.AddNew("SalesManagers")
' Add Roles to Cube96.
Set CubeRole_FinanceMgrs = Cube96.Roles.AddNew("FinanceManagers")
Set CubeRole_ProductionMgrs = Cube96.Roles.AddNew("ProductionManagers")
Set CubeRole_SalesMgrs = Cube96.Roles.AddNew("SalesManagers")

' Add Roles to Cube97.
Set CubeRole_FinanceMgrs = Cube97.Roles.AddNew("FinanceManagers")
Set CubeRole_ProductionMgrs = Cube97.Roles.AddNew("ProductionManagers")
Set CubeRole_SalesMgrs = Cube97.Roles.AddNew("SalesManagers")

' Update the repository for the cubes.
Cube95.Update
Cube96.Update
Cube97.Update

**See Also**

Collections, clsDatabaseRole
Methods, clsDatabaseRole
Properties, clsDatabaseRole
Role Interface
Analysis Services Programming

**Collections, clsDatabaseRole**

An object of **ClassType clsDatabaseRole** implements the following collections of the **Role** interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commands</strong></td>
<td>The collection of commands for the role</td>
</tr>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

**See Also**

- [clsDatabaseRole](#)
- [Role Interface](#)
Analysis Services Programming

**Methods, clsDatabaseRole**

An object of `ClassType clsDatabaseRole` implements the following methods of the `Role` interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies the properties of the role to a different role object</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks the role object</td>
</tr>
<tr>
<td>SetPermissions</td>
<td>Sets role permissions for a given key</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks a previously locked object</td>
</tr>
<tr>
<td>Update</td>
<td>Updates the definition of the database role in the meta data repository</td>
</tr>
</tbody>
</table>

**See Also**

- [clsDatabaseRole](#)
- [Role Interface](#)
Properties, clsDatabaseRole

An object of **ClassType clsDatabaseRole** implements the following properties of the **Role** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The description of a database role</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the role structure is valid</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of a database role</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent object, using the <strong>MDStore</strong> interface of the parent object</td>
</tr>
<tr>
<td><strong>ParentObject</strong></td>
<td>Returns a reference to the parent object, using the default interface of the parent object</td>
</tr>
<tr>
<td><strong>Permissions</strong></td>
<td>The permissions for the database role for a given key</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td><strong>UsersList</strong></td>
<td>A semicolon-delimited list of users of the database role</td>
</tr>
</tbody>
</table>

**See Also**

- [clsDatabaseRole](#)
- [Role Interface](#)
clsDataSource

An object of **ClassType clsDataSource** specifies an external database that will be used as a source of data for an object of **ClassType clsDatabase**, **clsCube**, or **clsPartition**. The object of **ClassType clsDataSource** provides collections, methods, and properties though its own internal interface.

Remarks

Connections to data sources are initiated when Decision Support Objects (DSO) requires access to data or property information in the source database. Data sources are only connected to when needed or when explicitly requested by the program. Executing the **IsConnected** method of an object of **ClassType clsDataSource** causes the Analysis server to attempt to connect to the specified data source.

An object of **ClassType clsDatabase** may contain multiple objects of **ClassType clsDataSource** in its **DataSources** collection. Objects of **ClassType clsCube** and **clsPartition** can only contain a single object of **ClassType clsDataSource** in their respective **DataSources** collection. An aggregation object (**ClassType clsAggregation**) does not implement the **DataSources** collection of the **MDStore** interface.

Examples

A. Creating a New Database

The following example demonstrates how to connect to the Analysis server and create a new database, attach a data source, and add a shared dimension and level. It uses the sample FoodMart 2000 database. After building and running the example code, you should be able to view the new database using Analysis Manager.

Option Explicit
Public dsoServer As DSO.Server
Const strConnect = "Provider=MSDASQL.1;Persist Security Info=False;Data Source=FoodMart 2000;Connect Timeout=15"

'Note: Add command control to form to enable ' the cmdCreateDatabase_Click method

Private Sub cmdCreateDatabase_Click()
    On Error GoTo CreateDatabase_Err

    Dim dsoDB As DSO.MDStore
    Dim dsoDS As DSO.Datasource

    'Create database and add connection string
    Set dsoDB = dsoServer.MDStores.AddNew("MyDatabase")
    Set dsoDS = dsoDB.Datasources.AddNew("NewSales")
    dsoDS.ConnectionString = strConnect
    dsoDS.Update

    'Create dimension and set data source
    Dim dsoDim As DSO.Dimension
    Set dsoDim = dsoDB.Dimensions.AddNew("Products")
    Set dsoDim.Datasource = dsoDS
    dsoDim.FromClause = "product"
    dsoDim.JoinClause = ""

    'Add levels
    Dim dsoLev As DSO.Level
    Set dsoLev = dsoDim.Levels.AddNew("Product Id")
    'Point to table and column
    dsoLev.MemberKeyColumn = """"product_class"""".""""product_family"
    dsoLev.ColumnSize = 4        'Width of column in bytes
    dsoLev.ColumnType = adInteger  'ADODB Data Type

    dsoDim.Update
Debug.Print "<<success>>"

Exit Sub

CreateDatabase_Err:
    Debug.Print "Error creating new database"
    Debug.Print Err.Description
    Err.Clear
End Sub

Private Sub Form_Load()
    On Error GoTo FormLoad_Err

    'Connect to the Analysis server
    Set dsoServer = New DSO.Server
    'MyServer is the name of the Analysis server
    dsoServer.Connect ("MyServer")
    Debug.Print ("Connected")
    Exit Sub

FormLoad_Err:
    Debug.Print ("Error connecting to server")
    Debug.Print Err.Description
    Err.Clear
End Sub

B. Connecting to Data Source Providers

Connection string examples are also provided for the following data source providers:

Microsoft® OLE DB Provider for Jet 3.51 OLE DB:
ConnectionString="Provider=Microsoft.Jet.OLEDB.3.51;" & _
"Persist Security Info=False;" & _
"Data Source=C:\Program Files" & _
"Microsoft Analysis Services\ Samples\ FoodMart 2000.mdb"

Microsoft OLE DB Provider for Jet 4.0:

ConnectionString="Provider=Microsoft.Jet.OLEDB.4.0;" & _
"Persist Security Info=False;" & _
"Data Source=C:\Program Files" & _
"Microsoft Analysis Services\Samples\FoodMart 2000.mdb;" & _
"JET OLEDB:SFP=True;"

Microsoft OLE DB Provider for ODBC (Microsoft Access):

ConnectionString="Provider=MSDASQL.1;" & _
"Persist Security Info=False;" & _
"Data Source=FoodMart 2000;" & _
"Connect Timeout=15"

Microsoft SQL Server™:

ConnectionString = "Provider=SQLOLEDB.1;" & _
"Persist Security Info=False;" & _
"User ID=sa;" & _
"Initial Catalog=FoodMart 2000;" & _
"Data Source={SQL Server};" & _
"Connect Timeout=15"

See Also

.clsDatabase
.clsCube
.clsPartition
.Collections, clsDataSource
Methods, clsDataSource
Properties, clsDataSource
Collections, clsDataSource

An object of ClassType clsDataSource implements the following collection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

Access
Read/write

See Also

clsDataSource
Methods, clsDataSource

An object of ClassType clsDataSource implements the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies the properties and collections of a data source object to another data source object.</td>
</tr>
<tr>
<td>IsConnected</td>
<td>Connects to a data source. The method returns True if the data source is connected, and False if it is not connected and is unable to connect.</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks a data source object.</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Unlocks a previously locked data source object.</td>
</tr>
<tr>
<td>Update</td>
<td>Updates the data source object definition in the meta data repository.</td>
</tr>
</tbody>
</table>

See Also

clsDataSource
Clone (clsDataSource)

The **Clone** method of an object of **ClassType clsDataSource** copies the property values and the collections of a data source object to a target data source object.

**Syntax**

```
object.Clone(ByVal TargetObject As ICommon, Optional ByVal Options As CloneOptions = cloneMajorChildren)
```

*object*

The object of **ClassType clsDataSource** to be copied.

*TargetObject*

An existing object of **ClassType clsDataSource**.

*Options*

One of the values in the **CloneOptions** enumeration. If no value is specified, the **cloneMajorChildren** option is used. Because objects of **ClassType clsDataSource** contain no major or minor objects, the entire object is cloned regardless of the option specified. For more information, see [CloneOptions](#).

**See Also**

[clsDataSource](#)
IsConnected (clsDataSource)

The IsConnected method of an object of ClassType clsDataSource verifies the connection to the data source specified by the ConnectionString property. If the data source is not connected, the method attempts to connect to the source.

Syntax

\[ bRet = \text{object.IsConnected}(\text{ErrorMsg}\text{ As String}) \]

\( bRet \)

A Boolean variable that receives the returned value: True is returned if the data source is connected, False if the connection cannot be established.

\( \text{object} \)

An object of ClassType clsDataSource.

\( \text{ErrorMsg} \)

An optional string variable that receives the error definition if the connection cannot be established.

Remarks

If the data source is already connected when the method is executed, the method returns True. If the data source is not connected, the method attempts to connect to the data source, returning True if the connection is established or False if the connection cannot be established.

It is not necessary to establish a connection to a data source before it can be used. Decision Support Objects (DSO) will automatically establish the connection when necessary (for example, to read a property value from the source database or to access data in the database).

Example

Use the following code to establish a connection to a data source with the
**IsConnected** property:

'Assume an object (dsoDS) of ClassType clsDataSource exists
If Not dsoDS.IsConnected Then
    ' Code to handle connection error
Else
    ' Connection is established
End If

**See Also**

[clsDataSource](#)

[ConnectionString](#)
Analysis Services Programming

LockObject (clsDataSource)

The LockObject method of an object of ClassType clsDataSource locks a data source object to prevent actions of multiple users of the object from colliding.

Syntax

object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String)

object

The data source object to lock.

LockType

One of the lock types defined in the OlapLockTypes enumeration. For more information, see OlapLockTypes.

LockDescription

A string containing the description of the lock, available to other applications attempting to obtain a lock.

Remarks

It is sometimes possible for an application to request an additional lock on an already locked object. For example, other applications can request and receive an olapLockRead lock on an object already locked using the olapLockProcess lock.

See Also

clsDataSource
UnlockObject (clsDataSource)

The **UnlockObject** method of an object of **ClassType clsDataSource** releases a lock on a data source object previously established by the **LockObject** method.

**Syntax**

```csharp
object.UnlockObject
```

*object*

The data source object from which to remove a lock.

**Remarks**

For a complete discussion of object locking, see **LockObject**.

**See Also**

**clsDataSource**
Update (clsDataSource)

The **Update** method of an object of **ClassType clsDataSource** updates the definition of a data source object in the metadata repository.

**Syntax**

```
object.Update
```

*object*

An object of **ClassType clsDataSource**.

**See Also**

[clsDataSource](#)
## Properties, clsDataSource

An object of **ClassType clsDataSource** implements the following properties. The table also shows whether the property is read/write (R/W) or read-only (R).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific class type</td>
<td>R</td>
</tr>
<tr>
<td><strong>CloseQuoteChar</strong></td>
<td>The right (closing) quote character used by the source database</td>
<td>R</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>A reference to a Microsoft® ActiveX® Data Objects (ADO) <strong>Connection</strong> object, used to connect to a relational database</td>
<td>R</td>
</tr>
<tr>
<td><strong>ConnectionString</strong></td>
<td>A string containing the initialization parameters for the source database</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A description of the data source</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>IsReadOnly</strong></td>
<td>Indicates whether the data source is read-only</td>
<td>R</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the structure of the data source object is valid</td>
<td>R</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the data source object</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>OpenQuoteChar</strong></td>
<td>The left (opening) quote character used by the source database</td>
<td>R</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
<td>R</td>
</tr>
<tr>
<td><strong>SupportedTxnDDL</strong></td>
<td>Returns the value of the connection object's <strong>Transaction DDL</strong> property,</td>
<td>R</td>
</tr>
</tbody>
</table>
which indicates the source database's ability to support data definition language (DDL) statements in transactions

**See Also**

[clsDataSource](#)
ClassType (clsDataSource)

The `ClassType` property of an object of `ClassType clsDataSource` object returns an enumeration constant that identifies the specific class type.

Data Type

Integer representing a value from the `ClassTypes` enumeration. For more information, see `ClassTypes`.

Access

Read-only

Remarks

Most objects in Decision Support Objects (DSO) have a `ClassType` and a `SubClassType` property. The `SubClassType` property uses an enumerated value to provide additional information about the object. This property supports the `clsDataSource` value from the `ClassTypes` enumeration.

See Also

- clsDataSource
- SubClassType
**CloseQuoteChar (clsDataSource)**

The *CloseQuoteChar* property of an object of *ClassType clsDataSource* returns the right (closing) quote character used by the source database.

**Data Type**

Variant

**Access**

Read-only

**Remarks**

To properly qualify a table or column name that contains white space, a data source may require the name be delimited or enclosed using a quote character. The quote character is generally specific to the data source or data source driver.

**Example**

The following code specifies a *FromClause* in a dimension and uses the proper delimiter characters for the data source:

'Assume an object (dsoDimension) of
'ClassType clsDatabaseDimension exists and
'get the quoting characters from the data source
Dim sLQuote As String, sRQuote As String
sLQuote = dsoDimension.Datasource.OpenQuoteChar
sRQuote = dsoDimension.Datasource.CloseQuoteChar

'Set the comma separated list of the dimension tables
dsoDimension.FromClause = sLQuote & "store" & sRQuote

**See Also**
clsDataSource

FromClause
Connection (clsDataSource)

The Connection property of an object of ClassType clsDataSource returns a reference to a Microsoft® ActiveX® Data Objects (ADO) Connection object, used to connect to a relational database.

Data Type
ADODB.Connection

Access
Read-only

Remarks
You can use this ADO Connection object to access the source database directly. For more information, see the ADO documentation.

See Also
clsDataSource
**ConnectionString (clsDataSource)**

The `ConnectionString` property of an object of `ClassType clsDataSource` returns a string containing the OLE DB initialization parameters for the source database.

**Data Type**
String

**Access**
Read/write

**Remarks**
For more information about valid connection string parameters and format, see the OLE DB documentation or the source database documentation.

**Example**
Use the following code to set the `ConnectionString` property for a `clsDataSource` object.

'Assume an object (dsoDatasource) of ClassType clsDataSource exists
'Set the OleDB connection string.
'The connection string is used to establish the connection
'to the relational database that contains the dimension and
'fact tables. We will use OleDB provider for ODBC drivers
dsoDatasource.ConnectionString = _
   "Provider=MSDASQL.1;Data Source=FoodMart;Connect Timeout=

'Save the datasource definition in the meta data repository
dsoDatasource.Update
See Also

clsDataSource
Description (clsDataSource)

The **Description** property of an object of *ClassType clsDataSource* sets or returns the description of the data source.

**Data Type**

String

**Access**

Read/write

**Remarks**

You can use this property to provide a description of the data source, for example:

dsoDS.Description = "1997 Sales Data verified 2/1/1998"

**See Also**

[clsDataSource](#)
IsReadOnly (clsDataSource)

The IsReadOnly property of an object of ClassType clsDataSource identifies whether the source database is read-only.

Data Type

Boolean

Access

Read-only

Remarks

This property returns True if the source database is read-only or False if the source database is read/write.

See Also

clsDataSource
IsValid (clsDataSource)

The IsValid property of an object of ClassType clsDataSource identifies whether the structure of an object is valid.

Data Type
Boolean

Access
Read-only

Remarks
Validity checking consists of verifying that the object's Name and Parent properties are not empty and that the IsConnected property is True. If all properties are valid, the IsValid property returns True. If any of the properties are invalid, the IsValid property returns False.

See Also

clsDataSource
ConnectionString
Name
Parent
Name (clsDataSource)

The Name property of an object of ClassType clsDataSource sets or returns the name of the object.

Data Type
String

Access
Read/write (read-only after the object has been named)

Remarks
The Name property contains the valid name of the Decision Support Objects (DSO) object when it was created. Typically, an object cannot be renamed once a value has been supplied for the Name property.

See Also
clsDataSource
OpenQuoteChar (clsDataSource)

The `OpenQuoteChar` property of an object of `ClassType clsDataSource` contains the left (opening) quote character used by the source database.

**Data Type**

Variant

**Access**

Read-only

**Remarks**

To properly qualify a table or column name that contains white space, a data source may require the name be delimited or enclosed using a quote character. The quote character is generally specific to the data source or data source driver.

**Example**

The following code specifies a `FromClause` for a dimension and uses the proper delimiter characters for the data source:

'Assume an object (dsoDimension) of 'ClassType clsDatabaseDimension exists and 'get the quoting characters from the data source
Dim sLQuote As String, sRQuote As String
sLQuote = dsoDimension.Datasource.OpenQuoteChar
sRQuote = dsoDimension.Datasource.CloseQuoteChar

'Set the comma-separated list of the dimension tables
dsoDimension.FromClause = sLQuote & "store" & sRQuote
See Also

clsDataSource

FromClause
Parent (clsDataSource)

The Parent property of an object of ClassType clsDataSource contains a reference to the parent MDStore object that contains the DataSource object. For more information about MDStore objects, see MDStore Interface.

Data Type

MDStore

Access

Read-write

Remarks

You can only set this property to an object of ClassType clsDatabase. Attempting to set this property to an object of any other class type returns an error.

See Also

clsDataSource
SubClassType (clsDataSource)

The **SubClassType** property of an object of **ClassType clsDataSource** contains an enumeration constant identifying the subclass type of the object.

**Data Type**

SubClassTypes

**Access**

Read-only

**Remarks**

For objects of **ClassType clsDataSource**, the value of **SubClassType** is always sbclsRegular. For more information about the **SubClassTypes** enumeration, see Enumerations.

**See Also**

clsDataSource

SubClassTypes
Analysis Services Programming

**SupportedTxnDDL (clsDataSource)**

The **SupportedTxnDDL** property of an object of **ClassType clsDataSource** returns the value of the connection object's **Transaction DDL** property, which indicates the source database's ability to support data definition language (DDL) statements in transactions.

**Data Type**

Long

**Access**

Read-only

**Remarks**

The meaning of the value returned is specific to the database provider. For more information, see the Microsoft® ActiveX® Data Objects (ADO) documentation and the OLE DB documentation.

**See Also**

[clsDataSource](#)
**clsMemberProperty**

The member property object defines a property for a level member. Like level members, these properties are read from the dimension table. A level can have any number of member properties. An object of ClassType clsMemberProperty provides collections and properties through its own internal interface. There are no methods associated with this object class. clsMemberProperty objects are contained in a parent level object's MemberProperties collection.

**Remarks**

Access to the properties of an object of ClassType clsMemberProperty depends on the context in which it is used. clsMemberProperty objects are created and managed in the context of a database level and have read/write access. Cube and partition levels inherit member properties from the database level. Member properties accessed through cube and partition level objects are read-only.

Member properties are versatile objects that can be used to facilitate a number of different tasks. One task, for example, is that of sorting the members of a level by a particular attribute. For example, consider the States level of the Geography dimension. A member property can be defined that refers to the population of the state. Client applications can then sort on this population property.

**See Also**

- clsAggregationLevel
- clsDatabaseLevel
- clsCubeLevel
- clsPartitionLevel
- Collections, clsMemberProperty
- Properties, clsMemberProperty
Collections, clsMemberProperty

An object of `ClassType clsMemberProperty` implements the following collection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined Property objects</td>
</tr>
</tbody>
</table>

Access

Read/write

See Also

- clsMemberProperty
- Property Object
Properties, clsMemberProperty

An object of **ClassType clsMemberProperty** implements the following properties. The table also shows whether the property is read/write (R/W) or read-only (R).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption</td>
<td>The name of the column that contains the member property in the members and axis schema rowsets</td>
<td>R/W</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
<td>R</td>
</tr>
<tr>
<td>ColumnSize</td>
<td>The size (in bytes) of the data stored in the column referenced by the <strong>SourceColumn</strong> property</td>
<td>R/W</td>
</tr>
<tr>
<td>ColumnType</td>
<td>The data type of the source column on which the member property is based</td>
<td>R/W</td>
</tr>
<tr>
<td>Description</td>
<td>A description of the property</td>
<td>R/W</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the member property is visible to client applications</td>
<td>R/W</td>
</tr>
<tr>
<td>Language</td>
<td>Identifies the language used</td>
<td>R/W</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the member property</td>
<td>R/W (R after the object has been named)</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the <strong>clsMemberProperty</strong> object in the <strong>MemberProperties</strong> collection</td>
<td>R</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent <strong>Level</strong> object</td>
<td>R</td>
</tr>
<tr>
<td>PropertyType</td>
<td>Categorizes the content of information provided by the member property</td>
<td>R/W</td>
</tr>
<tr>
<td>SourceColumn</td>
<td>The dimension table name and column that contains values for the member</td>
<td>R/W</td>
</tr>
<tr>
<td>property</td>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

**See Also**

- [clsMemberProperty](#)
- [MemberProperties](#)
Caption (clsMemberProperty)

The Caption property of an object of ClassType clsMemberProperty contains the name of the column that contains the member property in the members and axis schema rowsets. This is useful for creating language-specific versions of member properties.

Data Type
String

Access
Read/write

Remarks
Individual member properties can have identical Caption values only if they have different values for Language. When a client application requests the contents of a member property, the Analysis server compares the locale ID of the client application to the Language property. The member property with the most appropriate Language value will be sent to the client application.

Two values for the Caption property are noteworthy: MEMBER_CAPTION and DESCRIPTION. The MEMBER_CAPTION schema column should be used to define member captions. Under normal circumstances, a client application should use this schema column for text when displaying members. Similarly, the DESCRIPTION column should be used to define textual member descriptions. By combining these Caption values with specific Language values, the administrator can define localized member captions and descriptions that will be used automatically by any client application.

The following schema column names are defined by OLE DB for OLAP and should not be used as values for the Name or Caption properties of member properties.

- CATALOG_NAME
- SCHEMA_NAME
- CUBE_NAME
- DIMENSION_UNIQUE_NAME
- HIERARCHY_UNIQUE_NAME
- LEVEL_UNIQUE_NAME
- LEVEL_NUMBER
- MEMBER_ORDINAL
- MEMBER_NAME
- MEMBER_UNIQUE_NAME
- MEMBER_TYPE
- MEMBER_GUID
- CHILDREN_CARDINALITY
- PARENT_LEVEL
- PARENT_UNIQUE_NAME
• PARENT_COUNT

See Also

clsMemberProperty
**ClassType (clsMemberProperty)**

The **ClassType** property of an object of **ClassType clsMemberProperty** contains an enumeration constant identifying the specific class type of the **clsMemberProperty** object.

**Data Type**

**ClassTypes**

**Access**

Read-only

**See Also**

**clsMemberProperty**
ColumnSize (clsMemberProperty)

The ColumnSize property of a clsMemberProperty object identifies the size (in bytes) of the data stored in the column referenced by the SourceColumn property.

Data Type

Integer

Access

Read/write

See Also

clsMemberProperty
SourceColumn (clsMemberProperty)
ColumnType (clsMemberProperty)

The ColumnType property of a clsMemberProperty object identifies the data type of the source column, specified in the SourceColumn property, on which the member property is based.

Data Type

Integer representing a constant from the ADODB.DataTypeEnum enumeration.

Access

Read-write

Remarks

For more information about using the ADODB.DataTypeEnum enumeration, see the Microsoft® ActiveX® Data Objects (ADO) documentation.

See Also

clsMemberProperty
SourceColumn (clsMemberProperty)
Description (clsMemberProperty)

The **Description** property of an object of **ClassType clsMemberProperty** contains a description of the object.

**Data Type**

String

**Access**

Read/write

**See Also**

[clsMemberProperty]
IsVisible (clsMemberProperty)

The **IsVisible** property of an object of **ClassType clsMemberProperty** indicates whether the member property is visible to client applications.

**Data Type**
Boolean

**Access**
Read-write

**Remarks**
Member properties that are not visible are not listed in schema rowsets, but they are still accessible by name through Multidimensional Expressions (MDX) expressions.

**See Also**

[clsMemberProperty](#)
Language (clsMemberProperty)

The **Language** property of an object of **ClassType clsMemberProperty** object identifies the client language for the object.

**Data Type**

LanguageValues

**Access**

Read/write

**Remarks**

When multiple member properties have the same **Caption**, the OLAP server returns the one whose **Language** property best matches the locale ID of the client application. If no match is available, the server returns the member property with a **Language** value of **languageAny**. For more information about the **LanguageValues** enumeration, see the Microsoft® Visual Basic® documentation.

**See Also**

clsMemberProperty
Analysis Services Programming

**Name (clsMemberProperty)**

The **Name** property of an object of **ClassType clsMemberProperty** contains the name of the object.

**Data Type**

String

**Access**

Read/write (read-only after the object has been named)

**Remarks**

Decision Support Objects (DSO) uses some member property objects internally to manage each cube it creates. Setting the **Name** property to one of the following reserved member property names raises an error:

- CUSTOM_ROLLUP
- KEY
- NAME
- PARENT
- SKIPPED_LEVELS
- ID
- UNARY_OPERATOR
In addition, setting the **Name** property to the name of a schema rowset column also raises an error. For more information about schema rowset column names, see [Schema Rowsets](#).

**See Also**

[clsMemberProperty](#)
OrdinalPosition (clsMemberProperty)

The **OrdinalPosition** property of an object of **ClassType clsMemberProperty** contains the ordinal position of a **MemberProperty** object within its parent object's collection.

**Data Type**

Integer

**Access**

Read-only

**See Also**

[clsMemberProperty](#)
Parent (clsMemberProperty)

The **Parent** property of an object of `ClassType clsMemberProperty` contains a reference to the **Level** object to which the `clsMemberProperty` object belongs.

**Data Type**

[Level]

**Access**

Read-only

**See Also**

[clsMemberProperty]
PropertyType (clsMemberProperty)

The PropertyType property of an object of ClassType clsMemberProperty categorizes the content of information provided by the member property.

Data Type

PropertyTypeValue

Access

Read/write

Remarks

This property is passed to the client application, which then determines how to interpret the data provided by the member property. This allows client applications to create custom functions to process the data stored in member properties. For example, if you define a custom member property called E-Mail Address for members of the Customer dimension, you can set this property to propWebMailAlias. This would enable a client application to automatically display the member with a mailto URL link whenever the other properties of the member were displayed.

See Also

clsMemberProperty
SourceColumn (clsMemberProperty)

The **SourceColumn** property of an object of **ClassType clsMemberProperty** contains a reference to the column in the dimension table that contains values for the member property.

**Data Type**

String

**Access**

Read/write

**Examples**

**Specifying the SourceColumn Property for a Member Property Object**

Use the following code to specify the **SourceColumn** for a new object of **ClassType clsMemberProperty**:

'Assume an object (dsoLevel) of ClassType clsDimensionLevel exists.
' Create a member property containing the name of the store manager.
Dim dsoMemProp As DSO.MemberProperty
Set dsoMemProp = dsoLevel.MemberProperties.AddNew("Store Mana")
'Set the column which contains the names of the managers.
dsoMemProp.SourceColumn = """"store"""".""store_manager""""
SubClassType (clsMemberProperty)

The SubClassType property of an object of ClassType clsMemberProperty contains an enumeration constant identifying the subclass type of the object.

Data Type

SubClassTypes

Access

Read-only

Remarks

A member property object's SubClassType property can have a value of sbclsRegular only.

See Also

clsMemberProperty
clsMiningModel

Objects of ClassType clsMiningModel contain the definitions for data mining models that are contained in the MiningModels collection of a clsServer object. Each model contains a collection, called the Columns collection, of data mining columns (that is, objects of ClassType clsColumn) that correspond to the case table definition for the model. Each of these data mining columns can, in turn, contain its own collection of data mining columns in the Columns collection. Such columns are referred to as nested columns. In addition to the Columns collection, the mining model object also contains references to Roles, DataSources and CustomProperties collections, which are used in the same manner as their counterparts under the cube object.

Remarks

After you create a mining model object by invoking the AddNew method of the server object's MiningModels collection, define the structure of the mining model. The most important step in this process is to determine the model's subclass type by setting this property to either sbclsOLAP or sbclsRelational. Data mining models whose SubClassType is sbclsOLAP (that is, OLAP data mining models) are based on an OLAP cube. Data mining models whose SubClassType is sbclsRelational (that is, relational data mining models) are based on a table from a relational database. The choice of the model's SubClassType determines how the rest of the model's structure is defined. After that, you can determine other elements of the model design, such as the data mining algorithm that the model will use and the roles that will be associated with the model.

OLAP Data Mining Models

To establish the case set for an OLAP data mining model, set the CaseDimension property of the clsMiningModel object to a dimension within a cube; this automatically defines the case level that provides case key columns for the mining model as the last enabled and visible level in the selected dimension. Then set the individual columns in the Columns collection of the
**Relational Data Mining Models**

To establish the case set for a relational data mining model, select the key columns from a table or view in a relational database and then add column objects to the *Columns* collection that refer to these key columns. Next, add columns that refer to other columns in the table to the *Columns* collection, to supply input and predictable information to the data mining model.

**Examples**

**A. Creating an OLAP Mining Model Manually**

The following example builds a mining model based on the Sales cube in the *FoodMart 2000* sample database:

```vbscript
Public Sub CreateOlapMiningModel_1()
    '------------------------------------------------------------------------
    ' Declarations - Identify all of the variables that will be needed to
    ' create the data mining model.
    '------------------------------------------------------------------------
    Dim dsoSvr As New DSO.Server ' Server object
    Dim dsoDmm As DSO.MiningModel ' Note that because events are
    ' this object is being invoked directly instead of through an MDStore
    Dim dsoColumn As DSO.Column
    Dim dsoRole As DSO.Role
    Dim dsoNestedCol As DSO.Column
    Dim dsoCb As DSO.MDStore
    Dim dsoDim As DSO.Dimension
    Dim dsoLvl As DSO.Level

    '------------------------------------------------------------------------
    ' Connect to the server and walk through the schema for the cube that the
    ' data mining model will be based on. Save the references to the
```
'subordinate objects that will be needed later in this example.

' Connect to the server on this computer.

dsoSvr.Connect "LocalHost"

'Select the FoodMart database.
Set dsoDb = dsoSvr.MDStores("Foodmart 2000")

'Select the Sales cube.
Set dsoCb = dsoDb.MDStores("Sales")

'Select the Customers dimensions.
Set dsoDim = dsoCb.Dimensions("Customers")

'Select the Name level of the Customers dimension.
Set dsoLvl = dsoDim.Levels("Name")

' Before the model is created, check for a previous incarnation of it.
' If it exists, delete it. Then create a new one.
' Give the new model a new data source, and give it a role.
' Then describe the model for browsing the schema, and declare the
' algorithm that will be used to predict with.
' Finally, set up the OLAP properties that will be needed by the model.

'Check for the existence of the model on this computer.
If Not dsoDb.MiningModels("CustSalesModel") Is Nothing Then
    'If this model exists, delete it.
    dsoDb.MiningModels.Remove "CustSalesModel"
End If

'Create a new mining model called CustSalesModel.
Set dsoDmm = dsoDb.MiningModels.AddNew("CustSalesModel", { }

'Designate FoodMart 2000 as the data source for this mining model.
dsoDmm.DataSources.AddNew "Foodmart 2000"
'Create a new mining model role called All Users.
Set dsoRole = dsoDmm.Roles.AddNew("All Users")

'Describe this new mining model.
dsoDmm.Description = "Analyzes the purchasing behavior of customers"
'use the Decision Trees algorithm in this model.
dsoDmm.MiningAlgorithm = "Microsoft_Decision_Trees"
'Declare that the Sales cube will be used as the source for this model.
dsoDmm.SourceCube = "Sales"
'Declare that the case dimension will be based on the Customers dimension from the Sales cube.
dsoDmm.CaseDimension = "Customers"
'Use the Name level of the Customers dimension for cases.
dsoDmm.CaseLevel = "Name"
'Let DSO figure out the training query by leaving this property blank.
dsoDmm.TrainingQuery = ""

'------------------------------------------------------------------------
' Add a new column, Customer Id, to the mining model and relate this column to the Name level of the Customers dimension. Describe the level's type and make it a key for the model.
'------------------------------------------------------------------------

'Add Customer Id as a new column in the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Customer Id")
'Identify the level in Sales that this column is based on.
Set dsoColumn.SourceOlapObject = dsoLvl
'Identify the type of column this is.
dsoColumn.DataType = adInteger ' This enumeration is from ADO.
'Identify this column as a key.
dsoColumn.IsKey = True

'------------------------------------------------------------------------
' Add a new column to the mining model called Gender and relate this
' column to the Gender member property of the Name level of the 
' Customers dimension. Declare that the data in this column is 
' statistically discrete.

'Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Gender")
'Identify the member property of the Customers dimension 
'that this column is based on.
'Set the column's description for browsers of the schema.
dsoColumn.Description = "Based on the Gender member property " 
  "of the Name level of the Customers dimension."
Set dsoColumn.SourceOlapObject = dsoLvl.MemberProperties("Gender")
'Identify its type.
dsoColumn.DataType = adWChar
'Make this column related to the Customer Id column.
dsoColumn.RelatedColumn = "Customer Id"
'Identify this column as one containing discrete data.
dsoColumn.ContentType = "DISCRETE"

' Add a new column to the mining model called Unit Sales and relate 
' this column to the Sales cube measure of the same name. Set the 
' columns data type to Integer, and identify the data content in it as 
' being continuous and logarithmically normalized. Finally, identify thi 
' column as being predictable.

'Add another column to the model.
Set dsoColumn = dsoDmm.Columns.AddNew("Unit Sales")
'Identify this column as being based on the Unit Sales measure.
Set dsoColumn.SourceOlapObject = dsoCb.Measures("Unit Sales")
'Identify the column type.
dsoColumn.DataType = adInteger
'Identify this column's content as being continuous.
dsoColumn.ContentType = "CONTINUOUS"
'Identify the statistical distribution of this data.
dsoColumn.Distribution = "LOG_NORMAL"
'Identify the column as being predictable.
dsoColumn.IsPredictable = True

'------------------------------------------------------------------------

' Save the mining model and update its LastUpdated property.
'------------------------------------------------------------------------

'Set the date of last update to today's date.
dsoDmm.LastUpdated = Now
'Save the model definition.
dsoDmm.Update

'------------------------------------------------------------------------

' Lock the cube, process it, and then unlock it.
' Note: During processing a number of events will be fired. These even
' are trapped by the database object's ReportAfter, Report Before,
' ReportProgress, and ReportError events.
'------------------------------------------------------------------------

'Because the model is about to be processed, it must be locked.
dsoDmm.LockObject olapLockProcess, "Processing the data mining
'Fully process the model.
dsoDmm.Process processFull
'Unlock the model after processing is complete.
dsoDmm.UnlockObject
End Sub

**B. Creating an OLAP Mining Model Automatically**

The following example automatically creates and OLAP mining model based on
the **Sales** cube in the **FoodMart2000** database:

Public Sub CreateOlapMiningModel_2()
'Declarations - Identify all of the variables that will be needed to create the data mining model.

Dim dsoSvr As New DSO.Server
Dim dsoDmm As DSO.MiningModel
Dim dsoColumn As DSO.Column
Dim dsoRole As DSO.Role
Dim dsoNestedCol As DSO.Column

Before the model is created, check for a previous incarnation of it. If it exists, delete it. Then create a new one. Give the new model a new data source, and give it a role. Then describe the model for browsing of the schema, and declare the algorithm that will be used to predict with. Finally, set up the OLAP properties that the model will need.

dsoSvr.Connect "LocalHost"
Set dsoDb = dsoSvr.MDStores("Foodmart 2000")

If Not dsoDb.MiningModels("CustSales_Olap2") Is Nothing Then
dsoDb.MiningModels.Remove "CustSales_Olap2"
End If

Set dsoDmm = dsoDb.MiningModels.AddNew("CustSales_Olap2",

'Create a new mining model role called All Users.
Set dsoRole = dsoDmm.Roles.AddNew("All Users")

dsoDmm>Description = "Analyzes the purchasing behavior of customer
dsoDmm>MiningAlgorithm = "Microsoft_Decision_Trees"
dsoDmm.SourceCube = "Sales"
dsoDmm.CaseDimension = "Customers"
dsoDmm.TrainingQuery = "" 'Let DSO figure out the training query.

'------------------------------------------------------------------------
'
In this next step, the update method checks to see whether there are any columns in the columns collection. In this case, because there are not any, the update method will automatically add columns based on the structure of the Sales cube.

'------------------------------------------------------------------------

dsoDmm.Update 'Let DSO automatically populate the Columns collection

'Enable the Products dimension.
'Set dsoColumn = dsoDmm.Columns("Products")
'dsoColumn.IsDisabled = False

'Make the Unit Sales measure predictable.
Set dsoColumn = dsoDmm.Columns("Unit Sales")
'Enable the column.
dsoColumn.IsDisabled = False
'Make the column predictable.
dsoColumn.IsPredictable = True

' Set the last updated date to today's date.
dsoDmm.LastUpdated = Now
'Save the model's meta data.
dsoDmm.Update

'------------------------------------------------------------------------
'
Lock the cube, process it, and then unlock it.
'Note: During processing a number of events will be fired. These even are trapped by the database object's ReportAfter, ReportBefore, ReportProgress, and ReportError events.

'------------------------------------------------------------------------
'Because the model is about to be processed, lock it.
  dsoDmm.LockObject olapLockProcess, "Processing the data mining
'Process the model.
  dsoDmm.Process processFull
'Unlock the model.
  dsoDmm.UnlockObject
End Sub

See Also

AddNew
clsColumn
Collections, clsMiningModel

An object of ClassType clsMiningModel supports the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns</strong></td>
<td>The collection of Column objects that represent the structure of the mining model. Each column may contain a nested collection of columns. For more information, see Data Mining Model Structure.</td>
</tr>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties for the mining model.</td>
</tr>
<tr>
<td><strong>DataSources</strong></td>
<td>The collection of data source objects used by the mining model.</td>
</tr>
<tr>
<td><strong>Roles</strong></td>
<td>The collection of role objects defined for the mining model.</td>
</tr>
</tbody>
</table>

See Also

clsMiningModel
Methods, clsMiningModel

An object of **ClassType clsMiningModel** supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone</td>
<td>Copies an existing object to a target object of the same class type. It also creates a copy of the property values and provides the option of creating collections of major and minor objects.</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks the mining model.</td>
</tr>
<tr>
<td>Process</td>
<td>Creates and trains the mining model on the server.</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Releases a lock previously established by the LockObject method.</td>
</tr>
<tr>
<td>Update</td>
<td>Saves and updates the mining model's meta data.</td>
</tr>
<tr>
<td>ValidateStructure</td>
<td>Validates the properties and structure of a mining model object including the Columns collection. If it finds an invalid structure it raises an error with an appropriate message.</td>
</tr>
</tbody>
</table>

See Also

- clsMiningModel
Clone (clsMiningModel)

The **Clone** method of an object of **ClassType clsMiningModel** copies the properties and levels of an existing object to a target object of the same class type.

**Syntax**

```vba
object.Clone(ByVal TargetObject As MiningModel, [ByVal Options As CloneOptions = cloneMajorChildren])
```

- **object**
  - The mining model object whose properties are to be copied.
- **TargetObject**
  - A previously created object of the same class type.
- **Options**
  - One of the values of the **CloneOptions** enumeration. If no value is specified, the **cloneMajorChildren** option is used. For more information, see **CloneOptions**.

**Remarks**

The **Clone** method, depending on the clone option specified in **Options**, copies properties and objects to a new object with the same **ClassType** property value.

**Example**

The following example copies the properties of **dsoDMMSource** to **dsoDMMTarget**:

```vba
'Assume an object (dsoDB) of ClassType clsDatabase exists.
Dim dsoDMMSource As DSO.MiningModel
Set dsoDMMSource = dsoDB.MiningModels("Source")
```
'Create target mining model and clone just the properties.
Dim dsoDMMTarget As DSO.MiningModel
Set dsoDMMTarget = dsoDB.MiningModels.AddNew("Target")
dsoDMMSource.Clone dsoDMMTarget, cloneObjectProperties

See Also

clsMiningModel
LockObject (clsMiningModel)

The **LockObject** method of an object of **ClassType clsMiningModel** locks a mining model to prevent multiple users from concurrently changing the object.

**Syntax**

```csharp
object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String)
```

- **object**
  - The object to lock.

- **LockType**
  - One of the constants of the **OlapLockTypes** enumeration. For more information, see **OlapLockTypes**.

- **LockDescription**
  - A string containing the description of the lock, available to other applications attempting to obtain a lock.

**Remarks**

It is sometimes possible for an application to request an additional lock on an already locked object. For example, other applications can request and receive an **olapLockRead** lock on an object already locked using the **olapLockProcess** lock.

**Example**

The following example updates an existing mining model and saves it. It then locks the model with an informational message and processes the model. After processing is complete, the model is unlocked.

```csharp
dsoDmm.LastUpdated = Now
dsoDmm.Update
```
dsoDmm.LockObject olapLockProcess, "Processing the mining model
dsoDmm.Process processFull
dsoDmm.UnlockObject

See Also

clsMiningModel
LockObject
Analysis Services Programming

**Process (clsMiningModel)**

The **Process** method of an object of **ClassType clsMiningModel** creates and trains a mining model on the Analysis server.

**Syntax**

```csharp
object.Process([ByVal Options As ProcessTypes])
```

*object*

The mining model object to process.

*Options*

One of the constants in the **ProcessTypes** enumeration. For more information, see **ProcessTypes**.

The following **ProcessTypes** values are valid for processing a mining model.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processFull</td>
<td>Creates, updates, and trains the mining model on the Analysis server</td>
</tr>
<tr>
<td>processRefreshData</td>
<td>Retrains a mining model on the Analysis server</td>
</tr>
</tbody>
</table>

**Remarks**

If you set a value for the **TrainingQuery** property, the value is used to train the mining model. If you do not set a value for **TrainingQuery**, the SHAPE query used to train the mining model is generated from the **Columns** collection of the model.

**Example**

The following example updates an existing mining model and saves it. It then locks the model with an informational message and processes the model. After processing the model is complete it unlocks the model.
dsoDmm.LastUpdated = Now
dsoDmm.Update
dsoDmm.LockObject olapLockProcess, "Processing the mining model
dsoDmm.Process processFull
dsoDmm.UnlockObject

See Also

clsColumn
clsMiningModel
ProcessTypes
TrainingQuery
UnlockObject (clsMiningModel)

The UnlockObject method of an object of ClassType clsMiningModel releases a lock on a mining model object previously established by the LockObject method.

Syntax

object.UnlockObject

object

The mining model object to unlock.

Remarks

If the UnlockObject method is called without first calling the LockObject method, an error is raised.

See Also

clsMiningModel
Update (clsMiningModel)

The **Update** method of an object of **ClassType clsMiningModel** saves the mining model along with its **Columns** collection to the repository.

**Syntax**

```
object.Update

object
```

The mining model object to update.

**Remarks**

For mining models of **SubClassType sbclsOlap**, the **Update** method checks to see whether the **Columns** collection is empty. If it is, the method automatically populates the **Columns** collection based on the structure of the source cube before saving to the repository.

By default, only the **Column** object that corresponds to the **CaseLevel** property of the mining model is enabled; the **CaseLevel** is the same as the level object from the **SourceCube** of the mining model, and it provides the cases for the model. Users can then select and enable other columns by setting the **IsDisabled** property of the **Column** objects to False.

**Examples**

**Creating an OLAP Mining Model**

The following example creates an OLAP mining model without explicitly assigning any columns to the model. The **Update** method then automatically builds the structure of the Columns collection based upon the source cube's architecture and sets their **IsDisabled** properties to True. The example then enables some of the columns and makes the **UnitSales** column predictable.

```
Public Sub CreateOlapMiningModel_2()
```
' Declare all of the variables that will be needed to create the data mining model.

Dim dsoSvr As New DSO.Server
Dim dsoDmm As DSO.MiningModel
Dim dsoColumn As DSO.Column
Dim dsoRole As DSO.Role
Dim dsoNestedCol As DSO.Column

' Before the model is created, check for a previous incarnation of it.
' If it exists, delete it. Then create a new one.
' Give the new model a new data source, and give it a role.
' Then describe the model for browsing of the schema, and declare the algorithm that will be used to predict with.
' Lastly, set up the OLAP properties that will be needed by the model.

dsoSvr.Connect "LocalHost"
Set dsoDb = dsoSvr.MDStores("Foodmart 2000")

If Not dsoDb.MiningModels("CustSales_Olap2") Is Nothing Then
    dsoDb.MiningModels.Remove "CustSales_Olap2"
End If

Set dsoDmm = dsoDb.MiningModels.AddNew("CustSales_Olap2",

' Create a new mining model role called All Users.
Set dsoRole = dsoDmm.Roles.AddNew("All Users")
dsoDmm.Description = "Analyzes the purchasing behavior of customers"
dsoDmm.MiningAlgorithm = "Microsoft_Decision_Trees"
dsoDmm.SourceCube = "Sales"
dsoDmm.CaseDimension = "Customers"
dsoDmm.CaseLevel = "Name"
dsoDmm.TrainingQuery = "" 'Let DSO figure out the training query.

' In the next step, the Update method checks to see whether there are any columns in the columns collection. In this case, because there aren't any, the update method will automatically add columns based on the structure of the Sales cube.

' Let DSO automatically populate the Columns collection.
dsoDmm.Update 'Let DSO automatically populate the Columns collection.

'Enable the Products dimension.
'Set dsoColumn = dsoDmm.Columns("Products")
'dsoColumn.IsDisabled = False

'Make the Unit Sales measure predictable.
Set dsoColumn = dsoDmm.Columns("Unit Sales")
'Enable the column.
dsoColumn.IsDisabled = False
'Make the column predictable.
dsoColumn.IsPredictable = True

' Set the last updated date to today's date.
dsoDmm.LastUpdated = Now
' Save the model's meta data.
dsoDmm.Update

' Lock the cube, process it, and then unlock it.
' Note: During processing a number of events will be fired. These even
' are trapped by the database object's ReportAfter, Report Before, ReportProgress, and ReportError events.

'Because the model is about to be processed, it must be locked.
dsoDmm.LockObject olapLockProcess, "Processing the data mining model" 
'Process the model.
dsoDmm.Process processFull 
'Unlock the model.
dsoDmm.UnlockObject

End Sub

See Also

CaseLevel
clsColumn
clsMiningModel
IsDisabled
Level Interface
SourceCube
ValidateStructure (clsMiningModel)

The ValidateStructure method of an object of ClassType clsMiningModel validates the structure of the object, raising an error if an invalid structure element is encountered.

Syntax

object.ValidateStructure

object

The mining model object whose structure is to be validated.

Remarks

The ValidateStructure method ensures that the following requirements are met for all data mining models:

- The MiningAlgorithm property contains the name of a valid data mining algorithm.

- At least one column exists in the Columns collection. A column is an object with a ClassType of clsColumn.

- At least one column in the Columns collection must be enabled.

- All columns in the Columns collection must be valid.

For clsMiningModel objects with a SubClassType of sbclsRelational, the following additional requirement must be met:

- The FromClause property must not be empty.

For clsMiningModel objects with a SubClassType of sbclsOlap, the following additional requirements must be met:
- The **SourceCube** property must contain the name of a valid cube in the same database as the OLAP mining model.

- The cube named in the **SourceCube** property must be visible and cannot contain data mining dimensions.

- The **CaseDimension** property must contain the name of a valid dimension in the same database as the OLAP mining model.

- The dimension named in the **CaseDimension** property must be visible and cannot be a virtual dimension created by an earlier version of Microsoft® SQL Server™ 2000 Analysis Services.

**Example**

The following example validates the OLAP data mining model Customer Pattern Discovery:

```vbs
' Assume an object (dsoDB) of ClassType clsDatabase exists.
Dim dsoDMM As DSO.MiningModel
Set dsoDMM = dsoDB.MiningModels("Customer Pattern Discovery

' Validate the data mining model.
On Error Resume Next
dsoDMM.ValidateStructure
If Err.Number <> 0 Then MsgBox "An error occurred while" & _
     " validating the mining model:" & vbCrLf & _
     Err.Description

See Also

[clsMiningModel](#)
Properties, clsMiningModel

An object of ClassType clsMiningModel supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreKeysUnique</td>
<td>Indicates whether key columns defined in the Columns collection uniquely identify members in the case table.</td>
<td>R/W**</td>
</tr>
<tr>
<td>CaseDimension</td>
<td>Identifies the dimension that contains cases for the mining model.</td>
<td>R/W*</td>
</tr>
<tr>
<td>CaseLevel</td>
<td>Identifies the level of the CaseDimension that contains the cases for the mining model.</td>
<td>R*</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
<td>R</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the mining model.</td>
<td>R/W</td>
</tr>
<tr>
<td>Filter</td>
<td>Filters the case rows used to train the mining model.</td>
<td>R/W**</td>
</tr>
<tr>
<td>FromClause</td>
<td>Specifies the FROM clause of the SQL query that returns the cases for the mining model.</td>
<td>R/W**</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the mining model is visible to client applications.</td>
<td>R/W</td>
</tr>
<tr>
<td>JoinClause</td>
<td>Specifies the JOIN clause of the SQL query that returns the cases for the mining model.</td>
<td>R/W**</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>The date and time when the mining model was last processed.</td>
<td>R</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>A user-specified date. It is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
<td>R/W</td>
</tr>
<tr>
<td>MiningAlgorithm</td>
<td>Identifies the mining algorithm used by the mining model.</td>
<td>R/W</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the mining model.</td>
<td>R/W</td>
</tr>
<tr>
<td>Parameters</td>
<td>The string that contains parameter value settings for the MiningAlgorithm property.</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object.</td>
<td><strong>R</strong></td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td><strong>SourceCube</strong></td>
<td>Returns a reference to the cube used to define a mining model.</td>
<td><strong>R/W</strong>*</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Indicates the status of the mining model.</td>
<td><strong>R</strong></td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type.</td>
<td><strong>R</strong></td>
</tr>
<tr>
<td><strong>TrainingQuery</strong></td>
<td>Identifies the query used for training the mining model.</td>
<td><strong>R/W</strong></td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td>Returns the Extensible Markup Language (XML) representation of a trained mining model.</td>
<td><strong>R</strong></td>
</tr>
</tbody>
</table>

* This property applies only to mining models of **SubClassType** `sbclsOlap`.

** This property applies only to mining models of **SubClassType** `sbclsRelational`. 
AreKeysUnique (clsMiningModel)

The AreKeysUnique property of a clsMiningModel object indicates whether key columns (that is, a clsColumn object with an IsKey property set to True) defined in the Columns collection uniquely identify members in the case table.

Note This property applies only to mining model objects of SubClassType sbclsRelational.

Data Type

Boolean

Access

Read/write

Remarks

The AreKeysUnique property determines whether the relational mining model adds the DISTINCT keyword to the SQL SELECT query used to retrieve the training data set from the case tables. If the values for the key columns identified in the data mining model are unique in the case tables, setting this property to True can improve performance when the relational data mining model is trained.

See Also

clsColumn
CaseDimension (clsMiningModel)

The CaseDimension property of an object of ClassType clsMiningModel identifies the dimension that contains the cases for the mining model. This property applies only to mining models of SubClassType sbclsOlap.

Data Type
String

Access
Read/write

Remarks
The CaseDimension property must be set to a visible shared or private dimension used by the source cube (that is, an object of ClassType clsDatabaseDimension or clsCubeDimension whose IsVisible property is set to True). If the IsVisible property of the shared or private dimension is set to False, or if the dimension is not used by the source cube specified in the SourceCube property, an error is raised.

Examples

Building an OLAP Mining Model

The following example builds an OLAP data mining model and sets its case dimension to Customers:

dsoDmm.Description = "Analyzes the purchasing behavior of customers"
dsoDmm.MiningAlgorithm = "Microsoft_Decision_Trees"
dsoDmm.SourceCube = "Sales"
dsoDmm.CaseDimension = "Customers"
dsoDmm.TrainingQuery = ""Let DSO figure out the training query."
See Also

clsMiningModel
SourceCube
CaseLevel (clsMiningModel)

The CaseLevel property of an object of ClassType clsMiningModel identifies the level of the CaseDimension that contains the cases for the mining model. This property applies only to mining models of SubClassType sbclsOlap.

Data Type

String

Access

Read-only

Remarks

The value of the CaseLevel property represents the name of the lowest enabled and visible level of the dimension specified in the CaseDimension property.

Example

The following example builds an OLAP data mining model and sets its case dimension to Customers. The lowest enabled and visible level in the Customers dimension is Name, so the CaseLevel property is set to the Name level.

dsoDmm.Description = "Analyzes the purchasing behavior of custome
dsoDmm.MiningAlgorithm = "Microsoft_Decision_Trees"
dsoDmm.SourceCube = "Sales"
dsoDmm.CaseDimension = "Customers"
' Save the changes to the data mining model. This also populates
' the Columns collection and sets the CaseLevel property.
dsoDmm.Update
' The dsoDmm.CaseLevel property should have a value of "Name",
' the lowest enabled level of the Customers dimension.
Debug.Print dsoDmm.CaseLevel
See Also

clsMiningModel
ClassType (clsMiningModel)

The **ClassType** property of an object of **ClassType clsMiningModel** returns an enumeration constant that identifies the specific class type.

**Data Type**

ClassTypes

**Access**

Read-only

**Remarks**

The **ClassType** property always returns **clsMiningModel** for this object. To distinguish between relational and OLAP data mining models, use the **SubClassType** property.

**See Also**

clsMiningModel

SubClassType (clsMiningModel)
Description (clsMiningModel)

The Description property of an object of ClassType clsMiningModel sets or returns the description of the mining model. This property is used only by Decision Support Objects (DSO) and is not accessible by client applications.

Data Type

String

Access

Read/write

Example

Use the following code to set the Description property for a mining model object:

' Assume an object (dsoDMM) of ClassType clsMiningModel exists.
dsoDMM.Description = "1999 Sales Patterns"

See Also

clsMiningModel
Filter (clsMiningModel)

The filter property is an SQL filter expression that is used to restrict the cases that are used by mining model objects.

Data Type
String

Access
Read/Write

Remarks
This pass-through filter condition is applied to the SQL query that returns the cases for the mining model object.

This property applies only to objects with a ClassType of clsMiningModel and a SubClassType of sbclsRelational.

Example
The following filter restricts the cases to customers involved in the first million transactions:

"sales_fact_1997.transaction_id <= 1000000"

See Also

clsMiningModel
FromClause (clsMiningModel)

The FromClause property of an object of ClassType clsMiningModel specifies the FROM clause of the SQL pass-through query that is used when training the mining model. This property applies only to mining models of SubClassType sbclsRelational.

Data Type

String

Access

Read/Write

Remarks

The FromClause property contains the string used by the data source provider to construct a FROM clause for the SQL pass-through query that is used to return the training data set for the mining model.

Note You must separate the table and column names with the delimiters appropriate to the source database. You can use the CloseQuoteChar and OpenQuoteChar properties of the DataSource object to determine the correct delimiters.

Example

The following code example shows the FromClause property being set use to two tables, sales_fact_1997 and customer, to provide training data:

' Assume the existence of a clsMiningModel object named dsoDMM. dsoDMM.FromClause = """sales_fact_1997"", ""customer""'"

The previous code example sets the FromClause property to the following string:
"sales_fact_1997", "customer"

**See Also**

[clsMiningModel](#)
IsVisible (clsMiningModel)

The IsVisible property of an object of ClassType clsMiningModel determines whether the mining model is visible to client applications.

**Data Type**

Boolean

**Access**

Read/write

**See Also**

[clsMiningModel](#)
JoinClause (clsMiningModel)

The JoinClause property of an object of ClassType clsMiningModel specifies the JOIN clause of the SQL query that returns the cases for the mining model. This property applies only to mining models of SubClassType sbclsRelational.

Data Type

String

Access

Read/write

Remarks

This property specifies the INNER JOIN clause of the SQL pass-through query that is used to generate the training cases for the mining model. Use this property when the case information is distributed in more than one table.

Note You must separate the table and column names with the delimiters appropriate to the source database. You can use the CloseQuoteChar and OpenQuoteChar properties of the DataSource object to determine the correct delimiters.

Example

In the following code example, the JoinClause is used to join the sales_fact_1997 and customer tables:

' Assume the existence of a clsMiningModel object named dsoDMM. dsoDMM.JoinClause = """"sales_fact_1997"".""customer_id"
" & _ """"customer"".""customer_id""

The previous code example sets the JoinClause property to the following string: "sales_fact_1997"."customer_id" = "customer"."customer_id"."
See Also

clsMiningModel
LastProcessed (clsMiningModel)

The LastProcessed property of an object of ClassType clsMiningModel contains the date and time the mining model was last processed.

Data Type
Date

Access
Read-only

See Also
clsMiningModel
LastUpdated (clsMiningModel)

The LastUpdated property of an object of ClassType clsMiningModel is not used by Microsoft® SQL Server™ 2000 Analysis Services. You can set this to any date/time value you want. For example, you can use it to indicate when the source data was last changed.

The LastUpdated property of an object of ClassType clsMiningModel is user controlled and not set by Analysis Services. That is, the user controls the value and context of this property; the server does not set this value or change it at any time. This means that you can use it to indicate the date when the data in a source was last changed, or the last time the mining model was accessed.

Data Type
String

Access
Read/write

Remarks
The LastUpdated property is not automatically set by any method in the Decision Support Objects (DSO) object model. It is provided as a means for client applications to specify a date or time that represents the validity of information. For example, a date of 12/31/1997 may mean that the information stored in a data mining model is not valid after December 1997.

See Also

clsMiningModel
**MiningAlgorithm** (clsMiningModel)

The **MiningAlgorithm** property of an object of **ClassType clsMiningModel** identifies the mining algorithm used by the mining model. Only algorithms listed in the MINING_SERVICES schema rowset can be used.

**Data Type**

String

**Access**

Read/write

**Remarks**

By default, Microsoft® SQL Server™ 2000 Analysis Services supports two algorithms, **Microsoft_Clustering** and **Microsoft_Decision_Trees**. Because the list of mining algorithms may vary dynamically, the **MiningAlgorithm** property is a string and not an enumeration.

**See Also**

- clsMiningModel
- Data Mining Schema Rowsets
Name (clsMiningModel)

The Name property of an object of ClassType clsMiningModel contains the name of the mining model.

Data Type

String

Access

Read/write (read-only after object is named)

Example

Use the following code to return the name of a mining model:

' Assume an object (dsoDMM) of ClassType clsMiningModel exists.
Dim strName As String
strName = dsoDMM.Name

See Also

clsMiningModel
Analysis Services Programming

**Parameters (clsMiningModel)**

The **Parameters** property of an object of **ClassType clsMiningModel** stores parameter value settings for the algorithm specified in the **MiningAlgorithm** property of the mining model. Parameters can be combined within a string by separating each one with a semicolon.

**Data Type**

String

**Access**

Read/write

**Remarks**

Settings for the **Parameters** property must conform to the parameters specified in the SERVICE_PARAMETERS schema rowset. Decision Support Objects (DSO) does not validate the settings used in the property string. Therefore, the string is appended without validation to the CREATE MINING MODEL (for relational data mining models) or CREATE OLAP MINING MODEL (for OLAP data mining models) statement used to create the data mining model. The Analysis server, on the other hand, checks for valid parameter settings and returns errors as appropriate.

**Example**

The following example sets the parameters for a data mining algorithm.

' Assume an object (dsoDMM) of ClassType clsMiningModel and SubClassType sbclsRelational exists.
' Set the MiningAlgorithm property to use Microsoft Decision Trees.
dsoDMM.MiningAlgorithm = "Microsoft Decision Trees"
This algorithm supports the MINIMUM_LEAF_CASES mining parameter. dsoDMM.Parameters = "MINIMUM_LEAF_CASES=15"

**See Also**

[clsMiningModel](#)

[Data Mining Schema Rowsets](#)
Parent (clsMiningModel)

The **Parent** property of an object of **ClassType clsMiningModel** contains a reference to the parent database object of the mining model.

**Data Type**

MDStore

**Access**

Read-only

**Example**

The following example will print the string "*mining model is owned by database*, where *mining model* is the name of the mining model and *database* is the name of the database that owns the mining model:

' Assume the existence of a mining model object called dsoDMM. Debug.Print dsoDMM.Name & " is owned by " & dsoDMM.Parent.Name

**See Also**

clsMiningModel
SourceCube (clsMiningModel)

The SourceCube property of an object of ClassType clsMiningModel specifies the cube that provides the source data for the mining model. This property applies only to mining models of SubClassType sbclsOlap.

Data Type

String

Access

Read/write

Remarks

The source cube of a mining model must reside in the same database as the mining model itself. The specified source cube must be visible (that is, the IsVisible property of the clsCube object must be set to True). If the IsVisible property of the source cube is set to False, an error is raised.

Note  A mining model cannot use a virtual cube which already contains a mining dimension as a source cube.

Example

The following example specifies the City level of the Customer dimension be used to generate training cases for the mining model from the Sales cube.

' Assume the existence of a mining model object named dsoDMM.
  dsoDMM.SourceCube = "Sales"
  dsoDMM.CaseDimension = "Customer"

See Also

clsMiningModel
State (clsMiningModel)

The State property of an object of ClassType clsMiningModel returns an enumeration constant that indicates the processing state of the object on the server.

Data Type

OlapStateTypes

Access

Read-only

Remarks

The supported OlapStateTypes enumeration constants for the State property are:

- olapStateNeverProcessed
- olapStateCurrent
- olapStateStructureChanged

When a mining model is first created, the value for the State property is olapStateNeverProcessed. After processing, the value becomes olapStateCurrent. If structural changes are made to the Columns collection of the model after processing, the value becomes olapStateStructureChanged. If source mapping changes are made to the Columns collection after processing (that is, if changes are made to the clsColumn SourceTable or SourceColumn properties), the value becomes olapStateSourceMappingChanged.

See Also

clsMiningModel
SubClassType (clsMiningModel)

The **SubClassType** property of an object of **ClassType clsMiningModel** returns an enumeration constant identifying the specific subclass type.

**Data Type**

*SubClassTypes*

**Access**

Read-only

**Remarks**

Objects of **ClassType clsMiningModel** can have a **SubClassType** property value of **sbclsRegular**, **sbclsOlap**, or **sbclsRelational**. A mining model has a **SubClassType** value of **sbclsRelational** if it is defined on one or more relational tables. If the mining model is defined on a cube residing in the same **clsDatabase** object, the **SubClassType** value is **sbclsOlap**. The **sbclsRelational** constant is equivalent to the **sbclsRegular** value and is provided for convenience and readability in source code.

**Example**

The following example prints the types of each data mining model in the **FoodMart 2000** database:

```
' Assume the existence of a server object, s, that has been connected to
Dim db as DSO.DB ' declare an interface for the database.
Dim dmm as DSO.MiningModel
Dim sDmmType as String ' Description of each enumeration value.
set db = s.MDStores("FoodMart")
For each dmm in db.MiningModels
   Select Case dmm.subclasstype
```
Case sbclsOlap
  sDmmType = "sbclsOlap"
Case sbclsRelational
  sDmmType = "sbclsRelational"
Case else
  sDmmType = "Unknown subclass type!"
End Select
  debug.print dmm.name & " is type " & sDmmType
Next

See Also

clsMiningModel
TrainingQuery (clsMiningModel)

The TrainingQuery property of an object of ClassType clsMiningModel identifies the SQL INSERT statement used to train the mining model.

Data Type

String

Access

Read/write

Remarks

If the TrainingQuery property is not set, the SQL INSERT statement for this property is automatically created by Decision Support Objects (DSO) based on the Columns collection of the mining model. TrainingQuery property values are not validated beforehand; they are sent directly to the Analysis server for training the mining model.

See Also

clsMiningModel
XML (clsMiningModel)

The XML property of an object of ClassType clsMiningModel returns the Extensible Markup Language (XML) representation of the data mining model.

**Data Type**

String

**Access**

Read-only

**Remarks**

The XML property only returns the XML representation of a data mining model if the data mining model has been trained. If the model has not been trained using the Process method, this property returns an empty string.

**See Also**

clsMiningModel
**clsMiningModelRole**

An object of the **ClassType clsMiningModelRole** provides a specific implementation of the Decision Support Objects (DSO) **Role** interface for data mining models. This object provides collections, methods, and properties through the **Role** interface.

You use objects of **ClassType clsMiningModelRole** to manage the set of users who can access a mining model and the manner in which they can access it. A mining model role has a name, a description, a parent object, a class type, a list of users, and a set of permissions. Each permission has a key and a corresponding permission expression.

You create roles at the database level (database roles) and then assign them to mining models (mining model roles) by adding them to the collection of roles associated with the mining model.

You can remove a database role by removing it from the database's collection of role objects. When you do so, the system automatically removes the corresponding mining model roles from the mining model's collection of role objects.

You can remove a mining model role by removing it from the mining model's collection of role objects. When you do so, the corresponding database role is not affected. However, the definition of the mining model role remains in effect until you update or process the cube.

**Examples**

**Using clsMiningModelRole**

If dsoDb.DataSources("DMTest") Is Nothing Then
  Set dsoDs = dsoDb.DataSources.AddNew("DMTest")
  dsoDs.ConnectionString = "provider=Microsoft.Jet.OLEDB.4.0;dat:
  dsoDs.Update
End If
'Create a new mining model role.
If dsoDb.Roles("DMDev") Is Nothing Then
    Set dsoRole = dsoDb.Roles.AddNew("DMDev")
    dsoRole.UsersList = "DOMAIN\SomeUser"
    dsoRole.Update
End If

'Check to see whether the mining model exists.
If Not dsoDb.MiningModels("CustSalesRel") Is Nothing Then
    'Delete it if it does.
    dsoDb.MiningModels.Remove "CustSalesRel"
End If
'Now create the model afresh.
Set dsoDmm = dsoDb.MiningModels.AddNew("CustSalesRel")
'Add a new datasource for the model
dsoDmm.DataSources.AddNew "DMTest"
'Add a data mining role to the new mining model.
Set dsoRole = dsoDmm.Roles.AddNew("DMDev")

See Also

Collections, clsMiningModelRole
Methods, clsMiningModelRole
Properties, clsMiningModelRole
Role Interface
Security
Collections, clsMiningModelRole

An object of ClassType clsMiningModelRole implements the following collections of the Role interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>The collection of commands for the role</td>
</tr>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

See Also

clsMiningModelRole

Role Interface
Methods, clsMiningModelRole

An object of ClassType clsMiningModelRole implements the following method of the Role interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPermissions</td>
<td>Sets the permissions for the cube role for a given key</td>
</tr>
</tbody>
</table>

See Also

clsMiningModelRole

Role Interface
Properties, clsMiningModelRole

An object of ClassType clsMiningModelRole implements the following properties of the Role interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the mining model role</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the role structure is valid</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the mining model role</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent object, using the MDStore interface of the parent object</td>
</tr>
<tr>
<td>ParentObject</td>
<td>Returns a reference to the parent object, using the default interface of the parent object</td>
</tr>
<tr>
<td>Permissions</td>
<td>The permissions for the mining model role for a given key</td>
</tr>
<tr>
<td>SubClassType</td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td>UsersList</td>
<td>A semicolon-delimited list of users of the mining model role</td>
</tr>
</tbody>
</table>

See Also

clsMiningModelRole
Role Interface
clsPartition

An object of ClassType clsPartition serves as a data store for multidimensional cubes. It provides an implementation of the Decision Support Objects (DSO) MDStore interface specific to partitions. This object provides collections, methods, and properties through the MDStore interface.

For more information about partitions, see Partitions.

Example

Use the following code to create an object of ClassType clsPartition:

```vbnet
'Assume an object (dsoCube) of ClassType clsCube exists
Dim dsoPartition As DSO.MDStore
Set dsoPartition = dsoCube.MDStores.AddNew("MyPartition")
```

See Also

MDStore Interface  
Collections, clsPartition  
Methods, clsPartition  
Properties, clsPartition
Analysis Services Programming

**Collections, clsPartition**

An object of **ClassType clsPartition** implements the following collections of the **MDStore** interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties for the partition</td>
</tr>
<tr>
<td><strong>DataSources</strong></td>
<td>The collection of data source objects used by the partition</td>
</tr>
<tr>
<td><strong>Dimensions</strong></td>
<td>The collection of dimension objects defined in the partition</td>
</tr>
<tr>
<td><strong>MDStores</strong></td>
<td>The collection of aggregation objects defined in the partition</td>
</tr>
<tr>
<td><strong>Measures</strong></td>
<td>The collection of measure objects defined in the partition</td>
</tr>
</tbody>
</table>

**See Also**

- clsPartition
- MDStore Interface
Analysis Services Programming

**Methods, clsPartition**

An object of **ClassType clsPartition** implements the following methods of the **MDStore** interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clone</strong></td>
<td>Copies the property values (and optionally) the collections of major and minor objects from one partition object to another</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>Locks an object to prevent multiple users from concurrently changing the object</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>Merges two partitions</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Processes the partition</td>
</tr>
<tr>
<td><strong>UnlockObject</strong></td>
<td>Releases a lock previously established by the <strong>LockObject</strong> method</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td>Updates the partition definition in the meta data repository</td>
</tr>
</tbody>
</table>

**See Also**

- clsPartition
- MDStore Interface
Properties, clsPartition

An object of **ClassType clsPartition** implements the following properties of the **MDStore** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowDrillThrough</td>
<td>Indicates whether drillthrough is allowed on the partition.</td>
</tr>
<tr>
<td>AggregationPrefix</td>
<td>The aggregation prefix for the partition store.</td>
</tr>
<tr>
<td>Analyzer</td>
<td>The partition analyzer object for this partition.</td>
</tr>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
</tr>
<tr>
<td>DefaultMeasure</td>
<td>The name of the default measure for the partition.</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the partition.</td>
</tr>
<tr>
<td>DrillThroughColumns</td>
<td>The list of columns that are included in a drillthrough query.</td>
</tr>
<tr>
<td>DrillThroughFilter</td>
<td>A statement restricting rows that are returned by a drillthrough query.</td>
</tr>
<tr>
<td>DrillThroughFrom</td>
<td>An SQL FROM clause with the names of the tables used in drillthrough queries.</td>
</tr>
<tr>
<td>DrillThroughJoins</td>
<td>An SQL JOIN clause with the names of the tables used in drillthrough queries.</td>
</tr>
<tr>
<td>EnableRealTimeUpdates</td>
<td>For relational OLAP (ROLAP) partitions, indicates whether real-time update capability is enabled for the partition.</td>
</tr>
<tr>
<td>EstimatedRows</td>
<td>The estimated number of rows in the partition.</td>
</tr>
<tr>
<td>EstimatedSize</td>
<td>The estimated size of all the rows in bytes.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>Contains the SQL FROM clause from the list of tables used to define the partition's dimensions and measures.</td>
</tr>
<tr>
<td><strong>IsDefault</strong></td>
<td>Indicates whether the partition is the default partition.</td>
</tr>
<tr>
<td><strong>IsTemporary</strong></td>
<td>Indicates whether the partition should be stored in the repository.</td>
</tr>
<tr>
<td><strong>IsReadWrite</strong></td>
<td>Indicates whether the partition object is writable.</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the structure of the partition is valid.</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>The JOIN clause (list of join conditions, separated by AND) for the partition.</td>
</tr>
<tr>
<td><strong>LastProcessed</strong></td>
<td>The date and time the partition was last processed.</td>
</tr>
<tr>
<td><strong>LastUpdated</strong></td>
<td>A user-specified date. It is not used by Microsoft® SQL Server™ 2000 Analysis Services.</td>
</tr>
<tr>
<td><strong>LazyOptimizationProgress</strong></td>
<td>Indicates the progress of lazy optimization processing on a multidimensional OLAP (MOLAP) partition.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the partition.</td>
</tr>
<tr>
<td><strong>OlapMode</strong></td>
<td>Returns an enumeration constant that identifies the type of OLAP storage mode.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent <strong>MDStore</strong> object.</td>
</tr>
<tr>
<td><strong>ProcessingKeyErrorLimit</strong></td>
<td>Sets the number of allowable errors that cause processing to cease.</td>
</tr>
<tr>
<td><strong>ProcessingKeyErrorLogFileName</strong></td>
<td>The UNC path to a file for logging dimension key errors encountered during processing.</td>
</tr>
<tr>
<td><strong>RemoteServer</strong></td>
<td>The name of the remote server where the data for the partition is stored.</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Returns a reference to the <strong>DSO.Server</strong> object.</td>
</tr>
<tr>
<td><strong>SourceTable</strong></td>
<td>The name of the fact table for the partition.</td>
</tr>
<tr>
<td><strong>SourceTableAlias</strong></td>
<td>The alias of the source table for the partition.</td>
</tr>
<tr>
<td><strong>SourceTableFilter</strong></td>
<td>Contains the WHERE clause of the SQL statement used to determine which source table rows are to be included in the partition.</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Returns an enumeration constant that indicates the difference between the partition object referenced by the client application and corresponding partition on the Analysis server.</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object.</td>
</tr>
</tbody>
</table>

**See Also**

**clsPartition**

**MDStore Interface**
Analysis Services Programming

**clsPartitionAnalyzer**

In multidimensional database technology, you must balance precalculated aggregation storage requirements against online query process performance. A high percentage of aggregations increases query speed but requires more storage space.

![Performance vs. Size](image)

The number of aggregations that must be precalculated and stored increases proportionally to the level of query performance.

A Decision Support Objects (DSO) object of **ClassType clsPartitionAnalyzer** encapsulates an algorithm that automatically designs a set of aggregations in a partition. It analyzes the schema of a partition and generates a collection of aggregations that improves query performance. You can run the analysis without constraints, or you can constrain the analysis in either of the following ways:

- Specify one or more goal queries that you want to optimize.

- Include existing aggregations or aggregations that should be preserved before the analysis is run.
To analyze a partition using DSO, follow these steps:

1. Initialize the analysis session using the `InitializeDesign` method.

2. Add one or more goal queries using the `AddGoalQuery` and `PrepareGoalQueries` methods. The resulting members of the `DesignedAggregations` collection will be optimized for this set of goal queries. If no goal queries are specified, the analysis will yield a generalized optimization.

3. Add one or more existing aggregations using the `AddExistingAggregation` method.

4. Perform an initial analysis using the `NextAnalysisStep` method.
   
   The analysis generates new aggregations that are added to the `DesignedAggregations` collection. It also returns the calculated percentage performance gain, aggregation storage requirements, and total number of aggregations created.

5. Review the results of the analysis step and determine whether you want to perform another analysis iteration. Running subsequent analysis steps adds new aggregations to the `DesignedAggregations` collection and recalculates the percentage performance gain, aggregation storage requirements, and total number of aggregations created.

6. Manually or programmatically determine the point at which you want to conclude the analysis.

7. Optionally, when the partition analyzer is finished running, replace the aggregations of the partition with the members of the `DesignedAggregations` collection.

8. Close the analysis with the `CloseAggregationsAnalysis` method.
An object of **ClassType clsPartitionAnalyzer** provides collections, methods, and properties through its own internal interface.

**Example**
This example analyzes the default partition of a cube and designs aggregations that can fulfill 20% of all possible queries without having to access the fact table:

' CreateAggregations - design aggregations for the cube.
',

Public Sub CreateAggregations()
  ' aggregations are designed per partition
  ' get the default partition from the cube
  ' m_dsoCube is a publicly declared variable
  ' of DSO ClassType clsCube
  Dim dsoPartition As DSO.MDStore
  Set dsoPartition = m_dsoCube.MDStores(1)

  ' First set the storage mode of the partition.
  ' This example sets it to MOLAP
  ' (facts and aggregations are loaded into
  ' multidimensional structures on the OLAP server).
  ' olapmodeMolapIndex is an enumerated constant indicating
  ' that the storage mode for a partition is MOLAP.
  dsoPartition.OlapMode = olapmodeMolapIndex

  ' Get the partition analyzer.
  Dim dsoPartitionAnalyzer As DSO.PartialAnalyzer
  Set dsoPartitionAnalyzer = dsoPartition.Analyzer

  ' Initialize the analyzer.
  dsoPartitionAnalyzer.InitializeDesign

  ' Design aggregations for 20% of queries.
' NextAnalysisStep incrementally builds the
' optimal set of aggregations.
' Tell the partition analyzer to stop designing
' aggregations when PercentageBenefit reaches 20.
Dim PercentageBenefit As Double
Dim AccumulatedSize As Double
Dim AggregationsCount As Long
Do While dsoPartitionAnalyzer.NextAnalysisStep(PercentageBenefit,
    AccumulatedSize, _
    AggregationsCount)
    If PercentageBenefit > 20# Then
        Exit Do
    End If
Loop

' Apply the designed aggregations to the partition.
Dim dsoAggregation As DSO.MDStore
For Each dsoAggregation In dsoPartitionAnalyzer.DesignedAggregations
    dsoPartition.MDStores.Add dsoAggregation
Next

' Close the analyzer.
dsoPartitionAnalyzer.CloseAggregationsAnalysis

' Save the cube definition in the meta data repository.
On Error GoTo Err_Update
dsoPartition.Update
Exit Sub

Err_Update:
    ' Failed to persist the cube definition in the meta data repository
    ' Possible reasons:
' - the meta data repository is unreachable
' you can see where the meta data repository resides by looking
' up the following registry entry:
' HKEY_LOCAL_MACHINE\Software\Microsoft\OLAP Server\Connection Info
'     Repository Connection String
' - the DSO cube object is being locked by another DSO application
'   It is not possible for two DSO applications to persist the
'   same object at the same time.
'   It is not possible to persist a DSO object because another DSO
'   application has explicitly locked it.

MsgBox "Aggregation design for partition failed" & vbCrLf & Err.Description
End Sub

See Also

Aggregations
Collections, clsPartitionAnalyzer
Methods, clsPartitionAnalyzer
Properties, clsPartitionAnalyzer
Collections, clsPartitionAnalyzer

An object of **ClassType clsPartitionAnalyzer** implements the following collection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignedAggregations</td>
<td>The designed aggregations generated by the object of <strong>ClassType clsPartitionAnalyzer</strong></td>
</tr>
</tbody>
</table>

**Access**

Read-only

**See Also**

[clsPartitionAnalyzer](#)
DesignedAggregations (clsPartitionAnalyzer)

The DesignedAggregations collection of an object of ClassType clsPartitionAnalyzer acts as a temporary container for aggregation objects during the partition analyzer session.

Data Type
VBA.Collection

Access
Read-only

Remarks
This collection contains aggregations (that is, objects of ClassType clsAggregation) that were added manually using the AddExistingAggregation method or were automatically generated using the NextAnalysisStep method. At the conclusion of the partition analyzer session you can either save the aggregations to the partition (and make them available for client applications) or discard them.

Example
Use the following code to repeatedly invoke the NextAnalysisStep method and then save the DesignedAggregations in a Microsoft® Visual Basic® collection. The analysis continues until one of the following goals is reached:

- Twenty or more aggregations are designed.
- The storage requirements for the designed aggregations exceed 100,000 bytes.

For more information, see InitializeDesign.
'Assume the existence of objects (dsoPartAnalyzer) of ClassType 'clsPartitionAnalyzer and (dsoPartition) of ClassType clsPartition.

Private blnStopAdding As Boolean
Private colDesignedAggs As Collection

dsoPartAnalyzer.InitializeDesign

'Iterate through analysis until either goal is reached.
Do Until blnStopAdding
  If Not dsoPartAnalyzer.NextAnalysisStep(dblPercentageBenefit, _
    dblAccumulatedSize, lngAggregationsCount) Then
    blnStopAdding = True 'No New Aggregations Designed
  Else
    blnStopAdding = (lngAggregationsCount >= 20) Or _
    (dblAccumulatedSize >= 100000)
  End If
Loop

'Save the designed aggregations to the partition.
Dim dsoAggregation As DSO.MDStore
For Each dsoAggregation In dsoPartAnalyzer.DesignedAggregations
  dsoPartition.MDStores.Add dsoAggregation
Next

See Also

AddExistingAggregation
clsAggregation
clsPartition
clsPartitionAnalyzer
NextAnalysisStep
Methods, clsPartitionAnalyzer

An object of ClassType clsPartitionAnalyzer implements the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddExistingAggregation</td>
<td>Adds an existing aggregation to the DesignedAggregations collection</td>
</tr>
<tr>
<td>AddGoalquery</td>
<td>Adds a specific query for the analyzer to optimize</td>
</tr>
<tr>
<td>CloseAggregationsAnalysis</td>
<td>Closes the partition analyzer session and clears the objects used during the analysis</td>
</tr>
<tr>
<td>InitializeDesign</td>
<td>Checks the partition structure to ensure that the partition analyzer can be run and initializes the objects necessary to perform the analysis</td>
</tr>
<tr>
<td>NextAnalysisStep</td>
<td>Adds new aggregations to the DesignedAggregations collection and calculates the incremental performance gain and the additional aggregation storage requirements</td>
</tr>
<tr>
<td>PrepareGoalQueries</td>
<td>Prepares the goal queries that were entered using the AddGoalQuery method for use in this analysis session</td>
</tr>
</tbody>
</table>

See Also

clsPartitionAnalyzer
AddExistingAggregation (clsPartitionAnalyzer)

The **AddExistingAggregation** method of an object of **ClassType clsPartitionAnalyzer** adds an aggregation to the **DesignedAggregations** collection. It examines the aggregation and returns the calculated percentage performance gain, the total size of the aggregation, and the total number of aggregations in the partition that result from the inclusion of the aggregation.

**Syntax**

```
object. AddExistingAggregation(ByVal agg As MDStore, PercentageBenefit As Double, AccumulatedSize As Double, AggregationsCount As Long)
```

- **object**
  - The object of **ClassType clsPartitionAnalyzer** used to perform the analysis.

- **agg**
  - An **MDStore** object, representing the aggregation to add.

- **PercentageBenefit**
  - The estimated percentage performance improvement that would be realized using the current collection of **DesignedAggregations**, as opposed to querying against the underlying fact table. This argument is used as an output parameter.

- **AccumulatedSize**
  - The estimated hard disk storage requirements (in bytes) for the current collection of **DesignedAggregations**. This argument is used as an output parameter.

- **AggregationsCount**
  - The number of aggregations contained in the current collection of **DesignedAggregations**. This argument is used as an output parameter.

**Remarks**
This method allows you to evaluate the impact of a particular aggregation on query performance. Subsequent analysis steps performed either by adding another existing aggregation or by using the `NextAnalysisStep` method include the added aggregation.

**Example**

Use the following code to add an aggregation, named `Agg123`, from the `MDStores` collection, which contains objects of `ClassType clsAggregation` of an existing partition to the `DesignedAggregations` collection, and then run several analysis steps. The analysis continues until one of the following two goals is reached:

- Twenty or more aggregations are designed.
- The storage requirements for the designed aggregations exceed 100,000 bytes.

For more information, see `InitializeDesign`.

'Assume an object (dsoCube) of ClassType clsCube exists 'with two partitions referenced by partition objects 'dsoPart1 and dsoPart2.
Dim bStopAdding As Boolean
Dim dblPercentageBenefit As Double
Dim dblAccumulatedSize As Double
Dim lngAggregationsCount As Long

'Get existing aggregation "Agg123" from first partition.
Dim dsoExistAgg As DSO.MDStore
Set dsoExistAgg = dsoPart1.MDStores("Agg123")

'Add aggregation to second partition.
Dim dsoPartAnalyzer As DSO.PartitionAnalyzer
Set dsoPartAnalyzer = dsoPart2.Analyzer
dsoPartAnalyzer.AddExistingAggregation dsoExistAgg, _
dsoPartAnalyzer.InitializeDesign

'Iterate through analysis until either goal is reached. Do Until bStopAdding
   If Not dsoPartAnalyzer.NextAnalysisStep(dblPercentageBenefit, _
      dblAccumulatedSize, lngAggregationsCount) Then
      bStopAdding = True 'No New Aggregations Designed
   Else
      bStopAdding = (lngAggregationsCount >= 20) Or _
         (dblAccumulatedSize >= 100000)
   End If
Loop

See Also

clsAggregation

clsPartition

clsPartitionAnalyzer

DesignedAggregations

NextAnalysisStep
AddGoalQuery (clsPartitionAnalyzer)

The **AddGoalQuery** method of an object of **ClassType clsPartitionAnalyzer** describes a specific aggregation that is to be generated during a partition analyzer session.

**Syntax**

```csharp
object. AddGoalQuery(ByVal DatasetName As String, ByVal Frequency As Double)
```

**object**

The object of **ClassType clsPartitionAnalyzer** used to perform the analysis.

**DatasetName**

A numeric string that specifies which level from each dimension is to be included in the query. This string must have the same number of characters as the number of dimensions in the partition. For example, the string "223" would refer to a query involving three dimensions, using the second level from the first dimension, the second level from the second dimension, and the third level from the third dimension.

**Frequency**

A weighting factor that corresponds to the number of times that an existing **DatasetName** query has previously been executed. If this **DatasetName** is a new query, the value of **Frequency** that is entered is used to weight the partition analysis. As the frequency becomes greater, the weight that is placed on the query during the analysis increases.

**Remarks**

The analysis performed by the partition analyzer can be constrained to optimize a particular subset of queries. Each of the queries in this subset is called a **goal query** and is specified by picking a level from each of the dimensions of the partition. This method adds the goal query to an internal collection that is used to
generate the aggregations.

Goal queries can be obtained from the query log database maintained by the Analysis server. The clsCubeAnalyzer object can retrieve query log recordsets containing the dataset names of logged queries for a specific cube. For more information about query log recordsets, see OpenQueryLogRecordset (clsCubeAnalyzer).

You do not have to select goal queries before running the NextAnalysisStep method to generate aggregations. The clsPartitionAnalyzer object will create a generic set of aggregations without guidance. However, if you do specify one or more goal queries, the partition analyzer will create aggregations according to your requests only.

Different goal query subsets can be optimized with significantly different members of the DesignedAggregations collection. The performance and storage requirements may warrant constructing separate partitions for each goal query subset.

The levels referenced in the DatasetName string are identified by matching each numeric value with a dimension. The order of reference is determined by the order of levels in the Dimensions collection of the partition.

**Example**

Assume that a partition contains the following dimensions and levels.

<table>
<thead>
<tr>
<th></th>
<th>Customers dimension</th>
<th>Products dimension</th>
<th>Store Locations dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level #1</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Level #2</td>
<td>Groups</td>
<td>Brand</td>
<td>Country</td>
</tr>
<tr>
<td>Level #3</td>
<td>Customer#</td>
<td>SKU</td>
<td>State</td>
</tr>
<tr>
<td>Level #4</td>
<td></td>
<td></td>
<td>City</td>
</tr>
</tbody>
</table>

A particular group of users needs to perform the following queries:

- All customers (Level #1) by product SKU (Level #3) by city (Level #4)
  
  This goal query aggregation is specified with DatasetName = "134"

- Individual customers (Level #3) by all products (Level #1) by state
Use the following code to add these goal queries. For more information, see `InitializeDesign`.

' Assume the existence of an object (dsoPartAnalyzer) of ClassType clsPartitionAnalyzer.
' First, call InitializeDesign.
   dsoPartAnalyzer.InitializeDesign

' Add goal queries.
   dsoPartAnalyzer.AddGoalQuery "134", 1
   dsoPartAnalyzer.AddGoalQuery "313", 1
' Because the following goal query is executed more often than
' the other two, a higher frequency value is assigned to give
' it more weight when being considered for aggregation purposes.
   dsoPartAnalyzer.AddGoalQuery "222", 5

' Required after all goal queries have been added.
   dsoPartAnalyzer.PrepareGoalQueries

You can now run a series of `NextAnalysisStep` methods until you achieve the level optimization you want.

**See Also**

- `clsAggregation`
- `clsPartition`
- `clsPartitionAnalyzer`
- `NextAnalysisStep`
clsCubeAnalyzer
CloseAggregationsAnalysis (clsPartitionAnalyzer)

The CloseAggregationsAnalysis method of an object of ClassType clsPartitionAnalyzer closes the partition analyzer session and clears the temporary objects used during the analysis.

Syntax

object. CloseAggregationsAnalysis

object

The object of ClassType clsPartitionAnalyzer used to perform the analysis.

Remarks

This method does not permanently save the DesignedAggregations collection in the partition. It clears the temporary objects used during the analysis. If you want to save the results from a partition analyzer session you must do so programmatically. The sections in the following example show how to save the results of an analysis in the partition.

Example

Use the following code to run several analysis steps, save the results, and close the analysis. The analysis continues until one of the following goals is reached:

- Twenty or more aggregations are designed.
- The storage requirements for the designed aggregations exceed 100,000 bytes.

For more information, see InitializeDesign.

' Assume the existence of an object (dsoPart) of ClassType ' clsPartition and an object (dsoPartAnalyzer) of ClassType ' clsPartitionAnalyzer.
Private blnStopAdding As Boolean
Dim dblPercentageBenefit As Double
Dim dblAccumulatedSize As Double
Dim lngAggregationsCount As Long

dsoPartAnalyzer.InitializeDesign

' Iterate through analysis until either goal is reached.
Do Until blnStopAdding
    If Not dsoPartAnalyzer.NextAnalysisStep(dblPercentageBenefit, _
        dblAccumulatedSize, lngAggregationsCount) Then
        blnStopAdding = True ' No new aggregations designed.
    Else
        blnStopAdding = (lngAggregationsCount >= 20) Or _
            (dblAccumulatedSize >= 100000)
    End If
Loop

' Delete existing aggregations.
Do While dsoPart.MDStores.Count
    dsoPart.MDStores.Remove 1
Loop

' Add designed aggregations to partition.
For Each Agg In dsoPartAnalyzer.DesignedAggregations
    dsoPart.MDStores.Add Agg
Next Agg

dsoPartAnalyzer.CloseAggregationsAnalysis 'Close the analysis.

See Also

clsAggregation
clsPartition

clsPartitionAnalyzer
InitializeDesign (clsPartitionAnalyzer)

The InitializeDesign method of an object of ClassType clsPartitionAnalyzer checks the partition structure to ensure that clsPartitionAnalyzer can be run on it and initializes the objects necessary for performing the analysis.

Syntax

object. InitializeDesign([ByVal OlapMode])

object

The object of ClassType clsPartitionAnalyzer used to perform the analysis.

OlapMode

An optional Variant value representing an OlapStorageModes constant. If no value is supplied, the value supplied by the AggregationsOLAPMode property of the parent object is used.

Remarks

The InitializeDesign method does not work on partitions associated with linked cubes, because no aggregations are allowed for linked cubes.

Example

Use the following code to create an object of ClassType clsPartitionAnalyzer and initialize its design.

Note  This example will fail if the first cube in the server's collection is a linked cube, because you cannot design aggregations for linked cubes.

Dim dsoServer As DSO.Server
Dim dsoDB As DSO.MDStore
Dim dsoCube As DSO.MDStore
Dim dsoPart As DSO.MDStore
Dim dsoAgg As DSO.MDStore
Dim dsoPartAnalyzer As DSO.PortionAnalyzer
Dim strErr As String

' Initialize server
' LocalHost defaults to your Windows 2000 or Windows NT 4.0 comp
t name.
Set dsoServer = New DSO.Server
dsoServer.Connect("LocalHost")

Set dsoDB = Server.MDStores(1) ' Get first database on server.
Debug.Print "Database Opened: " & dsoDB.Name
Set dsoCube = dsoDB.MDStores(1) ' Get first cube in database.
Debug.Print "Cube Opened: " & dsoCube.Name
Set dsoPart = dsoCube.MDStores(1) ' Get first partition in cube.
Debug.Print "Partition Opened: " & dsoPart.Name
Set dsoPartAnalyzer = dsoPart.Analyzer ' Instantiate an analyzer objec

dsoPartAnalyzer.InitializeDesign

See Also

clsAggregation
clsPartition
clsPartitionAnalyzer
NextAnalysisStep (clsPartitionAnalyzer)

The NextAnalysisStep method of an object of ClassType clsPartitionAnalyzer adds a set of aggregations to the DesignedAggregations collection. It calculates the improved query performance and the storage requirements for the new aggregations.

Syntax

\[ bRet = \text{object. NextAnalysisStep(PercentageBenefit As Double, AccumulatedSize As Double, AggregationsCount As Long)} \]

\[ bRet \]

This value is True if the method completed successfully, False otherwise.

\[ \text{object} \]

The object of ClassType clsPartitionAnalyzer used to perform the analysis.

\[ \text{PercentageBenefit} \]

The estimated percentage performance improvement that would be realized using the current collection of DesignedAggregations, as opposed to querying against the underlying fact table. This is an output parameter.

\[ \text{AccumulatedSize} \]

The estimated hard disk storage requirements (in bytes) for the current collection of DesignedAggregations. This is an output parameter.

\[ \text{AggregationsCount} \]

The number of aggregations contained in the current collection of DesignedAggregations. This is an output parameter.

Remarks

NextAnalysisStep analyzes the schema of a partition and generates a collection of aggregations that improves query performance. You can run the analysis
without constraints. If no constraints are specified, the analysis yields a generalized optimization. For more information, see AddGoalQuery and PrepareGoalQueries.

Example

Use the following code to run a series of analyses until either of the following two goals is reached:

- Twenty or more aggregations are designed.
- The storage requirements for the designed aggregations exceed 100,000 bytes.

For more information, see CloseAggregationsAnalysis and InitializeDesign.

Place the following code in your form's Declarations section:

' Assume the existence of an object (dsoPartAnalyzer) of ClassType ' clsPartitionAnalyzer.
Private blnStopAdding As Boolean
Private dblPercentageBenefit As Double
Private dblAccumulatedSize As Double
Private lngAggregationsCount As Long

' Iterate through analysis until either goal is reached.
Do Until blnStopAdding
    If Not dsoPartAnalyzer.NextAnalysisStep(dblPercentageBenefit, _
        dblAccumulatedSize, lngAggregationsCount) Then
        blnStopAdding = True 'No new aggregations designed.
    Else
        blnStopAdding = (lngAggregationsCount >= 20) Or _
            (dblAccumulatedSize >= 100000)
    End If
Loop
See Also

clsPartitionAnalyzer
DesignedAggregations
PrepareGoalQueries (clsPartitionAnalyzer)

The PrepareGoalQueries method of an object of ClassType clsPartitionAnalyzer analyzes the goal queries that were added using the AddGoalQuery method.

Syntax

```
object. PrepareGoalQueries
```

object

The object of ClassType clsPartitionAnalyzer used to perform the analysis.

Remarks

If you added any goal queries during the partition analyzer session, use this method before calling the NextAnalysisStep method.

Example

For more information on examples using this method, see AddGoalQuery.

See Also

clsAggregation
clsPartition
clsPartitionAnalyzer
Properties, clsPartitionAnalyzer

An object of `ClassType clsPartitionAnalyzer` implements the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AggregationAnalysisInitialized</code></td>
<td>Indicates whether the partition analyzer has been initialized</td>
</tr>
<tr>
<td><code>Parent</code></td>
<td>Contains a reference to the parent <code>MDStore</code> <code>(ClassType clsPartition)</code> object</td>
</tr>
</tbody>
</table>

**Access**

Read-only

**See Also**

`clsPartitionAnalyzer`
AggregationAnalysisInitialized (clsPartitionAnalyzer)

The **AggregationAnalysisInitialized** property of an object of **ClassType clsPartitionAnalyzer** indicates the status of the last invocation of the **InitializeDesign** method.

**Data Type**

Boolean

**Access**

Read-only

**Remarks**

The **AggregationAnalysisInitialized** property returns True if the **InitializeDesign** method was called successfully, False otherwise.

**Example**

Use the following code to check the initialization status of an object of **ClassType clsPartitionAnalyzer**. For more information, see **InitializeDesign**.

' Assume the existence of an object (dsoPartAnalyzer) of ClassType ' clsPartitionAnalyzer.
If dsoPartAnalyzer.AggregationAnalysisInitialized Then
    Debug.Print "Initialization OK"
Else
    Debug.Print "Not Initialized"
End If

**See Also**

[clsAggregation](#)
clsPartition

clsPartitionAnalyzer

InitializeDesign
Parent (clsPartitionAnalyzer)

The Parent property of an object of ClassType clsPartitionAnalyzer contains a reference to the parent MDStore (ClassType clsPartition) object.

Data Type

MDStore

Access

Read-only

Example

Use the following code to refer to the parent object of the partition analyzer. For more information, see InitializeDesign.

' Assume the existence of an object (dsoPartAnalyzer) of ClassType ' clsPartitonAnalyzer.
' Print the name of the clsPartitionAnalyzer's parent partition.
Dim objParent As MDStore
Set objParent = dsoPartAnalyzer.Parent
Debug.Print objParent.Name

See Also

clsAggregation
clsPartition
clsPartitionAnalyzer
**clsPartitionDimension**

An object of **ClassType clsPartitionDimension** is used to maintain the **dimension** objects that a **partition** object contains. It provides a specific implementation of the Decision Support Objects (DSO) **Dimension** interface for dimensions associated with a specific partition. An object of **ClassType clsPartitionDimension** object provides collections and properties through the **Dimension** interface. There are no methods associated with an object of **ClassType clsPartitionDimension**.

**Remarks**

A partition cannot have fewer dimensions than its parent cube.

**Example**

Use the following code to reference an object of **ClassType clsPartitionDimension**:

```vbnet
' Assume an object (dsoCube) of ClassType clsCube exists.
Dim dsoPart As MDStore
Dim dsoPartDim As DSO.Dimension

' Retrieve the default partition.
Set dsoPart = dsoCube.MDStore(1)
```

**See Also**

- Collections, clsPartitionDimension
- Dimension Interface
- Properties, clsPartitionDimension
Collections, clsPartitionDimension

An object of ClassType clsPartitionDimension implements the following collections of the Dimension interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>Levels</td>
<td>The collection of level objects</td>
</tr>
</tbody>
</table>

See Also

clsPartitionDimension

Dimension Interface
Properties, clsPartitionDimension

An object of **ClassType clsPartitionDimension** implements the following properties of the **Dimension** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AggregationUsage</strong></td>
<td>Specifies how aggregations are designed for a dimension.</td>
</tr>
<tr>
<td><strong>AllowSiblingsWithSameName</strong></td>
<td>Indicates whether a dimension can contain members that have the same name.</td>
</tr>
<tr>
<td><strong>AreMemberKeysUnique</strong></td>
<td>Indicates whether member keys are unique for the dimension.</td>
</tr>
<tr>
<td><strong>AreMemberNamesUnique</strong></td>
<td>Indicates whether member names are unique for the dimension.</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type.</td>
</tr>
<tr>
<td><strong>DataMemberCaptionTemplate</strong></td>
<td>Contains a template string that is used to create captions for system-generated data members.</td>
</tr>
<tr>
<td><strong>DataSource</strong></td>
<td>Contains a reference to the data source object.</td>
</tr>
<tr>
<td><strong>DefaultMember</strong></td>
<td>Defines the default member of the dimension.</td>
</tr>
<tr>
<td><strong>DependsOnDimension</strong></td>
<td>Names a dimension on which the current dimension is dependent.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Contains the description of the dimension.</td>
</tr>
<tr>
<td><strong>DimensionType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of dimension.</td>
</tr>
<tr>
<td><strong>EnableRealTimeUpdates</strong></td>
<td>Indicates whether real-time updates are enabled for the dimension.</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>Contains the SQL FROM clause for a dimension.</td>
</tr>
<tr>
<td><strong>IsChanging</strong></td>
<td>Indicates whether members and/or levels are</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsReadWrite</td>
<td>Indicates whether dimension writebacks are available to clients with appropriate permissions.</td>
</tr>
<tr>
<td>IsShared</td>
<td>Indicates whether the dimension is shared among cubes.</td>
</tr>
<tr>
<td>IsTemporary</td>
<td>Indicates whether the dimension is temporary.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the dimension structure is valid.</td>
</tr>
<tr>
<td>IsVirtual</td>
<td>Indicates whether the dimension is virtual.</td>
</tr>
<tr>
<td>IsVisible</td>
<td>Indicates whether the dimension is visible to the client.</td>
</tr>
<tr>
<td>JoinClause</td>
<td>Contains the SQL JOIN clause for a dimension.</td>
</tr>
<tr>
<td>LastProcessed</td>
<td>Contains the date and time when the dimension was last processed.</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>A user-specified date. It is not used by DSO. This property can be used by client applications for their own uses.</td>
</tr>
<tr>
<td>MembersWithData</td>
<td>Determines which members in a dimension can have associated data in the fact table.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the dimension.</td>
</tr>
<tr>
<td>OrdinalPosition</td>
<td>Returns the ordinal position of the dimension object within its parent object's Dimensions collection.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a reference to the parent MDStore object.</td>
</tr>
<tr>
<td>SourceTable</td>
<td>Contains the name of the source table of the dimension.</td>
</tr>
<tr>
<td>SourceTableAlias</td>
<td>Returns the alias of the source table for the dimension.</td>
</tr>
<tr>
<td>SourceTableFilter</td>
<td>Restricts the members included in a dimension.</td>
</tr>
<tr>
<td>StorageMode</td>
<td>Determines the method of storing</td>
</tr>
</tbody>
</table>
dimension contents.

| SubClassType | Returns an enumeration constant that identifies the subclass type of the object. |

**See Also**

- clsPartitionDimension
- Dimension Interface
Analysis Services Programming

**clsPartitionLevel**

An object of ClassType `clsPartitionLevel` provides a specific implementation of the Decision Support Objects (DSO) Level interface for levels associated with a partition. It is used to maintain the levels objects a partition object contains.

An object of ClassType `clsPartitionLevel` provides collections and properties through the Level interface. There are no methods associated with this object.

**Remarks**

Levels describe the dimension hierarchy from the highest (most aggregated) level to the lowest (most detailed) level of data. The (All) level of a dimension is the top level of a dimension, and includes all the members of all the levels.

**Example**

Use the following code to reference a `clsPartitionLevel` object:

' Assume an object (dsoPart) of ClassType clsPartition exists.
Dim dsoLev As DSO.Level
' Retrieve the first level associated with the partition.
Set dsoLev = dsoPart.Levels(1)

**See Also**

Collections, clsPartitionLevel
Level Interface
Levels and Members
Properties, clsPartitionLevel
Slice Value
Collections, clsPartitionLevel

An object of ClassType clsPartitionLevel implements the following collection of the Level interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>MemberProperties</td>
<td>The collection of objects of ClassType clsMemberProperty</td>
</tr>
</tbody>
</table>

See Also

clsPartitionLevel
Level Interface
## Properties, clsPartitionLevel

An object of **ClassType clsPartitionLevel** implements the following properties of the **Level** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AreMemberKeysUnique</strong></td>
<td>Indicates whether the members of a level are uniquely identified by their member key column</td>
</tr>
<tr>
<td><strong>AreMemberNamesUnique</strong></td>
<td>Indicates whether the members of a level are uniquely identified by their member name column</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>ColumnSize</strong></td>
<td>The size (in bytes) of the data in the member key column of the level</td>
</tr>
<tr>
<td><strong>ColumnType</strong></td>
<td>The data type of the member key column of the level</td>
</tr>
<tr>
<td><strong>CustomRollUpColumn</strong></td>
<td>Contains the name of the column that contains member-specific rollup instructions</td>
</tr>
<tr>
<td><strong>CustomRollUpExpression</strong></td>
<td>Contains a Multidimensional Expressions (MDX) expression used to override the default rollup mode</td>
</tr>
<tr>
<td><strong>CustomRollUpPropertiesColumn</strong></td>
<td>Contains the name of the column that supplies cell properties for member-specific rollup instructions</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Contains the level description</td>
</tr>
<tr>
<td><strong>EnableAggregations</strong></td>
<td>Specifies whether aggregations are to be enabled for the level object</td>
</tr>
<tr>
<td><strong>EstimatedSize</strong></td>
<td>Contains the estimated number of members in a level</td>
</tr>
<tr>
<td><strong>FromClause</strong></td>
<td>Contains the SQL FROM clause for the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Grouping</strong></td>
<td>Indicates the type of grouping used by the Analysis server</td>
</tr>
<tr>
<td><strong>HideMemberIf</strong></td>
<td>Indicates whether a member should be hidden from client applications</td>
</tr>
<tr>
<td><strong>IsDisabled</strong></td>
<td>Indicates whether the level is disabled</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>Indicates whether the level is visible to client applications</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the level structure is valid</td>
</tr>
<tr>
<td><strong>JoinClause</strong></td>
<td>Contains the SQL JOIN clause for the level</td>
</tr>
<tr>
<td><strong>LevelNamingTemplate</strong></td>
<td>Defines how levels in a parent-child hierarchy are named</td>
</tr>
<tr>
<td><strong>LevelType</strong></td>
<td>Returns an enumeration constant that identifies the specific type of level</td>
</tr>
<tr>
<td><strong>MemberKeyColumn</strong></td>
<td>Returns the name of the column that contains member keys of the partition level</td>
</tr>
<tr>
<td><strong>MemberNameColumn</strong></td>
<td>Sets or returns the name of the column that contains member names</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Contains the name of the level</td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>Specifies the method to use when ordering the members of a level</td>
</tr>
<tr>
<td><strong>OrderingMemberProperty</strong></td>
<td>Specifies a member property used to determine the ordering of members</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the level in the Levels collection of the parent object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent dimension object</td>
</tr>
<tr>
<td><strong>ParentKeyColumn</strong></td>
<td>Identifies the parent of a member in a parent-child hierarchy</td>
</tr>
<tr>
<td><strong>RootMemberIf</strong></td>
<td>Determines how the root member or members of a parent-child hierarchy are</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SkippedLevelsColumn</strong></td>
<td>Identifies the column that holds the number of empty levels between a member and its parent</td>
</tr>
<tr>
<td><strong>SliceValue</strong></td>
<td>Contains the level member name used to define the partition slice</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
<tr>
<td><strong>UnaryOperatorColumn</strong></td>
<td>Contains the name of a column that stores mathematical operators serving as member-specific rollup instructions for the level</td>
</tr>
</tbody>
</table>

**See Also**

- `clsPartitionLevel`
- `Level Interface`
- `MemberKeyColumn`
clsPartitionMeasure

An object of ClassType clsPartitionMeasure provides a specific implementation of the Decision Support Objects (DSO) Measure interface. This object is used to maintain the measure objects a partition object contains.

An object of ClassType clsPartitionMeasure provides collections and properties through the Measure interface. There are no methods associated with an object of ClassType clsPartitionMeasure.

Remarks

Measures are the quantitative, numerical columns from the fact table of a cube. When a cube is processed, all of the measures, except for those based on the DistinctCount aggregate function, are aggregated across the dimensions in the cube.

Example

Use the following code to reference an object of ClassType clsPartitionMeasure:

' Assume an object (dsoPartition) of ClassType clsPartition exists.
Dim dsoPartMeasure As DSO.Measure
' Retrieve the first measure associated with the partition.
Set dsoPartMeasure = dsoPartition.Measures(1)

See Also

Measure Interface
Measures
Object Architecture
Partitions
Collections, clsPartitionMeasure

An object of ClassType clsPartitionMeasure implements the following collection of the Measure interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

See Also

- clsPartitionMeasure
- Measure Interface
Properties, clsPartitionMeasure

An object of **ClassType clsPartitionMeasure** implements the following properties of the Measure interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AggregateFunction</strong></td>
<td>Contains a value that corresponds to the type of aggregate function used for a measure</td>
</tr>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Contains the measure description</td>
</tr>
<tr>
<td><strong>FormatString</strong></td>
<td>Contains the format used to display the measure values</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the measure structure is valid</td>
</tr>
<tr>
<td><strong>IsVisible</strong></td>
<td>Indicates whether the measure is visible to the client application</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Contains the measure name</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the measure in the Measures collection of the parent object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent clsPartition object</td>
</tr>
<tr>
<td><strong>SourceColumn</strong></td>
<td>Contains the name of the measure column in the aggregated fact table</td>
</tr>
<tr>
<td><strong>SourceColumnType</strong></td>
<td>Returns a Microsoft® ActiveX® Data Objects (ADO) DB enumeration constant that identifies the data type of the column specified by the SourceColumn property</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

**See Also**

clsPartitionMeasure
Measure Interface
### clsRoleCommand

An object of **ClassType clsRoleCommand** provides a specific implementation of the **Command** interface. These objects provide collections and properties through the **Command** interface. There are no methods associated with an object of **ClassType clsRoleCommand**.

### Remarks

An object of **ClassType clsRoleCommand** encapsulates a user-defined command that is automatically executed on the Microsoft® SQL Server™ 2000 Analysis Services client computer when a cube is accessed by members of the specified role. You add a command to a role by adding it to the role's **Commands** collection. Such commands include calculated members, named sets, library references, and others.

For example, you may want to grant access to a calculated member called SalesBonus to members of the Manager role but not to members of the SalesPerson role. The command that builds the SalesBonus calculated member is automatically executed for members of the Manager role when they access a cube that contains that role command.

### Example

Use the following code to create an object of **ClassType clsRoleCommand**:

```vbscript
'Assume an object (dsoDB) of ClassType clsDatabase exists
Dim dsoRole As DSO.Role  'Role
Dim dsoCmd As DSO.Command  'Command

Set dsoRole = dsoDB.Roles(1)
Set dsoCmd = dsoRole.Commands.AddNew("RoleCmd1")
```

### See Also
Collections, clsRoleCommand
Command Interface
Commands
Properties, clsRoleCommand
Collections, clsRoleCommand

An object of **ClassType clsRoleCommand** implements the following collection of the **Command** interface.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CustomProperties</strong></td>
<td>The collection of user-defined properties</td>
</tr>
</tbody>
</table>

**See Also**

- clsRoleCommand
- Command Interface
Properties, clsRoleCommand

An object of **ClassType clsRoleCommand** implements the following properties of the **Command** interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassType</strong></td>
<td>Returns an enumeration constant that identifies the specific object type</td>
</tr>
<tr>
<td><strong>CommandType</strong></td>
<td>Returns an enumeration constant that identifies the specific command option</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Contains the description of the role command</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Indicates whether the Name and Statement properties are empty and whether the command object belongs to a collection</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Contains the name of the role command</td>
</tr>
<tr>
<td><strong>OrdinalPosition</strong></td>
<td>Returns the ordinal position of the command object in Commands collection of the parent role object</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>Returns a reference to the parent object, using the MDStore interface of the parent object</td>
</tr>
<tr>
<td><strong>ParentObject</strong></td>
<td>Returns a reference to the parent object, using the default interface of the parent object</td>
</tr>
<tr>
<td><strong>Statement</strong></td>
<td>Contains the text of the role command statement, in Multidimensional Expressions (MDX)</td>
</tr>
<tr>
<td><strong>SubClassType</strong></td>
<td>Returns an enumeration constant that identifies the subclass type of the object</td>
</tr>
</tbody>
</table>

**See Also**

- [clsRoleCommand](#)
- [Command Interface](#)
Analysis Services Programming

**clsServer**

An object of **ClassType clsServer** provides methods and properties that enable you to control an Analysis server. This object is the root of the Decision Support Objects (DSO) object model tree that specifies the databases, cubes, and user roles managed by the server. With an object of **ClassType clsServer** you can:

- Connect to a computer where the Analysis server service (MSSQLServerOLAPService) is running.

- Start and stop the server.

- Create and manage objects that define multidimensional data structures.

An object of **ClassType clsServer** provides collections, methods, and properties through its own internal interface.

**Examples**

**A. Creating and Initializing a Server**

Use the following code to create and initialize a server. You can use **LocalHost** to specify the Analysis server running on the same computer as your DSO application.

'Create instance of server and connect
Public dsoServer As DSO.Server
Set dsoServer = New DSO.Server
'ServerName is the Windows NT 4.0 Server or Windows 2000 Server or
'where the Analysis service is loaded and running.
'An error is raised if the connection attempt fails
dsoServer.Connect "ServerName"

This example accomplishes the same result:
DsoServer = New DSO.Server
dsoServer.Name = "ServerName"
dsoServer.Connect

**B. Creating and Connecting to a Server**

The following example shows how to create an instance of a DSO object of **ClassType clsServer** and connect to an Analysis server:

```vbs
Public Sub ConnectToServer()
    Dim dsoServer As DSO.Server

    On Error GoTo ErrHandler

    ' Initialize server.
    Set dsoServer = New DSO.Server

    ' Connect to the local Analysis server.
    ' If a connection cannot be made, an error is raised.
    dsoServer.Connect "LocalHost"

    ' Print server properties to the Debug window.
    With dsoServer
        Debug.Print "Server Properties --------------------------"
        Debug.Print "Name: " & .Name
        Debug.Print "Description: " & .Description
        Debug.Print "ConnectTimeout: " & .ConnectTimeout
        Debug.Print "LockTimeout: " & .LockTimeout
    End With

    ' Close connection to server.
    dsoServer.CloseServer
```
ExitRoutine:
    Set dsoServer = Nothing
Exit Sub

ErrHandler:
    Debug.Print "Error connecting to server:"
End Sub

See Also

Collections, clsServer
Methods, clsServer
Properties, clsServer
Collections, clsServer

An object of ClassType clsServer implements the following collections.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomProperties</td>
<td>The collection of user-defined properties</td>
</tr>
<tr>
<td>MDStores</td>
<td>The collection of databases that define the multidimensional data managed by the server</td>
</tr>
</tbody>
</table>

Access
Read/write

See Also
clsServer
Methods, clsServer

An object of ClassType clsServer implements the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseServer</td>
<td>Releases all server resources and sets the State property of the object to stateUnknown</td>
</tr>
<tr>
<td>Connect</td>
<td>Connects to the Analysis server service (MSSQLServerOLAPService)</td>
</tr>
<tr>
<td>CreateObject</td>
<td>Creates an object</td>
</tr>
<tr>
<td>LockObject</td>
<td>Locks a clsServer object</td>
</tr>
<tr>
<td>Refresh</td>
<td>Reads all current meta data from the repository and modifies all objects in the object model of a session to match the current repository state</td>
</tr>
<tr>
<td>UnlockAllObjects</td>
<td>Removes all locks issued by the current session from objects in the object hierarchy of the clsServer object</td>
</tr>
<tr>
<td>UnlockObject</td>
<td>Removes a lock from the clsServer object</td>
</tr>
<tr>
<td>Update</td>
<td>Updates an object definition in the meta data repository</td>
</tr>
</tbody>
</table>

See Also

clsServer
CloseServer (clsServer)

The CloseServer method of an object of ClassType clsServer releases all server resources and sets the State property of the object to stateUnknown.

Syntax

dsoServer.CloseServer
dsoServer

The Decision Support Objects (DSO) server object to be closed.

Example

The following example assumes that the server object exists and is connected to an Analysis server. The example closes a previously created connection:

dsoServer.CloseServer

See Also

clsServer
Connect (clsServer)

The **Connect** method of an object of **ClassType clsServer** connects to the Analysis server service (MSSQLServerOLAPService).

**Syntax**

```vbnet
dsoServer.Connect([ByVal ServerName As String])
```

- **dsoServer**
  A Decision Support Objects (DSO) server object.

- **ServerName**
  The name of the computer on which the Analysis server is installed and running. You can use **LocalHost** to specify the Analysis server running on the same computer as your DSO application.

**Remarks**

The **Connect** method sets the name of the server object. If you need to reconnect to the same server, do not specify the name on subsequent executions of the method on the same object.

**IMPORTANT** You cannot use an Internet Protocol (IP) address as the **ServerName** parameter to connect to a server. You must use the network name of the computer that hosts the Analysis server.

**Example**

Use the following code example to connect to the Analysis server:

```vbnet
Public dsoServer As DSO.Server
Set dsoServer = New DSO.Server
'Assume "Server1" to be the name of the computer
'where the Analysis server service is installed and running.
dsoServer.Connect "Server1" 'Error is raised if unable to connect
See Also

clsServer
CreateObject (clsServer)

The CreateObject method of an object of ClassType clsServer creates and returns an object of the specified ClassType and SubClassType.

Syntax

```
objRet = object.CreateObject(ObjectType As ClassTypes, [SubClassType As SubClassTypes = sbclsRegular])
```

<table>
<thead>
<tr>
<th>objRet</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Decision Support Objects (DSO) object of the type to be created.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>objectr</th>
</tr>
</thead>
<tbody>
<tr>
<td>An object of ClassType clsServer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ObjectType</th>
</tr>
</thead>
<tbody>
<tr>
<td>The class type of the object to be created. A member of the ClassTypes enumeration.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SubClassType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. The subclass type of the object to be created. A member of the SubClassTypes enumeration. Default is sbclsRegular.</td>
</tr>
</tbody>
</table>

Remarks

When you are using DSO to construct major objects such as cubes, partitions, dimensions, and so on, use the AddNew method instead. The AddNew method creates an object of the appropriate ClassType in a collection and initializes its Name and Parent properties.

Example

Use the following code to create an object of type clsCube:

' Assume objServer is a server object and objDB is a database object
Dim objCube as New DSO.MDStore
Set objCube = objServer.CreateObject(clsCube)
objCube.Name = "NewCube"
objDB.MDStores.Add objCube

See Also

AddNew

ClassTypes

clsServer

MDStore Interface

SubClassTypes
LockObject (clsServer)

The LockObject method of an object of ClassType clsServer locks the clsServer object to prevent multiple users from concurrently changing the object.

Syntax

```vba
object.LockObject(ByVal LockType As OlapLockTypes, ByVal LockDescription As String)
```

object

An object of ClassType clsServer.

LockType

One of the constants defined in the OlapLockTypes enumeration. For more information, see OlapLockTypes.

sLockDescription

A string containing the description of the lock, available to other applications attempting to obtain a lock.

Remarks

For more information about object locking, see LockObject.

See Also

clsServer
Refresh (clsServer)

The Refresh method of an object of ClassType clsServer reads all current metadata from the repository and modifies all objects in the session's object model to match the current repository state.

Syntax

object.Refresh

object

An object of ClassType clsServer.

Remarks

All objects in the session's object model are refreshed. This includes objects whose properties have not yet been saved to the repository, as well as objects whose properties in the repository have been changed by other users.

Note  Objects that are locked are not refreshed.

Example

The following example invokes the Refresh method:

' Assume dsoServer has already been connected to Analysis Services. dsoServer.Refresh

See Also

clsServer
UnlockAllObjects (clsServer)

The UnlockAllObjects method of an object of ClassType clsServer removes all locks issued by the current session from objects in the server's object model.

Syntax

\[ bRet = \text{object.UnlockAllObjects} \]

\( bRet \)

A Boolean variable. This value is set to True if the method is successful and False otherwise.

\( \text{object} \)

An object of ClassType clsServer.

Example

The following example removes all current locks:

' Assume dsoServer is connected to Analysis Services.

dsoServer.UnlockAllObjects

See Also

clsServer
UnlockObject (clsServer)

The UnlockObject method of an object of ClassType clsServer removes a lock from the clsServer object.

Syntax

object.UnlockObject

objServer

An object of ClassType clsServer.

See Also

clsServer
Update (clsServer)

The **Update** method of an object of *ClassType clsServer* updates an object definition in the meta data repository. This method must be called after any attribute for the server object has been changed, or the affected meta data for the changes will not be persistent past the server object scope.

**Syntax**

```vbnet
object.Update

object
```

An object of *ClassType clsServer*.

**Remarks**

Whenever a write-enabled server property is changed, its meta data in the repository must be updated with this method. Subordinate objects are automatically updated when you update an object that contains other objects, such as a cube that contains partitions, dimensions, and measures.

**Example**

The following example invokes the **Update** method after setting the server **timeout** property:

```vbnet
' Assume dsoServer is connected to Analysis Services.
dsoServer.Timeout = 30 ' = 30 seconds.
dsoServer.Update
```

**See Also**

[clsServer](#)
Properties, clsServer

An object of **ClassType clsServer** implements the following properties. The table also shows whether the property is read/write (R/W) or read-only (R).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassType</td>
<td>Returns an enumeration constant that identifies the specific class type</td>
<td>R</td>
</tr>
<tr>
<td>ConnectTimeout</td>
<td>The amount of time until a connection to an Analysis server fails due to timeout</td>
<td>R/W</td>
</tr>
<tr>
<td>Description</td>
<td>The description of the server object</td>
<td>R/W</td>
</tr>
<tr>
<td>Edition</td>
<td>The installed edition of Microsoft® SQL Server™ 2000 Analysis Services</td>
<td>R</td>
</tr>
<tr>
<td>IsValid</td>
<td>Indicates whether the server settings are valid</td>
<td>R</td>
</tr>
<tr>
<td>LockTimeout</td>
<td>The amount of time until a lock request fails due to timeout</td>
<td>R/W</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the server</td>
<td>R/W</td>
</tr>
<tr>
<td>Parent</td>
<td>A reference to the <strong>App</strong> object</td>
<td>R</td>
</tr>
<tr>
<td>ProcessingLogFileName</td>
<td>A UNC path to a file for logging status messages occurring during processing</td>
<td>R/W</td>
</tr>
<tr>
<td>ServiceState</td>
<td>Contains the state of the Analysis server service (MSSQLServerOLAPService)</td>
<td>R/W</td>
</tr>
<tr>
<td>State</td>
<td>Returns an enumeration constant indicating the status of the connection to the Analysis server</td>
<td>R</td>
</tr>
<tr>
<td>Timeout</td>
<td>The amount of time until a timeout error occurs during processing</td>
<td>R/W</td>
</tr>
<tr>
<td>Version</td>
<td>The version of the Analysis server</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**See Also**

clsServer
ClassType (clsServer)

The ClassType property of a server object returns an enumeration constant that identifies the specific class type.

Data Type

Integer representing a value from the ClassTypes enumeration. For more information, see ClassTypes.

Access

Read-only

Remarks

The ClassType property returns the value clsServer for all server objects.

See Also

clsServer
ConnectTimeout (clsServer)

The `ConnectTimeout` property of an object of `ClassType clsServer` sets or returns the maximum amount of time an idle connection to an Analysis server is maintained before the server is considered to have timed out.

**Data Type**

Long

**Access**

Read/write

**Remarks**

The default value is zero (0) seconds; that is, the server connection never times out. The maximum allowed value is one million (1,000,000) seconds, approximately 11 days and 14 hours. To have the server connected indefinitely, set the `ConnectTimeout` property to zero (0).

**Example**

Use the following code to set the `ConnectTimeout` property for a server. You must call the `Update` method for your changes to take effect.

```vbnet
Dim dsoS As New DSO.Server
dsoS.Connect "LocalHost"     ' server name
dsoS.ConnectTimeout = 18000  ' timeout value, in seconds
dsoS.Update
```

**See Also**

`clsServer`
Description (clsServer)

The Description property of an object of ClassType clsServer contains the server description.

Data Type
String

Access
Read/write

Example
The following code example prints the Description property of an object of ClassType clsServer to the Debug window.

Dim dsoServer As New DSO.Server

' Connect to the local Analysis server.
dsoServer.Connect "LocalHost"

' Print the Description property to the Debug window.
Debug.Print dsoServer.Description

See Also
clsServer
Edition (clsServer)

The **Edition** property of an object of **ClassType clsServer** identifies which edition of Microsoft® SQL Server™ 2000 Analysis Services is installed.

**Data Type**

[OlapEditions](#)

**Access**

Read-only

**Remarks**

The functionality of Analysis Services varies depending on the edition installed.

**Example**

The following code example checks the **Edition** property of a **clsServer** object to determine feature support.

Dim dsoServer As New DSO.Server

' Connect to the local Analysis server.
dsoServer.Connect "LocalHost"

' Check the Edition property.
Select Case dsoServer.Edition
    Case olapEditionUnlimited
        ' Insert code for Enterprise Edition features.
    Case olapEditionPivotOnly
        ' Reserved for future use.
    Case olapEditionNoPartitions
' Insert code for Standard Edition features.

Case olapEditionError
    ' An error occurred while retrieving this information.
End Select

See Also

clsServer
IsValid (clsServer)

The IsValid property of an object of ClassType clsServer indicates whether the server name is valid.

Data Type

Boolean

Access

Read-only

Remarks

The validity check on the server object verifies that the server has a valid name.

See Also

clsServer
Analysis Services Programming

**LockTimeout (clsServer)**

The `LockTimeout` property of an object of `ClassType clsServer` sets or returns the amount of time until a lock request fails due to timeout.

**Data Type**

Long

**Access**

Read/write

**Remarks**

The value of this property must not be less than zero. The default is 20 seconds. For more information about locking a server, see `LockObject`, `UnlockObject`, and `UnlockAllObjects`.

**See Also**

`clsServer`
Analysis Services Programming

**Name (clsServer)**

The `Name` property of an object of `ClassType clsServer` contains the name of the server object.

**Data Type**

String

**Access**

Read/write (read-only after the object has been named)

**Remarks**

You cannot use an Internet Protocol (IP) address for the `Name` property to connect to a server. You must use the network name of the computer that hosts the Analysis server.

**See Also**

`clsServer`
Parent (clsServer)

The **Parent** property of an object of **ClassType clsServer** contains a reference to the **App** object of the application.

**Data Type**

Object

**Access**

Read-only

**Remarks**

In an application using Decision Support Objects (DSO), an object of **ClassType clsServer** is the root of the object model tree. The Microsoft® Visual Basic® **App** object is returned as its parent for convenience.

**See Also**

[clsServer](#)
**ProcessingLogFileName (clsServer)**

The **ProcessingLogFileName** property of an object of **ClassType clsServer** stores a UNC path to a file for logging status messages from processing events.

**Data Type**

String

**Access**

Read/write

**Remarks**

The log file is a text file that contains status and error messages that are raised during mining model, cube, partition, or dimension processing events. The file also logs dimension key errors (that is, errors that are raised when rows in the fact table do not correspond to rows in the dimension source table). However, the details of these errors are logged in the file specified by the **ProcessingKeyErrorLogFileName** property of the parent **MDStore** interface.

**See Also**

- clsServer
- ProcessingKeyErrorLogFileName
The **ServiceState** property of an object of `ClassType clsServer` contains the execution state of the Analysis server service (MSSQLServerOLAPService).

**Data Type**
Long

**Values**
The values of this property are different depending on whether the property is being read or set.

The following values are returned when reading this property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_CONTINUE_PENDING</td>
<td>A previous request to continue a paused service is pending.</td>
</tr>
<tr>
<td>SERVICE_PAUSE_PENDING</td>
<td>A previous request to pause a running service is pending.</td>
</tr>
<tr>
<td>SERVICE_PAUSED</td>
<td>The service is paused.</td>
</tr>
<tr>
<td>SERVICE_RUNNING</td>
<td>The service is running.</td>
</tr>
<tr>
<td>SERVICE_START_PENDING</td>
<td>The service is starting.</td>
</tr>
<tr>
<td>SERVICE_STOP_PENDING</td>
<td>The service is stopping.</td>
</tr>
<tr>
<td>SERVICE_STOPPED</td>
<td>The service is not running.</td>
</tr>
</tbody>
</table>

The following table describes the values used to control the Analysis server.

<table>
<thead>
<tr>
<th>Value</th>
<th>Requested action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_PAUSED</td>
<td>Pause the service.</td>
</tr>
<tr>
<td>SERVICE_RUNNING</td>
<td>Start the service if stopped or paused.</td>
</tr>
<tr>
<td>SERVICE_STOP</td>
<td>Stop the service.</td>
</tr>
</tbody>
</table>
**Access**
Read/write

**Remarks**
Read the property to query the status of the service. To change the execution state of the service, set the property to a value. Decision Support Objects (DSO) partially implements the service control functions of the Microsoft® Win32® API.

If a requested action cannot be completed, such as attempting to pause a service that is not running, or the request times out (within 60 seconds), an error occurs.

**Example**
Use the following code to set the execution state of MSSQLServerOLAPService:

' Analysis server service control constants  
Const OLAP_SERVICE_RUNNING = &H4  
Const OLAP_SERVICE_PAUSED = &H7  
Const OLAP_SERVICE_STOP = &H1

' Analysis server status and error return constants  
Const SERVICE_CONTINUE_PENDING = &H5  
Const SERVICE_PAUSE_PENDING = &H6  
Const SERVICE_PAUSED = &H7  
Const SERVICE_RUNNING = &H4  
Const SERVICE_START_PENDING = &H2  
Const SERVICE_STOP_PENDING = &H3  
Const SERVICE_STOPPED = &H1

' Additional error return constants  
Const SERVICE_ACCEPT_PAUSE_CONTINUE = &H2  
Const SERVICE_ACCEPT_SHUTDOWN = &H4  
Const SERVICE_ACCEPT_STOP = &H1
Const SERVICE_ACTIVE = &H1
Const SERVICE_CHANGE_CONFIG = &H2
Const SERVICE_CONTROL_CONTINUE = &H3
Const SERVICE_CONTROL_INTERROGATE = &H4
Const SERVICE_CONTROL_PAUSE = &H2
Const SERVICE_CONTROL_SHUTDOWN = &H5
Const SERVICE_CONTROL_STOP = &H1
Const SERVICE_ENUMERATE_DEPENDENTS = &H8
Const SERVICE_INACTIVE = &H2
Const SERVICE_INTERROGATE = &H80
Const SERVICE_NO_CHANGE = &HFFFF
Const SERVICE_PAUSE_CONTINUE = &H40
Const SERVICE_QUERY_CONFIG = &H1
Const SERVICE_QUERY_STATUS = &H4
Const SERVICE_STATE_ALL = (SERVICE_ACTIVE Or SERVICE_
Const SERVICE_USER_DEFINED_CONTROL = &H100

Const SERVICE_WAIT_MAX_SECONDS As Integer = 30
'
'=====================================================================================================
'
OlapServiceControl function
' Returns True or False
' Calling parameters:
'   - objServer is an object of ClassType clsServer
'     that has been created and initialized
'   - iCmdReq is one of the Analysis server service
'     control constants
'   - lngStatus receives the status (one of the Analysis
'     server status constants)
'   - lngErr receives status if function fails (one of the Analysis
'     server status constants or one of the additional error constants)

Friend Function OlapServiceControl(objServer As Object, _
    ByVal iCmdReq As Integer, _
ByRef lngStatus As Long, _
ByRef lngErr As Long) As Boolean

Dim bRet As Boolean
Dim lngSrvStat As Long
Dim lngControlCmd As Long

lngSrvStat = objServer.ServiceState
bRet = False
lngControlCmd = iCmdReq
lngErr = 0

On Error GoTo Err_State

Select Case iCmdReq
    ' Caller wants to start the server
    Case SERVICE_RUNNING
        ' Check the current server status
        Select Case lngSrvStat
            ' If it is already running, return True
            Case SERVICE_RUNNING
                bRet = True
            Case SERVICE_PAUSED, SERVICE_STOPPED
                objServer.ServiceState = lngControlCmd
                bRet = True
        End Select
    End Case
    Case SERVICE_PAUSED   ' Caller wants to pause the server
        Select Case lngSrvStat
            Case SERVICE_PAUSED
                bRet = True
            Case SERVICE_RUNNING
                objServer.ServiceState = lngControlCmd
                bRet = True
        End Select
' Trying to pause a stopped service
' raises an error from the Server object.
Case SERVICE_STOPPED
  bRet = False
End Select

Case SERVICE_STOPPED  ' Caller wants to stop the server
Select Case lngSrvStat
  Case SERVICE_STOPPED
    bRet = True
  Case SERVICE_RUNNING
    objServer.ServiceState = lngControlCmd
    bRet = True
  ' Trying to stop a paused service
  ' raises an error from the Server object.
  Case SERVICE_PAUSED
    bRet = False
End Select
End Select

' Put the current state of the service into lngStatus
lngStatus = objServer.ServiceState

OlapServiceControl = bRet

Exit Function

Err_State:
' Catch the error returned by the server object
' Some reasons that can cause an error:
' Server object unable to contact service control
' manager or Analysis service application
' Service does not respond to state change
'request within 60 seconds
'An invalid request is sent to the service (for example,
'trying to pause a stopped service)

lngStatus = objServer.ServiceState
lngErr = Err.Number
OlapServiceControl = False

End Function

See Also
clsServer
State (clsServer)

The State property of an object of ClassType clsServer returns an enumeration constant that indicates the status of the connection to the Analysis server.

Data Type

ServerStates

Access

Read-only

See Also

clsServer
Analysis Services Programming

**Timeout (clsServer)**

The **Timeout** property of an object of **ClassType clsServer** sets or returns the maximum amount of time between reports from the Analysis server before the server is considered to have timed out.

**Data Type**

Long

**Access**

Read/write

**Remarks**

The default value is 3600 seconds (1 hour). Under some conditions, the server may encounter long wait times, for example, during the processing of large cubes or during queries to a source database. If you experience timeout errors under such conditions, you can increase the default value. The maximum allowed value is one million (1,000,000) seconds. This is approximately 11 days and 14 hours. To have the server wait indefinitely, set the **Timeout** property to zero (0).

**Example**

Use the following code to set the **Timeout** property for a server. You must call the **Update** method for your changes to take effect.

```vbs
Dim dsoS As New DSO.Server

' Connect to the local Analysis server.
  dsoServer.Connect "LocalHost"

' Set the timeout to 4 hours.
```
dsoServer.Timeout = 14400
' Update the Analysis server.
dsoServer.Update

' Close the connection to the Analysis server.
dsoServer.CloseServer

See Also

clsServer
Update
Version (clsServer)

The Version property of an object of ClassType clsServer returns a string representing the version information of the Analysis server to which the object is connected.

Data Type

String

Access

Read-only

Remarks

The expected return values are listed in the following table.

<table>
<thead>
<tr>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft® SQL Server™ 7.0 OLAP Services</td>
<td>7.0</td>
</tr>
<tr>
<td>SQL Server 2000 Analysis Services</td>
<td>8.0</td>
</tr>
</tbody>
</table>

See Also

clsServer
**Property Object**

Use the **Property** object to save user-defined items to a **CustomProperties** collection. You can define **Property** objects and add them to the **CustomProperties** collection of any Decision Support Objects (DSO) object to store information you want to associate with the DSO object. You provide a name, value, and data type for each **Property** object.

**Remarks**

The **Property** object has properties, but no collections or methods. The **Property** object is unlike other DSO objects in that it does not implement any of the interfaces, methods, properties, or collections of other DSO objects.

**Examples**

**Creating a New Custom Property**

Use the following code to create a new custom property:

'Assume an object of ClassType clsDimension exists.
'Add a custom property.
Dim dsoProp As DSO.Property
Set dsoProp = dsoDim.CustomProperties.Add(55, "Age", vbInteger)

'Retrieve custom property values.
Dim dsoProp2 As DSO.Property
Set dsoProp2 = dsoDim.CustomProperties(1)
Debug.Print dsoProp2.Name, dsoProp2.Value

**See Also**

[CustomProperties](#)

[Properties, Property Object](#)
Properties, Property Object

A Property object implements the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataType</strong></td>
<td>The Microsoft® Visual Basic® data type</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The name of the Property object</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>The value of the Property object</td>
</tr>
</tbody>
</table>

Access

Read/write

See Also

Property Object
DataType (Property Object)

The **DataType** property of a **Property** object contains the Microsoft® Visual Basic® data type of the custom property defined by the **Property** object.

**Data Type**

**VBA.VbVarType**

**Access**

Read/write

**Remarks**

For more information about the **VBA.VbVarType** enumeration, see the Visual Basic documentation.

**See Also**

[CustomProperties](#)

[Property Object](#)
Name (Property Object)

The Name property of a Property object contains the unique user-assigned name of the custom property defined by the Property object.

Data Type
String

Access
Read/write

See Also
CustomProperties
Property Object
Value (Property Object)

The Value property of a Property object can contain any value that is valid for the DataType defined for the Property object.

Data Type

Variant

Access

Read/write

See Also

CustomProperties

Property Object
Analysis Services Programming
## Enumerations

The Decision Support Objects (DSO) object model provides a number of enumerations. The following table lists the public enumerated types available through DSO. Click the name of an enumeration for a more detailed description.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregatesTypes</td>
<td>Enumerates values for the <strong>AggregateFunction</strong> property</td>
</tr>
<tr>
<td>ClassTypes</td>
<td>Enumerates values for the <strong>ClassType</strong> property</td>
</tr>
<tr>
<td>CloneOptions</td>
<td>Enumerates options for the <strong>Clone</strong> method</td>
</tr>
<tr>
<td>CommandTypes</td>
<td>Enumerates values for the <strong>CommandType</strong> property</td>
</tr>
<tr>
<td>DimensionAggUsageTypes</td>
<td>Enumerates values for the <strong>AggregationUsage</strong> property</td>
</tr>
<tr>
<td>DimensionTypes</td>
<td>Enumerates values for the <strong>DimensionType</strong> property</td>
</tr>
<tr>
<td>ErrorCodes</td>
<td>Enumerates error codes</td>
</tr>
<tr>
<td>GroupingValues</td>
<td>Enumerates options for level groups</td>
</tr>
<tr>
<td>HideIfValues</td>
<td>Enumerates options for hidden level members</td>
</tr>
<tr>
<td>LanguageValues</td>
<td>Enumerates the <strong>Language</strong> property of member properties</td>
</tr>
<tr>
<td>LevelTypes</td>
<td>Enumerates values for the <strong>LevelType</strong> property</td>
</tr>
<tr>
<td>MembersWithDataValues</td>
<td>Enumerates values for the <strong>MembersWithData</strong> property</td>
</tr>
<tr>
<td>OlapEditions</td>
<td>Enumerates values for the <strong>Edition</strong> property</td>
</tr>
<tr>
<td>OlapLockTypes</td>
<td>Enumerates values for the <strong>LockObject</strong> method</td>
</tr>
<tr>
<td>OlapStateTypes</td>
<td>Enumerates values for the <strong>State</strong> property</td>
</tr>
<tr>
<td>OlapStorageModes</td>
<td>Enumerates values for the <strong>OlapMode</strong> property</td>
</tr>
<tr>
<td>OrderTypes</td>
<td>Enumerates values for the <strong>Ordering</strong> property</td>
</tr>
<tr>
<td>ProcessOptimizationModes</td>
<td>Enumerates values for the <strong>ProcessOptimizationMode</strong> property</td>
</tr>
<tr>
<td>ProcessTypes</td>
<td>Enumerates values for the <strong>Process</strong> method</td>
</tr>
<tr>
<td><strong>PropertyType</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Enumerates the values used in the <strong>PropertyType</strong> property</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RootIfValues</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerates values for the <strong>RootMemberIf</strong> property</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ServerStates</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerates values for the <strong>State</strong> property</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>StorageModeValues</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerates values for the <strong>StorageMode</strong> property</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SubClassTypes</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerates values for the <strong>SubClassType</strong> property</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

**Using the ClassTypes Enumeration**

The following code uses the **ClassTypes** enumeration to retrieve the class type of an object and determine whether the object is a cube, a virtual cube, or some other object:

```vba
' Assume that the object dsoServer of ClassType clsServer exists.
Dim dsoDB as MDStore
Dim dsoCube as MDStore
Dim CubeCounter as Integer

Set dsoDB = dsoServer.MDStores(1)
For CubeCounter = 1 To dsoDB.MDStores.Count
    Set dsoCube = dsoDB.MDStores(CubeCounter)
    Debug.Print " Cube: " & dsoCube.Name
    If dsoCube.SubClassType = sbclsRegular Then
        Debug.Print " SubClassType: Regular"
        Debug.Print " SourceTable: " & dsoCube.SourceTable
    Else
        Debug.Print " SubClassType: Virtual"
    End If
Next CubeCounter
```
Analysis Services Programming

**AggregatesTypes**

Enumerates values for the `AggregateFunction` property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggCount</td>
<td>Uses the <strong>Count</strong> function for aggregation</td>
</tr>
<tr>
<td>aggDistinctCount</td>
<td>Uses the <strong>Distinct Count</strong> function for aggregation</td>
</tr>
<tr>
<td>aggMax</td>
<td>Uses the <strong>Max</strong> function for aggregation</td>
</tr>
<tr>
<td>aggMin</td>
<td>Uses the <strong>Min</strong> function for aggregation</td>
</tr>
<tr>
<td>aggSum</td>
<td>Uses the <strong>Sum</strong> function for aggregation</td>
</tr>
</tbody>
</table>
## ClassTypes

Enumerates values for the `ClassType` property used by objects in Decision Support Objects (DSO).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clsAggregation</code></td>
<td>Provides a specific implementation of the <code>MDStore</code> interface. Each instance is used to maintain a unique aggregation data store.</td>
</tr>
<tr>
<td><code>clsAggregationDimension</code></td>
<td>Provides a specific implementation of the <code>Dimension</code> interface. Each instance reviews the dimension collection of objects contained within an aggregation object.</td>
</tr>
<tr>
<td><code>clsAggregationLevel</code></td>
<td>Provides a specific implementation of the <code>Level</code> interface. Each instance is used to maintain the level objects within an aggregation dimension object.</td>
</tr>
<tr>
<td><code>clsAggregationMeasure</code></td>
<td>Provides a specific implementation of the <code>Measure</code> interface. Each instance is used to maintain the measure objects contained within an aggregation object.</td>
</tr>
<tr>
<td><code>clsCollection</code></td>
<td>Similar to a standard Microsoft® Visual Basic® collection; however, objects of <code>ClassType clsCollection</code> can contain only objects of the same type.</td>
</tr>
<tr>
<td><code>clsColumn</code></td>
<td>Objects of <code>ClassType clsColumn</code> are used to represent the structure of <code>clsMiningModel</code> objects.</td>
</tr>
<tr>
<td><code>clsCube</code></td>
<td>Provides a specific implementation of the <code>MDStore</code> interface. Objects of <code>ClassType clsCube</code> provide the primary logical unit for representing collections of multidimensional data.</td>
</tr>
<tr>
<td><code>clsCubeAnalyzer</code></td>
<td>An object that contains a single method used to</td>
</tr>
</tbody>
</table>
extract information from the query log. The query log stores the descriptions of queries executed on the Analysis server.

<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCubeCommand</td>
<td>Provides a specific implementation of the <strong>Command</strong> interface. Each instance encapsulates a user-defined command that is automatically executed at the client when the cube containing the command is accessed.</td>
</tr>
<tr>
<td>clsCubeDimension</td>
<td>Provides a specific implementation of the <strong>Dimension</strong> interface. Cube dimensions are associated with the dimensions (shared and private) of a database.</td>
</tr>
<tr>
<td>clsCubeLevel</td>
<td>Provides a specific implementation of the <strong>Level</strong> interface. Levels define the granularity of their parent dimension.</td>
</tr>
<tr>
<td>clsCubeMeasure</td>
<td>Provides a specific implementation of the <strong>Measure</strong> interface. A cube measure corresponds to a numerically valued column in a cube's fact table.</td>
</tr>
<tr>
<td>clsCubeRole</td>
<td>Provides a specific implementation of the <strong>Role</strong> interface. Objects of <strong>ClassType clsCubeRole</strong> are used to manage the permissions a set of users has when accessing a cube.</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>Provides a specific implementation of the <strong>MDStore</strong> interface. Databases contain cubes, which in turn contain partitions, which may in turn contain dimensions, levels, measures, and aggregations. A database may also contain virtual cubes.</td>
</tr>
<tr>
<td>clsDatabaseCommand</td>
<td>Provides a specific implementation of the <strong>Command</strong> interface. Each instance encapsulates a user-defined command that is automatically executed at the client when the cube containing the command is accessed.</td>
</tr>
<tr>
<td>clsDatabaseDimension</td>
<td>Provides a specific implementation of the <strong>Dimension</strong> interface. Database dimensions can</td>
</tr>
</tbody>
</table>
A shared database dimension can be associated with any number of cubes; however, a private dimension can be associated with only one cube.

<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsDatabaseLevel</td>
<td>Provides a specific implementation of the Level interface. Levels define the granularity of their parent dimension. When you add a dimension to a cube, it inherits whatever levels are defined for the database dimension.</td>
</tr>
<tr>
<td>clsDatabaseRole</td>
<td>Provides a specific implementation of the Role interface. Objects of ClassType clsDatabaseRole are used to manage the set of users who can access the database.</td>
</tr>
<tr>
<td>clsDataSource</td>
<td>Objects of ClassType clsDataSource are used to specify an external database that will be used as a source of data for Microsoft SQL Server™ 2000 Analysis Services databases, cubes, and partitions.</td>
</tr>
<tr>
<td>clsMemberProperty</td>
<td>Provides the ability to assign properties to level members.</td>
</tr>
<tr>
<td>clsMiningModel</td>
<td>Provides methods and properties that enable you to create and control data mining objects on the Analysis server.</td>
</tr>
<tr>
<td>clsMiningModelRole</td>
<td>Provides a specific implementation of the Role interface. Objects of ClassType clsMiningModelRole are used to manage the set of users who can access the data mining model.</td>
</tr>
<tr>
<td>clsPartition</td>
<td>Provides a specific implementation of the MDStore interface. Partitions are common to large data warehouses where massive amounts of data must be managed efficiently. Partitions enable you to segment your data in various storage modes and on various servers.</td>
</tr>
<tr>
<td>clsPartitionAnalyzer</td>
<td>Encapsulates an algorithm for automatically designing a set of aggregations in a partition.</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>Provides a specific implementation of the <strong>Dimension</strong> interface. Each instance is used to maintain the dimension objects contained within a partition.</td>
</tr>
<tr>
<td>clsPartitionLevel</td>
<td>Provides a specific implementation of the <strong>Level</strong> interface. Each instance is used to maintain the level objects that are contained within a partition dimension object.</td>
</tr>
<tr>
<td>clsPartitionMeasure</td>
<td>Provides a specific implementation of the <strong>Measure</strong> interface. Each instance is used to maintain the measure objects that are contained within a partition object.</td>
</tr>
<tr>
<td>clsRoleCommand</td>
<td>Provides a specific implementation of the <strong>Command</strong> interface. Each instance encapsulates a user-defined command that is automatically executed at the client when the cube containing the command is accessed.</td>
</tr>
<tr>
<td>clsServer</td>
<td>Provides methods and properties that enable you to control the Analysis server. The object is the root of the DSO object model tree.</td>
</tr>
</tbody>
</table>
CloneOptions

Enumerates options for the Clone method.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloneMajorChildren</td>
<td>Clones the values of properties and all major and minor objects contained in the source object's collections</td>
</tr>
<tr>
<td>cloneMinorChildren</td>
<td>Clones the values of the properties and the minor objects contained in the source object's collections</td>
</tr>
<tr>
<td>cloneObjectProperties</td>
<td>Clones the values of the properties of the source object</td>
</tr>
</tbody>
</table>

Note  The CustomProperties collection is always cloned, regardless of the CloneOption specified.
## CommandTypes

Enumerates values for the `CommandType` property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmdCreateAction</code></td>
<td>Defines one or more actions.</td>
</tr>
<tr>
<td><code>cmdCreateCellCalculation</code></td>
<td>Defines one or more calculated cells.</td>
</tr>
<tr>
<td><code>cmdCreateMember</code></td>
<td>Defines one or more calculated members.</td>
</tr>
<tr>
<td><code>cmdCreateSet</code></td>
<td>Defines one or more named sets of existing members.</td>
</tr>
<tr>
<td><code>cmdUnknown</code></td>
<td>Defines statements not included in any of the other command types in this table, such as DROP MEMBER statements or new statements that may be added to future versions.</td>
</tr>
<tr>
<td><code>cmdUseLibrary</code></td>
<td>Specifies DLLs that contain functions to be registered for use in Multidimensional Expressions (MDX) expressions. A user can write a DLL containing some special statistical functions, register this DLL with a USE LIBRARY command, and then run queries using these statistical functions.</td>
</tr>
</tbody>
</table>
**DimensionAggUsageTypes**

Enumerates values for the `AggregationUsage` property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DimAggUsageCustom</td>
<td>Creates aggregations for dimension levels as specified by level</td>
</tr>
<tr>
<td>dimAggUsageDetailsOnly</td>
<td>Creates aggregations on only the lowest level in the dimension</td>
</tr>
<tr>
<td>dimAggUsageStandard</td>
<td>Creates aggregations as determined by the aggregation design algorithm</td>
</tr>
<tr>
<td>dimAggUsageTopAndDetailsOnly</td>
<td>Creates aggregations only for the top (All) and lowest levels in the dimension</td>
</tr>
<tr>
<td>dimAggUsageTopOnly</td>
<td>Creates aggregations only for the top (All) level</td>
</tr>
</tbody>
</table>
## DimensionTypes

Enumerates values for the **DimensionType** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DimAccounts</td>
<td>Describes a dimension that contains an accounts structure with parent-child relationships.</td>
</tr>
<tr>
<td>DimBillOfMaterials</td>
<td>Describes a dimension that represents a material/component breakdown. The parent-child relationship implies a parent composed of its children.</td>
</tr>
<tr>
<td>DimChannel</td>
<td>Describes a dimension that contains information about a distribution channel.</td>
</tr>
<tr>
<td>DimCurrency</td>
<td>Describes a dimension that contains currency information.</td>
</tr>
<tr>
<td>DimCustomers</td>
<td>Describes a dimension that contains customer information. The lowest level represents individual customers.</td>
</tr>
<tr>
<td>DimGeography</td>
<td>Describes a dimension that contains a geographic hierarchy.</td>
</tr>
<tr>
<td>DimOrganization</td>
<td>Describes a dimension that represents the reporting structure of an organization.</td>
</tr>
<tr>
<td>DimProducts</td>
<td>Describes a dimension that contains product information. The lowest level represents individual products.</td>
</tr>
<tr>
<td>DimPromotion</td>
<td>Describes a dimension that contains information about marketing and advertising promotions.</td>
</tr>
<tr>
<td>DimQuantitative</td>
<td>Describes a dimension that contains quantitative elements (for example, income level, number of children, and so on).</td>
</tr>
<tr>
<td>DimRates</td>
<td>Describes a dimension that contains different types of rates (for example, buy, sell, discounted and so on).</td>
</tr>
<tr>
<td>DimRegular</td>
<td>The default dimension type, used for dimensions that</td>
</tr>
</tbody>
</table>
are not time-related.

<table>
<thead>
<tr>
<th>DimScenario</th>
<th>Describes a dimension that contains different business scenarios.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DimTime</td>
<td>Indicates that a dimension refers to time (year, month, week, day, and so on). The only valid levels in a time dimension are those that begin with &quot;levTime&quot; as defined in the LevelTypes enumeration.</td>
</tr>
<tr>
<td>DimUtility</td>
<td>Describes a dimension that contains only calculated members. This type of dimension is usually used for data visualization techniques.</td>
</tr>
</tbody>
</table>

**See Also**

[LevelTypes](#)
ErrorCodes

Enumerates error codes. Use this enumerator to determine the meaning of a returned error code in Decision Support Objects (DSO).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mderrAcceptError</td>
<td>An internal error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrAcquireCreditsError</td>
<td>An internal error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrAggregationUsageNotCustom</td>
<td>The EnableAggregations set for levels in Dimension Usage is not set.</td>
</tr>
<tr>
<td></td>
<td>The EnableAggregations field is set for levels in the Analysis server.</td>
</tr>
<tr>
<td>mderrBadParameterForServiceState</td>
<td>Invalid service state parameter on the computer.</td>
</tr>
<tr>
<td>mderrBadRequest</td>
<td>An internal request related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrBindError</td>
<td>An internal bind related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrCalculateError</td>
<td>An internal calculation related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrCanceled</td>
<td>The specified transaction was canceled.</td>
</tr>
<tr>
<td>mderrCannotAddVirtualDimension</td>
<td>Cannot add a virtual dimension because its source dimension is not in the database.</td>
</tr>
<tr>
<td>mderrCannotChangeRemoteServer</td>
<td>Cannot change the remote server if it has been set.</td>
</tr>
<tr>
<td>mderrCannotCloneObjectIntoItself</td>
<td>Cannot clone an object into itself.</td>
</tr>
<tr>
<td>mderrCannotCommitDatabase</td>
<td>Unable to create a database on the Analysis server.</td>
</tr>
</tbody>
</table>
| mderrCannotCreatePartition            | No system partition is available for this operation. System partitions have been programatically defined as user partitions. User-defined partitions are available only if you
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mderrCannotCreateVirtualDimensionFromAnother</td>
<td>Cannot create a virtual dimension based on another virtual dimension.</td>
</tr>
<tr>
<td>mderrCannotDeleteDataSource</td>
<td>At least one object has a reference to the data source, so the data source cannot be deleted.</td>
</tr>
<tr>
<td>mderrCannotDeleteDimension</td>
<td>A dimension cannot be deleted because it is used in a cube.</td>
</tr>
<tr>
<td>mderrCannotDeleteLastPartition</td>
<td>Cannot delete the last partition in a cube. (A cube must have at least one partition.)</td>
</tr>
<tr>
<td>mderrCannotDeleteLevel</td>
<td>Cannot delete a level if it is used in a virtual dimension.</td>
</tr>
<tr>
<td>mderrCannotDeleteMemberProperty</td>
<td>Cannot delete a member property if it is used in a virtual dimension.</td>
</tr>
<tr>
<td>mderrCannotEnableRealTimeUpdatesWithoutIndexedViews</td>
<td>Cannot enable real-time updates on the specified partition without indexed views.</td>
</tr>
<tr>
<td>mderrCannotExecFuncError</td>
<td>Cannot execute a function in a user-defined function library.</td>
</tr>
<tr>
<td>mderrCannotModifySharedObject</td>
<td>Cannot change a property of a shared dimension (or subordinate level) used in a cube.</td>
</tr>
<tr>
<td>mderrCannotRemoveMeasureFromDefaultAggregation</td>
<td>Cannot remove a measure from an aggregation created by the partition analyzer.</td>
</tr>
<tr>
<td>mderrCannotRenameObject</td>
<td>Only temporary objects can be renamed.</td>
</tr>
<tr>
<td>mderrCannotSaveInsideTransaction</td>
<td>Cannot save objects inside a DSO transaction.</td>
</tr>
<tr>
<td>mderrCellCalculationsNotAvailable</td>
<td>Calculated cells are available only if you install Analysis Services for Microsoft SQL Server™ 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrChildProcessFailed</td>
<td>A child process failed within a transaction.</td>
</tr>
<tr>
<td>mderrClassError</td>
<td>An internal class error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrCollectionItemNotFound</td>
<td>Raised if you try to remove an item from a collection that does not exist in the collection.</td>
</tr>
<tr>
<td>mderrCollectionReadOnly</td>
<td>Cannot add an object to, or remove an object from, a collection that is read-only.</td>
</tr>
<tr>
<td>mderrCOMError</td>
<td>An internal COM error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mderrCompatibilityError</td>
<td>An internal compatibility related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrConnectError</td>
<td>An error occurred while connecting to the Analysis server.</td>
</tr>
<tr>
<td>mderrCorruptedProperty</td>
<td>A corrupted property was found while merging partitions.</td>
</tr>
<tr>
<td>mderrCorruptedRegistrySettings</td>
<td>One or more registry settings in use by Analysis Services has been corrupted.</td>
</tr>
<tr>
<td>mderrCouldInitiateCubeUpdate</td>
<td>Could not initiate a cube update.</td>
</tr>
<tr>
<td>mderrCouldInitiateDimensionUpdate</td>
<td>Could not initiate a dimension update.</td>
</tr>
<tr>
<td>mderrCouldNotLockObject</td>
<td>Raised if you try to lock an object that is already locked (by a different application).</td>
</tr>
<tr>
<td>mderrCouldNotLogMissingMemberKeyErrors</td>
<td>Could not write errors regarding missing member key errors to the log file.</td>
</tr>
<tr>
<td>mderrCouldNotOpenService</td>
<td>The Analysis server runs as a Microsoft Windows NT® 4.0 or Windows® 2000 service. This error is raised if the service could not be opened. For more information about the error, see the Microsoft Win32® API documentation.</td>
</tr>
<tr>
<td>mderrCouldNotOpenServiceControlManager</td>
<td>The Analysis server runs as a Windows NT 4.0 or Windows 2000 service. This error is raised if the service control manager could not be opened. For more information about the error, see the Microsoft Win32® API documentation.</td>
</tr>
<tr>
<td>mderrCouldNotQueryTheService</td>
<td>The Analysis server runs as a Windows NT 4.0 or Windows 2000 service. This error is raised if the service could not be queried. For more information about the error, see the Microsoft Win32® API documentation.</td>
</tr>
<tr>
<td>mderrCouldNotUnLockObject</td>
<td>The specified object could not be unlocked.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mderrCubeDimHasNoDatabaseDim</td>
<td>The specified dimension to be associated with a cube does not have a corresponding database dimension.</td>
</tr>
<tr>
<td>mderrCubeNotProcessed</td>
<td>The specified cube has not yet been processed.</td>
</tr>
<tr>
<td>mderrCustomRollupsNotAvailable</td>
<td>Custom rollups are available only if you install Analysis Services for Microsoft SQL Server 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrDataError</td>
<td>An internal data related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrDefinitionCannotBeEmpty</td>
<td>An empty definition was found while merging partitions.</td>
</tr>
<tr>
<td>mderrDefinitionDoesNotContainNameAndValue</td>
<td>A definition which does not contain a name and value was found while merging partitions.</td>
</tr>
<tr>
<td>mderrDeletingTablesOutsideOfTransaction</td>
<td>Tables cannot be deleted outside of a transaction.</td>
</tr>
<tr>
<td>mderrDifferentAggregationDatasources</td>
<td>Partitions cannot be merged because source and target partitions have different relational data sources.</td>
</tr>
<tr>
<td>mderrDifferentAggregationNumber</td>
<td>Partitions cannot be merged because source and target partitions have different numbers of aggregations.</td>
</tr>
<tr>
<td>mderrDifferentAggregationOLAPMode</td>
<td>Partitions cannot be merged because source and target partitions have different storage modes.</td>
</tr>
<tr>
<td>mderrDifferentAggregationStructure</td>
<td>Partitions cannot be merged because source and target partitions have different structures or storage modes.</td>
</tr>
<tr>
<td>mderrDifferentRemoteServers</td>
<td>Cannot merge two partitions that are on different servers.</td>
</tr>
<tr>
<td>mderrDimensionChangingCannotAddLevel</td>
<td>The specified changing dimension in a cube, and either does not support adding a new lowest level or has an AggregationUsage of dimAggUsageStandard or dimAggUsageChanges when changing the top level.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mderrDimensionLockedByCube</td>
<td>Dimension is locked because it is currently being used in a cube. Remove</td>
</tr>
<tr>
<td></td>
<td>the dimension from the cube to unlock it.</td>
</tr>
<tr>
<td>mderrDimensionMemberNotFound</td>
<td>A member was found in the fact table, but not in the dimension.</td>
</tr>
<tr>
<td>mderrDimensionNotInUnderlyingCubes</td>
<td>Cannot add to a virtual cube a dimension that is not in any of the cubes</td>
</tr>
<tr>
<td></td>
<td>on which the virtual cube is based.</td>
</tr>
<tr>
<td>mderrDimensionWritebackNotAvailable</td>
<td>Dimension writebacks are available only if you install Analysis Services</td>
</tr>
<tr>
<td>mderrDuplicateKeyInCollection</td>
<td>Cannot add to a collection an item with the same name as an item already</td>
</tr>
<tr>
<td></td>
<td>in the collection.</td>
</tr>
<tr>
<td>mderrExecuteSQL</td>
<td>An error occurred while attempting to execute a SQL statement against a</td>
</tr>
<tr>
<td></td>
<td>data source.</td>
</tr>
<tr>
<td>mderrFileError</td>
<td>An internal file system error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrFormulaError</td>
<td>An internal formula error occurred.</td>
</tr>
<tr>
<td>mderrFuncNotSupportedError</td>
<td>An unsupported function was called by a Multidimensional Expressions</td>
</tr>
<tr>
<td></td>
<td>(MDX) statement.</td>
</tr>
<tr>
<td>mderrIllegalMeasureType</td>
<td>Invalid measure data type found in returned SQL rowset.</td>
</tr>
<tr>
<td>mderrIllegalObjectName</td>
<td>Cannot assign an invalid name to an object.</td>
</tr>
<tr>
<td>mderrImpersonateError</td>
<td>An internal error occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrInconsistentAggregations</td>
<td>An inconsistency has been found in the aggregations of a specified partition.</td>
</tr>
<tr>
<td>mderrInitializationFailed</td>
<td>Processing could not be initialized on the specified DSO object.</td>
</tr>
<tr>
<td>mderrInternal</td>
<td>An internal error occurred in the DSO library.</td>
</tr>
<tr>
<td>mderrInternetError</td>
<td>An error occurred with a linked cube that is available through an HTTP</td>
</tr>
<tr>
<td></td>
<td>connection.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mderrInvalidAggregateFunction</td>
<td>An invalid aggregate function was specified.</td>
</tr>
<tr>
<td>mderrInvalidAggregationLevel</td>
<td>An invalid aggregation level was specified.</td>
</tr>
<tr>
<td>mderrInvalidAggUsage</td>
<td>The AggregationUsage is incompatible with current settings for the dimension.</td>
</tr>
<tr>
<td>mderrInvalidCubeBadFactTableAlias</td>
<td>The SourceTableAlias is incorrectly defined.</td>
</tr>
<tr>
<td>mderrInvalidCubeDrillThroughNotProperlyDefined</td>
<td>The drillthrough options for the cube are not correctly defined.</td>
</tr>
<tr>
<td>mderrInvalidCubeInconsistentAggregations</td>
<td>Cannot create a measure and add aggregations that are not compatible with the distinct count function.</td>
</tr>
<tr>
<td>mderrInvalidCubeMultipleDistinctCountMeasures</td>
<td>Cannot create a measure with an aggDistinctCount measure.</td>
</tr>
<tr>
<td>mderrInvalidCubeNoVisibleDimensions</td>
<td>Cannot create a cube without at least one visible dimension or calculated member.</td>
</tr>
<tr>
<td>mderrInvalidCubeNoVisibleMeasures</td>
<td>Cannot create a cube without at least one visible measure.</td>
</tr>
<tr>
<td>mderrInvalidDataType</td>
<td>An invalid data type was specified.</td>
</tr>
<tr>
<td>mderrInvalidDimensionBadAreMemberKeysUnique</td>
<td>The AreMemberKeysUnique is True on a dimension with at least one level with AreMemberKeysUnique.</td>
</tr>
<tr>
<td>mderrInvalidDimensionBadAreMemberNamesUnique</td>
<td>The AreMemberNamesUnique is True on a dimension with at least one level with AreMemberNamesUnique.</td>
</tr>
<tr>
<td>mderrInvalidDimensionBadDependsOnDimension</td>
<td>The DependsOnDimension is nonexistent dimension.</td>
</tr>
<tr>
<td>mderrInvalidDimensionLevelsAfterHiddenMustBeUnique</td>
<td>Must have non-unique keys in levels that are below a hidden level.</td>
</tr>
<tr>
<td>mderrInvalidDimensionNoMemberValues</td>
<td>Cannot create a dimension that is unrelated to the fact table and has levels without custom rollup expressions or custom rollup columns.</td>
</tr>
<tr>
<td>mderrInvalidDimensionNoVisibleLevels</td>
<td>Cannot create a dimension without at least one visible level.</td>
</tr>
<tr>
<td>Message Code</td>
<td>Message Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>mderrInvalidDimensionParentChildInvalidLevel</td>
<td>Cannot create a parent-child dimension that contains a non-parent-child level (All) level.</td>
</tr>
<tr>
<td>mderrInvalidDimensionParentChildLevelMissing</td>
<td>Cannot create a parent-child dimension without a parent-child level.</td>
</tr>
<tr>
<td>mderrInvalidLevelBadCustomRollupColumn</td>
<td>The level has an invalid value for its CustomRollupColumn.</td>
</tr>
<tr>
<td>mderrInvalidLevelBadOrderingMemberProperty</td>
<td>The OrderingMemberProperty does not refer to a member property of the level.</td>
</tr>
<tr>
<td>mderrInvalidLevelBadParentKey</td>
<td>A parent-child level has an invalid value for its ParentKeyColumn.</td>
</tr>
<tr>
<td>mderrInvalidLevelBadSkippedLevelsColumn</td>
<td>A parent-child level has an invalid value for its SkippedLevelsColumn.</td>
</tr>
<tr>
<td>mderrInvalidLevelConflictingMemberProperties</td>
<td>A member property has a by another member property with an identical language setting.</td>
</tr>
<tr>
<td>mderrInvalidLevelGrouping</td>
<td>The value of the for the current dimension.</td>
</tr>
<tr>
<td>mderrInvalidLevelNamingTemplate</td>
<td>The LevelNamingTemplate to conflicting level names and may cause problems during processing.</td>
</tr>
<tr>
<td>mderrInvalidLockType</td>
<td>The LockType argument value specified in the LockObject method of a DSO object is invalid. For more information, see OlapLockTypes.</td>
</tr>
<tr>
<td>mderrInvalidMeasure</td>
<td>An invalid measure was specified.</td>
</tr>
<tr>
<td>mderrInvalidParent</td>
<td>An object that is not a member of a collection has no parent.</td>
</tr>
<tr>
<td>mderrInvalidPartBadFactTableAlias</td>
<td>The SourceTableAlias incorrect.</td>
</tr>
<tr>
<td>mderrInvalidPermission</td>
<td>An invalid member security attribute was specified in the DSO Role object.</td>
</tr>
<tr>
<td>mderrInvalidProcessType</td>
<td>An invalid process type was specified in the Process method of a DSO object. For more information, see OlapProcessTypes.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mderrInvalidPropertySetting</td>
<td>Cannot add an object to, or remove an object from, a collection that is read-only.</td>
</tr>
<tr>
<td>mderrInvalidRelatedColumn</td>
<td>An invalid column name was specified in the RelatedColumn object.</td>
</tr>
<tr>
<td>mderrInvalidRemotePartition</td>
<td>The RemoteServer contains the name of a nonexistent partition.</td>
</tr>
<tr>
<td>mderrInvalidRemoteServerName</td>
<td>The RemoteServer contains the name of a nonexistent server.</td>
</tr>
<tr>
<td>mderrInvalidSourceOlapObject</td>
<td>An invalid object was specified in the SourceOlapObject clsColumn object.</td>
</tr>
<tr>
<td>mderrInvalidStructure</td>
<td>The structure of the object that raised the error is invalid.</td>
</tr>
<tr>
<td>mderrInvalidTransactionOperation</td>
<td>Unable to begin, commit, or rollback a transaction on a DSO case of the BeginTrans transaction is in process. In the case of the CommitTrans transaction is currently in process.</td>
</tr>
<tr>
<td>mderrInvalidVirtualDimensionMustHaveAllLevel</td>
<td>Cannot create a virtual dimension that does not contain an (All) level.</td>
</tr>
<tr>
<td>mderrLastLevelMustBeUnique</td>
<td>The settings for the dimension require the AreMemberKeysUnique level in the dimension to be True.</td>
</tr>
<tr>
<td>mderrLinkedCubeCannotChangeProperty</td>
<td>Cannot change the ColumnType and measure in a linked cube.</td>
</tr>
<tr>
<td>mderrLinkedCubeInvalidConnectionString</td>
<td>The ConnectionString cube object contains incorrect or incomplete information. It must refer to a server in Microsoft SQL Server Analysis Services.</td>
</tr>
<tr>
<td>mderrLinkedCubeInvalidServer</td>
<td>The publishing and subscribing servers need to be different when creating a linked cube.</td>
</tr>
<tr>
<td>MD Error Code</td>
<td>Error Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mderrLinkedCubeInvalidSourceCube</td>
<td>The name of the published cube is invalid, or the user does not have permission to query the cube.</td>
</tr>
<tr>
<td>mderrLinkedCubeNoAggregationsAllowed</td>
<td>Aggregations are not allowed for linked cubes.</td>
</tr>
<tr>
<td>mderrLinkedCubeNotEnoughDimensions</td>
<td>While creating a linked cube, no dimensions were found in the source cube.</td>
</tr>
<tr>
<td>mderrLinkedCubesNotAvailable</td>
<td>Linked cubes are available only if you install Analysis Services for Microsoft SQL Server 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrLinkedCubeSynchronizationFailed</td>
<td>Linked cube structure synchronization between subscribing server and publishing server failed.</td>
</tr>
<tr>
<td>mderrListenError</td>
<td>An internal error related to real-time updates occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrLoadDLLLError</td>
<td>An error occurred while loading a user-defined function library.</td>
</tr>
<tr>
<td>mderrLockAccessError</td>
<td>Unable to lock an object that is already locked.</td>
</tr>
<tr>
<td>mderrLockCannotBeObtained</td>
<td>Unable to obtain a lock from the server.</td>
</tr>
<tr>
<td>mderrLockDescriptionTooLong</td>
<td>Lock description is longer than permitted.</td>
</tr>
<tr>
<td>mderrLockFileCorrupted</td>
<td>The server reported that the lock file is corrupted.</td>
</tr>
<tr>
<td>mderrLockFileMissing</td>
<td>The server reported that the lock file is missing.</td>
</tr>
<tr>
<td>mderrLockNetworkDown</td>
<td>Network error.</td>
</tr>
<tr>
<td>mderrLockNetworkNameNotFound</td>
<td>Cannot find name on the network.</td>
</tr>
<tr>
<td>mderrLockNetworkPathNotFound</td>
<td>Cannot find this network path.</td>
</tr>
<tr>
<td>mderrLockNotEnoughMemory</td>
<td>There is not enough memory available to create a lock on a DSO object using the method.</td>
</tr>
<tr>
<td>mderrLockObjectNotLocked</td>
<td>Cannot unlock an object that is not locked.</td>
</tr>
<tr>
<td>mderrLockSystemError</td>
<td>A lock cannot be obtained because of an unknown error.</td>
</tr>
<tr>
<td>mderrMeasureDoesNotHaveValidSourceColumn</td>
<td>Cannot add a measure to a virtual cube if the name of the measure's source column is not in the correct format.</td>
</tr>
<tr>
<td>mderrMemberPropertyNotFound</td>
<td>The member property was not found.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mderrMemoryError</td>
<td>An internal memory related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrMergedPartitionsMustBothUseIndexedViewsOrTables</td>
<td>Partitions to be merged must both use indexed views or aggregation tables.</td>
</tr>
<tr>
<td>mderrMiningModelNotProcessed</td>
<td>The mining model cannot be updated because it has not yet been processed.</td>
</tr>
<tr>
<td>mderrNameCannotBeChanged</td>
<td>Cannot change the name of a DSO object unless the object is a temporary.</td>
</tr>
<tr>
<td>mderrNameCannotBeEmpty</td>
<td>An object cannot have an empty name.</td>
</tr>
<tr>
<td>mderrNetworkError</td>
<td>An internal network related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrNoConnectionToServer</td>
<td>A connection cannot be opened to the Analysis server.</td>
</tr>
<tr>
<td>mderrNoEntryPointError</td>
<td>An entry point could not be found while loading a user-defined function.</td>
</tr>
<tr>
<td>mderrObjectCantBeProcessedWithItsDimensions</td>
<td>A dimension used by the specified DSO object has already been processed in the same transaction.</td>
</tr>
<tr>
<td>mderrObjectChangedByAnotherApp</td>
<td>Cannot save the object because it was not locked and was changed by another object.</td>
</tr>
<tr>
<td>mderrObjectIsNotWriteLocked</td>
<td>Cannot update an object that is not write-locked.</td>
</tr>
<tr>
<td>mderrObsoleteError</td>
<td>The reference to a DSO object has become obsolete.</td>
</tr>
<tr>
<td>mderrODBC</td>
<td>An internal error has occurred in an ODBC data source provider.</td>
</tr>
<tr>
<td>mderrODBCError</td>
<td>An internal ODBC related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrOSError</td>
<td>An internal operating system related error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrPartitionMustBeProcessed</td>
<td>The partition associated with the specified DSO object must first be processed.</td>
</tr>
<tr>
<td>mderrProcessError</td>
<td>An internal processing error has occurred within the DSO library.</td>
</tr>
<tr>
<td>mderrPropertyCannotBeChanged</td>
<td>Property cannot be changed in this context.</td>
</tr>
<tr>
<td>mderrPropertyCollectionCannotBeChanged</td>
<td>An internal error occurred while merging an internal property collection.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mderrRealTimeUpdatesNotAvailable</td>
<td>Real-time update features are available only if you install Analysis Services for Microsoft SQL Server 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrRegistryConnectFailed</td>
<td>An error occurred while connecting to the registry.</td>
</tr>
<tr>
<td>mderrRegistryOpenKeyFailed</td>
<td>An error occurred while opening a registry key.</td>
</tr>
<tr>
<td>mderrRegistryQueryValueFailed</td>
<td>An error occurred while retrieving a value from a registry key.</td>
</tr>
<tr>
<td>mderrRemotePartitionCannotHaveWriteableDimension</td>
<td>A remote partition cannot contain a write-enabled dimension.</td>
</tr>
<tr>
<td>mderrRepositoryConnectionFailed</td>
<td>Object repository may be read-only.</td>
</tr>
<tr>
<td>mderrRepositoryConnectionStringChanged</td>
<td>Another application has changed the repository connection string for the specified Analysis Server. You need to close and reopen this server connection in order to continue.</td>
</tr>
<tr>
<td>mderrRepositoryIncompatible</td>
<td>Repository is incompatible with this version of DSO. Verify that your DSO version is compatible with your repository version.</td>
</tr>
<tr>
<td>mderrRepositoryUpgradeFailed</td>
<td>An error occurred while attempting to update the repository for the specified Analysis server.</td>
</tr>
<tr>
<td>mderrRevertError</td>
<td>An internal error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrROLAPDimensionsNotAvailable</td>
<td>Relational OLAP (ROLAP) dimensions are available only if you install Analysis Services for Microsoft SQL Server 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrROLAPDimensionsRequireROLAPPertition</td>
<td>Cannot add a relational OLAP (ROLAP) dimension to a non-ROLAP partition.</td>
</tr>
<tr>
<td>mderrSecurityError</td>
<td>An internal security error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrSelectError</td>
<td>An internal SQL error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrServerInternal</td>
<td>An internal error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
</tr>
<tr>
<td>mderrServerObjectNotFound</td>
<td>The specified Analysis server could not be found.</td>
</tr>
<tr>
<td>mderrServerObjectNotOpened</td>
<td>The specified Analysis server was not opened before attempting an action with an object associated with the Analysis server.</td>
</tr>
<tr>
<td>mderrSkippedLevelsNotAvailable</td>
<td>Skipped levels and ragged hierarchies are available only if you install Analysis Services for Microsoft SQL Server 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrSourceDoesNotExist</td>
<td>Cannot merge partitions because the source partition does not exist.</td>
</tr>
<tr>
<td>mderrStructureHasChanged</td>
<td>The structure of the specified object has changed.</td>
</tr>
<tr>
<td>mderrTargetDoesNotExist</td>
<td>Cannot merge partitions because the target partition does not exist.</td>
</tr>
<tr>
<td>mderrTimeOut</td>
<td>Connection to the Analysis server timed out.</td>
</tr>
<tr>
<td>mderrTimeoutError</td>
<td>A timeout error has occurred on the Analysis server.</td>
</tr>
<tr>
<td>mderrTooManyDimensionMembers</td>
<td>More than the allowed maximum of 64,000 dimension member children for a single parent member.</td>
</tr>
<tr>
<td>mderrTooManyLevelsInDimension</td>
<td>The maximum number of levels in a dimension is 64, that is, 63 plus an (All) level.</td>
</tr>
<tr>
<td>mderrTooManyMissingMemberKeys</td>
<td>The maximum number of dimension processing errors has been exceeded.</td>
</tr>
<tr>
<td>mderrUnexpectedError</td>
<td>An unexpected error has occurred.</td>
</tr>
<tr>
<td>mderrUnsuccesfullServiceOperation</td>
<td>The Analysis server service (MSSQLServerOLAPService) is not running on the specified computer.</td>
</tr>
<tr>
<td>mderrUserDefinedPartitionsNotAvailable</td>
<td>User-defined partitions are available only if you install Analysis Services for Microsoft SQL Server 2000 Enterprise Edition.</td>
</tr>
<tr>
<td>mderrValidateLastLevelMustBeUnique</td>
<td>The AreMemberKeysUnique property of the last level of a regular dimension with the IsChanging property is False.</td>
</tr>
</tbody>
</table>
GroupingValues

Enumerates values for the Grouping property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupingAutomatic</td>
<td>Level members are grouped automatically by the Analysis server.</td>
</tr>
<tr>
<td>groupingNone</td>
<td>Level members are not grouped.</td>
</tr>
</tbody>
</table>
**HideIfValues**

Enumerates values for the **HideMemberIf** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hideIfBlankName</td>
<td>A level member is hidden when its name is empty.</td>
</tr>
<tr>
<td>hideIfOnlyChildAndBlankName</td>
<td>A level member is hidden when it is the only child of its parent and its name is null or an empty string.</td>
</tr>
<tr>
<td>hideIfOnlyChildAndParentsName</td>
<td>A level member is hidden when it is the only child of its parent and its name is the same as its parent's name.</td>
</tr>
<tr>
<td>hideIfParentsName</td>
<td>A level member is hidden when its name is identical to that of its parent.</td>
</tr>
<tr>
<td>hideNever</td>
<td>Level members are never hidden.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**LanguageValues**

Enumerates values for the `Language` property of a member property. These values are based on the PrimaryLangIDs defined in Microsoft® Windows NT® 4.0 and Windows® 2000. PrimaryLangIDs can be derived directly from a LocaleID.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>languageAfrikaans</td>
<td>Property associated with Afrikaans</td>
</tr>
<tr>
<td>languageAlbanian</td>
<td>Property associated with Albanian</td>
</tr>
<tr>
<td>languageAny</td>
<td>Property associated with any language</td>
</tr>
<tr>
<td>languageArabic</td>
<td>Property associated with Arabic</td>
</tr>
<tr>
<td>languageBasque</td>
<td>Property associated with Basque</td>
</tr>
<tr>
<td>languageBulgarian</td>
<td>Property associated with Bulgarian</td>
</tr>
<tr>
<td>languageByelorussian</td>
<td>Property associated with Byelorussian</td>
</tr>
<tr>
<td>languageCatalan</td>
<td>Property associated with Catalan</td>
</tr>
<tr>
<td>languageChinese</td>
<td>Property associated with Chinese</td>
</tr>
<tr>
<td>languageCzech</td>
<td>Property associated with Czech</td>
</tr>
<tr>
<td>languageDanish</td>
<td>Property associated with Danish</td>
</tr>
<tr>
<td>languageDutch</td>
<td>Property associated with Dutch</td>
</tr>
<tr>
<td>languageEnglish</td>
<td>Property associated with English</td>
</tr>
<tr>
<td>languageEstonian</td>
<td>Property associated with Estonian</td>
</tr>
<tr>
<td>languageFaeroese</td>
<td>Property associated with Faeroese</td>
</tr>
<tr>
<td>languageFarsi</td>
<td>Property associated with Farsi</td>
</tr>
<tr>
<td>languageFinnish</td>
<td>Property associated with Finnish</td>
</tr>
<tr>
<td>languageFrench</td>
<td>Property associated with French</td>
</tr>
<tr>
<td>languageGerman</td>
<td>Property associated with German</td>
</tr>
<tr>
<td>languageGreek</td>
<td>Property associated with Greek</td>
</tr>
<tr>
<td>languageHebrew</td>
<td>Property associated with Hebrew</td>
</tr>
<tr>
<td>languageHungarian</td>
<td>Property associated with Hungarian</td>
</tr>
<tr>
<td>languageIcelandic</td>
<td>Property associated with Icelandic</td>
</tr>
<tr>
<td>languageIndonesian</td>
<td>Property associated with Indonesian</td>
</tr>
<tr>
<td>language</td>
<td>Property associated with</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Italian</td>
<td>Italian</td>
</tr>
<tr>
<td>Japanese</td>
<td>Japanese</td>
</tr>
<tr>
<td>Kampuchean</td>
<td>Kampuchean</td>
</tr>
<tr>
<td>Korean</td>
<td>Korean</td>
</tr>
<tr>
<td>Laotian</td>
<td>Laotian</td>
</tr>
<tr>
<td>Latvian</td>
<td>Latvian</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>Lithuanian</td>
</tr>
<tr>
<td>Macedonian</td>
<td>Macedonian</td>
</tr>
<tr>
<td>Maltese</td>
<td>Maltese</td>
</tr>
<tr>
<td>Maori</td>
<td>Maori</td>
</tr>
<tr>
<td>Norwegian</td>
<td>Norwegian</td>
</tr>
<tr>
<td>Polish</td>
<td>Polish</td>
</tr>
<tr>
<td>Portuguese</td>
<td>Portuguese</td>
</tr>
<tr>
<td>RhaetoRomanic</td>
<td>RhaetoRomanic</td>
</tr>
<tr>
<td>Romanian</td>
<td>Romanian</td>
</tr>
<tr>
<td>Russian</td>
<td>Russian</td>
</tr>
<tr>
<td>Sami</td>
<td>Sami</td>
</tr>
<tr>
<td>ScotsGaelic</td>
<td>ScotsGaelic</td>
</tr>
<tr>
<td>SerboCroatian</td>
<td>SerboCroatian</td>
</tr>
<tr>
<td>Slovak</td>
<td>Slovak</td>
</tr>
<tr>
<td>Slovenian</td>
<td>Slovenian</td>
</tr>
<tr>
<td>Sorbian</td>
<td>Sorbian</td>
</tr>
<tr>
<td>Spanish</td>
<td>Spanish</td>
</tr>
<tr>
<td>Sutu</td>
<td>Sutu</td>
</tr>
<tr>
<td>Swedish</td>
<td>Swedish</td>
</tr>
<tr>
<td>Thai</td>
<td>Thai</td>
</tr>
<tr>
<td>Tsonga</td>
<td>Tsonga</td>
</tr>
<tr>
<td>Tswana</td>
<td>Tswana</td>
</tr>
<tr>
<td>Turkish</td>
<td>Turkish</td>
</tr>
<tr>
<td>Ukrainian</td>
<td>Ukrainian</td>
</tr>
<tr>
<td>Urdu</td>
<td>Urdu</td>
</tr>
<tr>
<td>Venda</td>
<td>Venda</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>Language</td>
<td>Property associated with</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Xhosa</td>
<td>Xhosa</td>
</tr>
<tr>
<td>Zulu</td>
<td>Zulu</td>
</tr>
</tbody>
</table>
LevelTypes

Enumerates values for the LevelType property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>levAccount</td>
<td>Indicates that a level exists within an account dimension.</td>
</tr>
<tr>
<td>levAll</td>
<td>Indicates the top (All) level of a dimension (the one that precalculates all the members of all lower levels).</td>
</tr>
<tr>
<td>levBOMResource</td>
<td>Indicates that a level is part of a bill of materials dimension.</td>
</tr>
<tr>
<td>levChannel</td>
<td>Indicates that a level exists within a distribution channel dimension.</td>
</tr>
<tr>
<td>levCompany</td>
<td>Indicates that a level contains information about a company.</td>
</tr>
<tr>
<td>levCurrencyDestination</td>
<td>Indicates that a level contains information about the resulting currency after a foreign exchange conversion.</td>
</tr>
<tr>
<td>levCurrencySource</td>
<td>Indicates that a level contains information about the starting currency before a foreign exchange conversion.</td>
</tr>
<tr>
<td>levCustomer</td>
<td>Indicates that a level contains information about an individual customer.</td>
</tr>
<tr>
<td>levCustomerGroup</td>
<td>Indicates that a level contains information about a customer group.</td>
</tr>
<tr>
<td>levCustomerHousehold</td>
<td>Indicates that a level contains information about an entire household.</td>
</tr>
<tr>
<td>levGeoCity</td>
<td>Indicates that a level refers to a city name.</td>
</tr>
<tr>
<td>levGeoContinent</td>
<td>Indicates that a level refers to a continent name.</td>
</tr>
<tr>
<td>levGeoCountry</td>
<td>Indicates that a level refers to a country name.</td>
</tr>
<tr>
<td>levGeoCounty</td>
<td>Indicates that a level refers to a county name.</td>
</tr>
<tr>
<td>levGeoPoint</td>
<td>Indicates that a level refers to a location type that</td>
</tr>
</tbody>
</table>
does not fit into the other geographic categories.

<table>
<thead>
<tr>
<th>levGeoPostalCode</th>
<th>Indicates that a level refers to a postal code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>levGeoRegion</td>
<td>Indicates that a level refers to a custom-defined region.</td>
</tr>
<tr>
<td>levGeoStateOrProvince</td>
<td>Indicates that a level refers to a state or province name.</td>
</tr>
<tr>
<td>levOrgUnit</td>
<td>Indicates that a level refers to the name of a unit within a larger organization.</td>
</tr>
<tr>
<td>levPerson</td>
<td>Indicates that a level refers to an individual within a larger organization.</td>
</tr>
<tr>
<td>levProduct</td>
<td>Indicates that a level refers to an individual product.</td>
</tr>
<tr>
<td>levProductGroup</td>
<td>Indicates that a level refers to a product group.</td>
</tr>
<tr>
<td>levPromotion</td>
<td>Indicates that a level refers to a promotion.</td>
</tr>
<tr>
<td>levQuantitative</td>
<td>Indicates that a level refers to a quantitative member within a quantitative dimension.</td>
</tr>
<tr>
<td>levRegular</td>
<td>Indicates that the level is not related to time.</td>
</tr>
<tr>
<td>levRepresentative</td>
<td>Indicates that a level refers to a sales representative.</td>
</tr>
<tr>
<td>levScenario</td>
<td>Indicates that a level refers to a scenario.</td>
</tr>
<tr>
<td>levTimeDays</td>
<td>Indicates that a level refers to days. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeHalfYears</td>
<td>Indicates that a level refers to half-years. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeHours</td>
<td>Indicates that a level refers to hours. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeMinutes</td>
<td>Indicates that a level refers to minutes. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeMonths</td>
<td>Indicates that a level refers to months. Must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeQuarters</td>
<td>Indicates that a level refers to (calendar) quarters. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeSeconds</td>
<td>Indicates that a level refers to seconds. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeUndefined</td>
<td>Indicates that a level refers to an indeterminate or nonstandard measurement of time. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>levTimeWeeks</td>
<td>Indicates that a level refers to weeks. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levTimeYears</td>
<td>Indicates that a level refers to years. It must be used in a dimension whose type is dimTime.</td>
</tr>
<tr>
<td>levUtility</td>
<td>Indicates that a level refers to a calculated member in a utility dimension.</td>
</tr>
</tbody>
</table>
**MembersWithDataValues**

Enumerates values for the `MembersWithData` property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DataForLeafMembersOnly</code></td>
<td>Only leaf members can have data in the fact table. A processing error occurs if data for a nonleaf member appears in the fact table.</td>
</tr>
<tr>
<td><code>DataForNonLeafMembersHidden</code></td>
<td>Any member (except the All member) can have data in the fact table. Data for nonleaf members is hidden (totals may not appear to add up correctly).</td>
</tr>
<tr>
<td><code>DataForNonLeafMembersVisible</code></td>
<td>Any member (except the All member) can have data in the fact table. Data for nonleaf members is visible in system-generated leaf members.</td>
</tr>
</tbody>
</table>
OlapEditions

Enumerates values for the **Edition** property of objects of **ClassType clsServer**.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlapEditionError</td>
<td>The edition of the Analysis server cannot be determined.</td>
</tr>
</tbody>
</table>
## OlapLockTypes

Enumerates values for the *LockType* parameter of the *LockObject* method, implemented by most objects in the Decision Support Objects (DSO) library.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OlapLockExtendedRead</strong></td>
<td>The object's properties can be read by other applications, but they cannot be changed or processed. This lock is used to prevent processing of dependent objects of an object that is being processed, such as dimensions that are shared by multiple cubes. Multiple <em>olapLockExtendedRead</em> locks can be applied to an object by multiple applications. However, no application can lock the object for processing or updating until all <em>olapLockExtendedRead</em> locks have been released.</td>
</tr>
<tr>
<td><strong>OlapLockProcess</strong></td>
<td>The object's Process method can be initiated, and other applications can read the object's properties only until the lock is released. Only one <em>olapLockProcess</em> lock can be applied to an object at a time, and other applications can only apply <em>olapLockRead</em> locks while the <em>olapLockProcess</em> lock is in place.</td>
</tr>
<tr>
<td><strong>OlapLockRead</strong></td>
<td>The properties of the object can be read from the repository and cannot be changed by another application until the lock is released. Other applications can issue <em>olapLockRead</em>, <em>olapLockExtendedRead</em>, and <em>olapLockProcess</em> locks, but not <em>olapLockWrite</em> locks, while the initial <em>olapLockRead</em> lock is in place.</td>
</tr>
<tr>
<td><strong>OlapLockWrite</strong></td>
<td>The properties of the object can be modified in the repository using the Update method, and</td>
</tr>
</tbody>
</table>
they are not available to other applications for any use until the lock is released. No other locks of any type can be applied to the object until the `olapLockWrite` lock is released.

See Also

LockObject
OlapStateTypes

Enumerates values for the **State** property for objects other than server objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlapStateCurrent</td>
<td>The state of the object is current.</td>
</tr>
<tr>
<td>OlapStateMemberPropertiesChanged</td>
<td>The member properties of the object have changed.</td>
</tr>
<tr>
<td>OlapStateNeverProcessed</td>
<td>The object has never been processed.</td>
</tr>
<tr>
<td>OlapStateSourceMappingChanged</td>
<td>The source mapping for the object, such as the <strong>MemberKeyColumn</strong> or <strong>ParentKeyColumn</strong> properties of a <strong>clsDatabaseDimension</strong> object, has changed.</td>
</tr>
<tr>
<td>olapStateStructureChanged</td>
<td>The structure of the object has changed.</td>
</tr>
</tbody>
</table>

See Also

[ServerStates](#)
OlapStorageModes

Enumerates values for the **OlapMode** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>olapmodeAggsMolapIndex</td>
<td>Reserved for future use.</td>
<td>n/a</td>
</tr>
<tr>
<td>olapmodeAggsRolap</td>
<td>Reserved for future use.</td>
<td>n/a</td>
</tr>
<tr>
<td>olapmodeHybridIndex</td>
<td>Fact table data is stored in relational OLAP (ROLAP), and aggregations are stored in multidimensional OLAP (MOLAP).</td>
<td>Databases, cubes, and partitions (excluding virtual cubes and linked cubes)</td>
</tr>
<tr>
<td>olapmodeMolapIndex</td>
<td>Fact table data is stored in MOLAP, and aggregations are stored in MOLAP.</td>
<td>Databases, cubes, and partitions (excluding virtual cubes)</td>
</tr>
<tr>
<td>olapmodeRolap</td>
<td>All data is stored in ROLAP.</td>
<td>Databases, cubes, and partitions (excluding virtual cubes and linked cubes)</td>
</tr>
</tbody>
</table>

**See Also**

[OlapMode](#)
**OrderTypes**

Enumerates values for the **Ordering** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderKey</td>
<td>Members are ordered in <strong>MemberKeyColumn</strong> sequence.</td>
</tr>
<tr>
<td>orderMemberProperty</td>
<td>Members are ordered according to the instructions of a member property.</td>
</tr>
<tr>
<td>orderName</td>
<td>Members are ordered in <strong>MemberNameColumn</strong> sequence.</td>
</tr>
</tbody>
</table>

For more information, see [OrderingMemberProperty](#).
ProcessOptimizationModes

Enumerates values for the ProcessOptimizationMode property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processOptimizationModeLazyOptimizations</td>
<td>The object supports lazy optimization processing. Data is read from the data source and stored within the processing transaction. Indexing and aggregating are performed afterward.</td>
</tr>
<tr>
<td>ProcessOptimizationModeRegular</td>
<td>The object uses normal processing. Data is read from the data source and stored, indexed, and aggregated within the processing transaction.</td>
</tr>
</tbody>
</table>

For more information, see ProcessOptimizationMode.
### ProcessTypes

Enumerates values for the Option parameter of the **Process** method.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>processBuildStructure</strong></td>
<td>Applies only to cubes. Processing a cube ordinarily causes the Analysis server to read all source data corresponding to the definition of the cube, create the cube, and populate it with data. This option causes the Analysis server to create the cube (that is, build its structure) but not populate it with data. Instead, the cube exists as an empty shell on the server. A user can connect to the cube, but it contains no data. This option can have performance benefits. If you do not use this option (see <strong>processDefault</strong>), the partitions in a cube are processed sequentially. If you do use it, you can process the partitions in parallel.</td>
</tr>
<tr>
<td><strong>processDefault</strong></td>
<td>The default option. It causes the system to decide what processing method is best. Typically this means that the system will try to refresh the object's data (<strong>processRefreshData</strong>) unless its structure has changed or it no longer exists. In the latter case, the system will perform a full processing (<strong>processFull</strong>).</td>
</tr>
<tr>
<td><strong>processFull</strong></td>
<td>Causes the object to be fully processed or rebuilt. The object's structure is changed if needed and its data is refreshed (that is, discarded and repopulated). This is the</td>
</tr>
</tbody>
</table>
most complete type of processing supported. See `processRefreshData`.

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>processReaggregate</code></td>
<td>Applies only to cubes and partitions. This option is similar to <code>processRefreshData</code>, except that it instructs the Analysis server to rebuild maps, full indexes, and aggregations for multidimensional OLAP (MOLAP) partitions.</td>
</tr>
<tr>
<td><code>processRefreshData</code></td>
<td>Causes the object data to be refreshed (that is, discarded and repopulated), but does not change the object's structure. This operation occurs inside a transaction, allowing you to continue using current data while the transaction takes place. When the transaction is committed, the new data is available. See <code>processFull</code>.</td>
</tr>
<tr>
<td><code>ProcessRefreshDataAndIndex</code></td>
<td>Applies only to cubes and partitions. Similar to <code>processRefreshData</code>, except this option instructs the Analysis server to build full indexes for the partitions.</td>
</tr>
<tr>
<td><code>processResume</code></td>
<td>Directs the Analysis server to resume responding to user queries against a cube that has had queries suspended. The Analysis server will automatically resume responding to queries after 5 minutes unless <code>processResume</code> is received first.</td>
</tr>
<tr>
<td><code>processSuspend</code></td>
<td>Directs the Analysis server to suspend responses to user queries against a cube so your application can perform operations such as merging fact tables. The Analysis server automatically resumes responding to queries after 5 minutes (earlier if you submit a <code>processResume</code> request).</td>
</tr>
</tbody>
</table>

**Suspend Timeout**

The timeout value for the `processResume` and `processSuspend` constants can be
changed by using the following registry key:

\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\OLAP Server\Olap Manager Info\SuspendTimeout

The registry key stores the value as a long integer representing the timeout in milliseconds. For example, to change the timeout to 2 minutes, the value would be set to 120000. If the registry key is missing, the default value of 5 minutes (300000 milliseconds) is used by Decision Support Objects (DSO).

**Processing and Dependent Objects**

The DSO hierarchy can be complex, involving objects that depend on other objects. This is particularly true of mining model objects, which may have dependent virtual cube or mining dimension objects.

If the object to be processed depends on other objects, such as a mining dimension which, in turn, depends on an OLAP mining model, processing the object will cause the objects on which it depends to be processed, but only if the **State** property of the objects on which it depends is not set to **olapStateCurrent**.

If other objects depend on the object to be processed, such as an OLAP mining model with an associated mining dimension or a virtual cube, processing the object causes the dependent objects to be processed, but only if the **State** property of the dependent objects is not set to **olapStateNeverProcessed**.
**PropertyTypeValue**

Enumerates the values used in the **PropertyType** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propAddress</td>
<td>Address</td>
</tr>
<tr>
<td>propAddressBuilding</td>
<td>Address - building number</td>
</tr>
<tr>
<td>propAddressCity</td>
<td>Address – city</td>
</tr>
<tr>
<td>propAddressCountry</td>
<td>Address – country</td>
</tr>
<tr>
<td>propAddressFax</td>
<td>Address – facsimile number</td>
</tr>
<tr>
<td>propAddressFloor</td>
<td>Address – floor number</td>
</tr>
<tr>
<td>propAddressHouse</td>
<td>Address – house number</td>
</tr>
<tr>
<td>propAddressPhone</td>
<td>Address – telephone number</td>
</tr>
<tr>
<td>propAddressQuarter</td>
<td>Address – quarter</td>
</tr>
<tr>
<td>propAddressRoom</td>
<td>Address – room number</td>
</tr>
<tr>
<td>propAddressStateorProvince</td>
<td>Address – state or province</td>
</tr>
<tr>
<td>propAddressStreet</td>
<td>Address – street name</td>
</tr>
<tr>
<td>propAddressZip</td>
<td>Address – postal code</td>
</tr>
<tr>
<td>propCaption</td>
<td>Caption</td>
</tr>
<tr>
<td>propCaptionAbreviation</td>
<td>Caption – abbreviation</td>
</tr>
<tr>
<td>propCaptionDescription</td>
<td>Caption – description</td>
</tr>
<tr>
<td>propCaptionShort</td>
<td>Caption – short name</td>
</tr>
<tr>
<td>PropDate</td>
<td>Date</td>
</tr>
<tr>
<td>propDateCanceled</td>
<td>Date – canceled date</td>
</tr>
<tr>
<td>propDateDuration</td>
<td>Date – duration date</td>
</tr>
<tr>
<td>propDateEnded</td>
<td>Date – end date</td>
</tr>
<tr>
<td>propDateModified</td>
<td>Date – modified date</td>
</tr>
<tr>
<td>propDateStart</td>
<td>Date – start date</td>
</tr>
<tr>
<td>propFormattingColor</td>
<td>Format – color</td>
</tr>
<tr>
<td>propFormattingFont</td>
<td>Format – font name</td>
</tr>
<tr>
<td>propFormattingFontEffects</td>
<td>Format – font effects</td>
</tr>
<tr>
<td>Prop Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>propFormattingFontSize</td>
<td>Format – font size</td>
</tr>
<tr>
<td>propFormattingOrder</td>
<td>Format – sort order</td>
</tr>
<tr>
<td>propFormattingSubTotal</td>
<td>Format – subtotal</td>
</tr>
<tr>
<td>propGeoBoundaryBottom</td>
<td>Geographical boundary – bottom</td>
</tr>
<tr>
<td>propGeoBoundaryFront</td>
<td>Geographical boundary – front</td>
</tr>
<tr>
<td>propGeoBoundaryLeft</td>
<td>Geographical boundary – left</td>
</tr>
<tr>
<td>propGeoBoundaryPolygon</td>
<td>Geographical boundary – polygon</td>
</tr>
<tr>
<td>propGeoBoundaryRear</td>
<td>Geographical boundary – rear</td>
</tr>
<tr>
<td>propGeoBoundaryRight</td>
<td>Geographical boundary – right</td>
</tr>
<tr>
<td>propGeoBoundaryTop</td>
<td>Geographical boundary – top</td>
</tr>
<tr>
<td>propGeoCentroidX</td>
<td>Geographical boundary – X centroid</td>
</tr>
<tr>
<td>propGeoCentroidY</td>
<td>Geographical boundary – Y centroid</td>
</tr>
<tr>
<td>propGeoCentroidZ</td>
<td>Geographical boundary – Z centroid</td>
</tr>
<tr>
<td>propID</td>
<td>Property – ID</td>
</tr>
<tr>
<td>propOrgTitle</td>
<td>Property – organizational title</td>
</tr>
<tr>
<td>propPersonContact</td>
<td>Person – contact person</td>
</tr>
<tr>
<td>propPersonDemographic</td>
<td>Person – demographic information</td>
</tr>
<tr>
<td>propPersonFirstName</td>
<td>Person – first name</td>
</tr>
<tr>
<td>propPersonFullName</td>
<td>Person – full name</td>
</tr>
<tr>
<td>propPersonLastName</td>
<td>Person – last name</td>
</tr>
<tr>
<td>propPersonMiddleName</td>
<td>Person – middle name</td>
</tr>
<tr>
<td>propPhysicalColor</td>
<td>Physical property – color</td>
</tr>
<tr>
<td>propPhysicalDensity</td>
<td>Physical property – density</td>
</tr>
<tr>
<td>propPhysicalDepth</td>
<td>Physical property – depth</td>
</tr>
<tr>
<td>propPhysicalHeight</td>
<td>Physical property – height</td>
</tr>
<tr>
<td>propPhysicalSize</td>
<td>Physical property – size</td>
</tr>
<tr>
<td>propPhysicalVolume</td>
<td>Physical property – volume</td>
</tr>
<tr>
<td>propPhysicalWeight</td>
<td>Physical property – weight</td>
</tr>
<tr>
<td>propPhysicalWidth</td>
<td>Physical property – width</td>
</tr>
<tr>
<td>propQtyRangeHigh</td>
<td>Quantity – high end of range</td>
</tr>
<tr>
<td>propQtyRangeLow</td>
<td>Quantity – low end of range</td>
</tr>
<tr>
<td>propRegular</td>
<td>Regular (default)</td>
</tr>
<tr>
<td>propRelationToParent</td>
<td>Relationship to parent</td>
</tr>
<tr>
<td>propSequence</td>
<td>Sequence</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>propVersion</td>
<td>Version</td>
</tr>
<tr>
<td>propWebHTML</td>
<td>HTML information</td>
</tr>
<tr>
<td>propWebMailAlias</td>
<td>E-mail address</td>
</tr>
<tr>
<td>propWebURL</td>
<td>URL address</td>
</tr>
<tr>
<td>propWebXMLorXSL</td>
<td>XML or XSL information</td>
</tr>
</tbody>
</table>
RootIfValues

Enumerates values for the **RootMemberIf** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootifParentIsBlank</td>
<td>Only members with a null, a zero, or an empty string in their <strong>ParentKeyColumn</strong> are treated as root members.</td>
</tr>
<tr>
<td>rootifParentIsBlankOrSelfOrMissing</td>
<td>Members are treated as root members if they meet one or more of the conditions specified by rootifParentIsBlank, rootifParentIsSelf, or rootifParentIsMissing.</td>
</tr>
<tr>
<td>rootifParentIsMissing</td>
<td>Only members with parents that cannot be found are treated as root members.</td>
</tr>
<tr>
<td>rootifParentIsSelf</td>
<td>Only members with themselves as parents are treated as root members.</td>
</tr>
</tbody>
</table>
ServerStates

Enumerates values for the State property of server objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateConnected</td>
<td>The attempt to connect to the server succeeded.</td>
</tr>
<tr>
<td>stateFailed</td>
<td>The attempt to connect to the server failed.</td>
</tr>
<tr>
<td>stateUnknown</td>
<td>The application has disconnected from the server or has not yet connected to the server.</td>
</tr>
</tbody>
</table>

See Also

[clsServer]
Analysis Services Programming

**StorageModeValues**

Enumerates values for the `StorageMode` property of dimension objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>storeasROLAP</code></td>
<td>Dimension members are not read during processing and are left in the relational data source.</td>
</tr>
<tr>
<td><code>storeasMOLAP</code></td>
<td>Dimension members are read during processing and are stored in the Analysis server.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

SubClassTypes

Enumerates values for the **SubClassType** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbclsRegular</td>
<td>Indicates that the object is a regular object.</td>
<td>All objects</td>
</tr>
<tr>
<td></td>
<td>In the case of a cube, it indicates that the cube is neither linked nor virtual.</td>
<td></td>
</tr>
<tr>
<td>sbclsLinked</td>
<td>Indicates that the cube is linked to another cube on a remote Analysis server.</td>
<td>clsCube, clsCubeDimension (only for a private dimension of a linked cube)</td>
</tr>
<tr>
<td>sbclsMining</td>
<td>Indicates that the dimension is based on the content of an OLAP data mining model that has processed against a cube.</td>
<td>clsCubeDimension</td>
</tr>
<tr>
<td>sbclsOLAP</td>
<td>Indicates that the data mining model or data mining column is based on an OLAP cube.</td>
<td>clsMiningModel, clsColumn</td>
</tr>
<tr>
<td>sbclsParentChild</td>
<td>Indicates that the level is a parent-child level.</td>
<td>clsAggregationLevel, clsDatabaseLevel, clsCubeLevel, clsPartitionLevel</td>
</tr>
<tr>
<td>sbclsRelational</td>
<td>Indicates that the data mining model or data mining column is based on a relational database.</td>
<td>clsMiningModel, clsColumn</td>
</tr>
<tr>
<td>sbclsRemote</td>
<td>Indicates that the partition is located on a remote Analysis server.</td>
<td>clsPartition</td>
</tr>
<tr>
<td>sbclsVirtual</td>
<td>Indicates that the object is a <strong>clsCube</strong></td>
<td></td>
</tr>
</tbody>
</table>
virtual cube.

See Also

Virtual Cubes
Analysis Services Programming
Collections

Collections used in Decision Support Objects (DSO) generally operate the way standard Microsoft® Visual Basic® collections do. However, there are several differences specific to DSO. DSO collections contain methods and properties that have implementations specifically designed for the DSO object model.

Unlike typical Visual Basic collections, DSO collections can contain only objects of the same type. For example, a Dimensions collection can contain only objects of the object classes that apply to dimensions, such as ClassType clsDatabaseDimension, clsCubeDimension, clsPartitionDimension, and clsAggregationDimension. The parent object of a collection determines the specific ClassType property value for the collection. For example, a Dimensions collection whose parent is of ClassType clsDatabase can contain only objects of ClassType clsDatabaseDimension.

A collection is considered to be static for an object if the range of acceptable objects in the collection is restricted. For example, the Roles collection for an object of ClassType clsCube is a static collection because it can only contain role objects that are defined for the cube's parent database object.

The following table lists the collections used in the DSO object model. With the exception of CustomProperties, all collections implement the methods and properties supported by the clsCollection object. For more information about the implemented properties and methods for clsCollection objects, see clsCollection.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Contains objects of ClassType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>clsCubeCommand clsDatabaseCommand</td>
</tr>
<tr>
<td>Columns</td>
<td>Column objects that make up a mining model's structure (ClassType does not apply)</td>
</tr>
<tr>
<td>CustomProperties</td>
<td>Property Object (ClassType does not apply)</td>
</tr>
<tr>
<td>DataSources</td>
<td>clsDataSource</td>
</tr>
<tr>
<td>Dimensions</td>
<td>clsAggregationDimension clsCubeDimension</td>
</tr>
</tbody>
</table>
### Levels
- clsAggregationLevel
- clsCubeLevel
- clsDatabaseLevel
- clsPartitionLevel

### MDStores
- clsAggregation
- clsCube
- clsDatabase
- clsPartition

### Measures
- clsAggregationMeasure
- clsCubeMeasure
- clsPartitionMeasure

### MemberProperties
- clsMemberProperty

### MiningModels
- clsMiningModel

### Roles
- clsCubeRole
- clsDatabaseRole

---

**See Also**

*Methods, clsCollection*

*Properties, clsCollection*

*Nested Collections*
Nested Collections

For members of a collection that is itself a collection of an object contained in another collection, you can use an intermediate object to access each collection. For example, if object dsoObject has a collection Collection1 that contains objects of type dsoObject1, and each of these has a collection Collection2 that contains objects of type dsoObject2, you can access properties of dsoObject2 objects by setting a temporary object to the intermediate dsoObject1.

Dim TempObject, TempProp
' Retrieve the first level object.
Set TempObject = dsoObject.Collection1(i)
' Retrieve the property.
TempProp = TempObject.Collection2(j).property

Access to the property directly through nested collection references is not implemented in Decision Support Objects (DSO) collections. For example, the following statement will produce an error:

TempProp = dsoObject.Collection1(i).Collection2(j).property

See Also

Collections
Analysis Services Programming

**Commands Collection**

A **Command** object encapsulates a user-defined command or sequence of commands that are automatically executed on the Microsoft® SQL Server™ 2000 Analysis Services client when the user accesses the cube or database. Such commands can include calculated members, named sets, library references, and so on.

Each of the following objects contains a **Commands** collection:

- **clsCube**
- **clsMiningModelRole**
- **clsCubeRole**
- **clsDatabaseRole**

Each **Commands** collection contains command objects of the same class type. The **ContainedClassType** property of a **Commands** collection specifies the class type of the contained objects.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Contained class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>clsCubeCommand</td>
</tr>
<tr>
<td>clsCubeRole</td>
<td>clsCubeCommand</td>
</tr>
<tr>
<td>clsDatabaseRole</td>
<td>clsDatabaseCommand</td>
</tr>
</tbody>
</table>

**Remarks**

The relative position of commands within a **Commands** collection determines the order in which the commands are executed. This is important because it is possible to write commands that depend on previous commands in the collection. For example, a named set can contain a previously defined calculated
As with other Decision Support Objects (DSO) collections, the Add, AddNew, Find, and Remove methods of Commands collections maintain all necessary parent-child relationships among the objects in the object model.

**Note** The Commands collection of objects of ClassType clsDatabase is reserved for future use.

**See Also**

Collections

Collection Methods

Collection Properties
CustomProperties Collection

The `CustomProperties` collection applies to all Decision Support Objects (DSO) objects and enables you to define unique properties for DSO objects. The `CustomProperties` collection contains `Property` objects that you define to store information you want to associate with a DSO object. For more information about defining custom properties, see `Property Object`.

Access

Read/write

Remarks

The `CustomProperties` collection implements its own methods and properties, which operate differently than those of other DSO collections.

Example

Use the following code example to add and retrieve custom property objects:

```vba
' Assume the existence of an object of ClassType clsDimension.
' Add a custom property.
    Dim dsoProp As DSO.Property
    Set dsoProp = dsoDim.CustomProperties.Add(55, "Age", vbInteger)

' Retrieve custom property values.
    Dim dsoProp2 As DSO.Property
    Set dsoProp2 = dsoDim.CustomProperties(1)
    Debug.Print dsoProp2.Name, dsoProp2.Value
```

See Also

(Collections)
Methods, CustomProperties

The following methods apply to the CustomProperties collection. They do not apply to other Decision Support Objects (DSO) collections. For information about methods for other DSO collections, which implement the OlapCollection interface, see Methods, clsCollection.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds a Property object to a CustomProperties collection</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears all Property objects from a CustomProperties collection</td>
</tr>
<tr>
<td>Item</td>
<td>Retrieves a Property object from a CustomProperties collection</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes a Property object from a CustomProperties collection</td>
</tr>
</tbody>
</table>

See Also

CustomProperties Collection
Add (CustomProperties Collection)

The Add method of a CustomProperties collection creates a new Property object, adds it to the collection, and returns a reference to the object. This method applies only to CustomProperties collections.

Syntax

Set vRet = object.Add(ByVal Value, [ByVal Name As String], [ByVal DataType As VBA.VbVarType])

vRet

A Variant variable that receives the instance of the new Property object. Instead of a variant, you can use a variable that has been declared as type DSO.Property to match the object being retrieved from the collection.

object

An instance of a CustomProperties collection.

Value

A Variant that contains the value of the Property object.

Name

(Optional) A string that specifies the name of the Property object.

DataType

(Optional) The data type of the property. A Microsoft® Visual Basic® data type defined in the VBA.VbVarType enumeration.

See Also

CustomProperties Collection

Property Object
Clear (CustomProperties Collection)

The Clear method of a CustomProperties collection clears the collection of all entries. This method applies only to CustomProperties collections.

Syntax

Object.Clear

object

An instance of a CustomProperties collection.

See Also

CustomProperties Collection
Item (CustomProperties Collection)

The Item method of a CustomProperties collection returns an instance of an item in the collection. This method applies only to CustomProperties collections.

Syntax

Set vRet = object.Item(ByVal Index)

vRet

A Variant variable that receives the instance of the new Property object. Instead of a variant, you can use a variable that has been declared as type DSO.Property to match the object being retrieved from the collection.

object

An instance of a CustomProperties collection.

Index

A Variant that specifies the name or index of the object to retrieve.

See Also

CustomProperties Collection
Remove (CustomProperties Collection)

The `Remove` method of a `CustomProperties` collection removes an item from the collection.

**Syntax**

```vbscript
object.Remove(ByVal Index)
```

- **object**
  
  An instance of a `CustomProperties` collection.

- **Index**
  
  A Variant that specifies the name or index of the object to remove.

**See Also**

- [CustomProperties Collection](#)
Properties, CustomProperties

The following properties apply to the CustomProperties collection. They do not apply to other Decision Support Objects (DSO) collections. For information about properties for other DSO collections, which implement the OlapCollection interface, see Properties, clsCollection.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>The number of Property objects in a CustomProperties collection</td>
</tr>
</tbody>
</table>

See Also

CustomProperties Collection
Count (CustomProperties Collection)

The Count property of a CustomProperties collection returns the number of items in the collection.

Data Type

Integer

Access

Read-only

Example

The following code example checks to see whether the CustomProperties collection of a database is empty:

Dim dsoServer As New DSO.Server
Dim dsoDB As DSO.Database

' Connect to local Analysis server.
dsoServer.Connect "LocalHost"

' Get reference to FoodMart 2000 database.
Set dsoDB = dsoServer.MDStores("FoodMart 2000")

' Check for custom properties.
If dsoDB.CustomProperties.Count > 0 Then
    ' There is at least one custom property in the
    ' CustomProperties collection.
End If

' Clean up.
Set dsoDB = Nothing

dsoServer.CloseServer

See Also

CustomProperties Collection
Columns Collection

The Columns collection contains the column objects that are the foundation of a data mining model's structure.

The following object contains a Columns collection:

- clsMiningModel

Remarks

In order to support nested tables, each column object in the collection also contains its own Columns collection. This collection is contained by the mining model object (that is, objects whose ClassType property is clsMiningModel). In addition to the Columns collection, the mining model object also contains other parameters that help to define a model's function.

For more information about nested columns, see Data Mining Columns.

See Also

clsMiningModel

clsColumn
DataSources Collection

A DataSources collection in a Decision Support Objects (DSO) object contains the data sources that are or can be used by the subordinate objects in the object.

Each of the following objects contains a DataSources collection:

- clsCube
- clsDatabase
- clsMiningModel
- clsPartition

Each subordinate object (that is, a cube, a partition, or a shared dimension) can have a unique data source. For example, although a cube can contain only one data source in its DataSources collection, each partition or shared dimension within the cube can have a unique data source.

Regardless of the object in which it resides, each DataSources collection contains objects of ClassType clsDataSource.

Remarks

The DataSources collection contains the name, connection string, and other information used to attach to a data provider.

Note An object of ClassType clsDatabase may have more than one data source contained in the DataSources collection, while an object of ClassType clsCube can contain only one data source in its DataSources collection. Objects of ClassType clsMiningModel and SubClassType sbclsRegular will have an OLE DB provider as a data source. For objects of ClassType clsMiningModel and SubClassType sbclsOlap, the data source is assumed to be the same as for the object where the mining model is stored.
See Also

clsCube
clsDatabase
clsMiningModel

Collections

Collection Methods

Collection Properties
Dimensions Collection

A **Dimensions** collection holds the dimension definitions for a Decision Support Objects (DSO) object. Each of the following objects contains a **Dimensions** collection:

- clsDatabase
- clsCube
- clsPartition
- clsAggregation

Each **Dimensions** collection contains dimension objects of the same class type. The **ContainedClassType** property of a **Dimensions** collection specifies the class type of the contained objects.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Contained class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregation</td>
<td>clsAggregationDimension</td>
</tr>
<tr>
<td>clsCube</td>
<td>clsCubeDimension</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>clsDatabaseDimension</td>
</tr>
<tr>
<td>clsPartition</td>
<td>clsPartitionDimension</td>
</tr>
</tbody>
</table>

**Remarks**

As with other collections in the DSO object model, the **AddNew**, **Find**, and **Remove** methods of **Dimensions** collections maintain all necessary parent-child relationships among the objects in the object model.

**See Also**

Collections
Collection Methods

Collection Properties
Levels Collection

The Levels collection holds the level definitions for a Decision Support Objects (DSO) object. Each of the following objects contains a Levels collection:

- clsDatabaseDimension
- clsCubeDimension
- clsPartitionDimension
- clsAggregationDimension

Each Levels collection contains level objects of the same class type. The ContainedClassType property of a Levels collection specifies the class type of the contained objects.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Contained class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregationDimension</td>
<td>clsAggregationLevel</td>
</tr>
<tr>
<td>clsCubeDimension</td>
<td>clsCubeLevel</td>
</tr>
<tr>
<td>clsDatabaseDimension</td>
<td>clsDatabaseLevel</td>
</tr>
<tr>
<td>clsPartitionDimension</td>
<td>clsPartitionLevel</td>
</tr>
</tbody>
</table>

Remarks

As with other collections in the DSO object model, the AddNew, Find, and Remove methods of Levels collections maintain all necessary parent-child relationships among the objects in the object model.

See Also

Collections
Collection Methods

Collection Properties

Dimension Interface
**MDStores Collection**

The MDStores collection holds objects that implement the MDStore interface. For example, a myDatabase.MDStores collection contains objects of ClassType clsCube, and a myCube. MDStores collection contains objects of ClassType clsPartition. For more information about this hierarchy, see Decision Support Objects.

Each of the following objects contains an MDStores collection:

- clsServer
- clsDatabase
- clsCube
- clsPartition

Each MDStores collection contains MDStore objects of the same class type, as noted in the following table. The ContainedClassType property of an MDStores collection specifies the class type of the contained objects.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Contained class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>clsPartition</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>clsCube</td>
</tr>
<tr>
<td>clsPartition</td>
<td>clsAggregation</td>
</tr>
<tr>
<td>clsServer</td>
<td>clsDatabase</td>
</tr>
</tbody>
</table>

**Remarks**

As with other collections in the Decision Support Objects (DSO) object model, the AddNew, Find, and Remove methods of MDStores collections maintain all necessary parent-child relationships among the objects in the object model.
See Also

Collections
Collection Methods
Collection Properties
Measures Collection

The Measures collection holds the measure definitions for a Decision Support Objects (DSO) object. Each of the following objects contains a Measures collection:

- clsCube
- clsPartition
- clsAggregation

Each Measures collection contains measure objects of the same class type. The ContainedClassType property of a Measures collection specifies the class type of the contained objects.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Contained class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsAggregation</td>
<td>clsAggregationMeasure</td>
</tr>
<tr>
<td>clsCube</td>
<td>clsCubeMeasure</td>
</tr>
<tr>
<td>clsPartition</td>
<td>clsPartitionMeasure</td>
</tr>
</tbody>
</table>

Remarks

As with other DSO collections, the AddNew, Find, and Remove methods of Measures collections maintain all necessary parent-child relationships among the objects in the object model.

See Also

Collections
Collection Methods
Collection Properties
MDStore Interface
**MemberProperties Collection**

The **MemberProperties** collection contains objects of ClassType **clsMemberProperty**. Each of the following objects contains a **MemberProperties** collection:

- **clsDatabaseLevel**
- **clsCubeLevel**
- **clsPartitionLevel**

**See Also**

- **clsAggregationLevel**
- **clsCubeLevel**
- **clsDatabaseLevel**
- **clsMemberProperty**
- **clsPartitionLevel**

**Collections**

**Collection Methods**

**Collection Properties**

**Level Interface**
MiningModels Collection

The **MiningModels** collection contains the data mining models within a database.

The following object contains a **MiningModels** collection:

- **clsDatabase**

Each **MiningModels** collection contains the data mining model objects that make up the data mining models for the entire database.

Remarks

The **ContainedClassType** property for this collection always returns **clsMiningModel**.

Example

The following example connects to an Analysis server and creates a data mining model in the **FoodMart 2000** sample database called FoodMartMiningModel:

```vba
Dim s As DSO.Server
Dim db As DSO.MDStores ' Create an interface for the FoodMart 2000 ' database.
Dim dmm As DSO.MiningModel ' Connect to the server.
Set s = New DSO.Server
s.Connect ("LocalHost")
' Get a reference to the FoodMart 2000 database.
set db = s.MDStores("FoodMart 2000")
' Create the data mining model using the AddNew method.
set dmm = db.MiningModels.AddNew("FoodMartMiningModel")
' ... Continue by setting various properties for the new object...
```
See Also

clsMiningModel
clsDatabase
clsServer
Roles Collection

The **Roles** collection holds the user role definitions for a Decision Support Objects (DSO) object. Each of the following objects contains a **Roles** collection:

- clsDatabase
- clsCube
- clsMiningModel

Each **Roles** collection contains role objects of the same class type. The **ContainedClassType** property of a **Roles** collection specifies the class type of the contained objects.

<table>
<thead>
<tr>
<th>Class type</th>
<th>Contained class type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clsCube</td>
<td>clsCubeRole</td>
</tr>
<tr>
<td>clsDatabase</td>
<td>clsDatabaseRole</td>
</tr>
<tr>
<td>clsMiningModel</td>
<td>clsMiningModelRole</td>
</tr>
</tbody>
</table>

Remarks

As with other DSO collections, the **AddNew**, **Find**, and **Remove** methods of **Roles** collections maintain all necessary parent-child relationships among the objects in the object model.

**Note** The **UsersList** property is maintained by **clsDatabaseRole**, and the **Permissions** property is maintained by **clsCubeRole**.

See Also

- **Collection Methods**
- **Collection Properties**
Collections
Permissions
UsersList
Analysis Services Programming
Add-ins

In Microsoft® SQL Server™ 2000 Analysis Services, Analysis Manager supports the integration of custom programs, referred to as add-ins, that can interact with and enhance the Analysis Manager user interface. You can create and register add-ins that will be called by the Microsoft OLAP Services Add-Ins Manager library in response to user activity in the Analysis Manager user interface. Your custom add-ins can optionally use Decision Support Objects (DSO) to manage server objects. Multiple add-ins can be registered.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Add-ins</td>
<td>General information about add-in programs</td>
</tr>
<tr>
<td>Building Add-ins</td>
<td>Step-by-step introduction to building an add-in program</td>
</tr>
<tr>
<td>Programmer's Reference (Add-ins)</td>
<td>References for the objects, properties, methods, and collections used in implementing an add-in program</td>
</tr>
</tbody>
</table>
Analysis Services Programming
About Add-ins

Microsoft® Management Console (MMC) is used by Microsoft SQL Server™ 2000 Analysis Services and other server software and services to offer a consistent user interface. Analysis Services provides a snap-in program that operates within MMC and calls the Microsoft OLAP Services Add-ins Manager library, which in turn calls registered add-ins. The Analysis Manager user interface is implemented as an add-in and is called by the Analysis Services Add-Ins Manager in the same way that your custom add-in will be called. The functionality for the Analysis Services Add-In Manager is supplied by the Microsoft OLAP Services Add-Ins Manager library, named Msmdadin.dll.

Your add-in can add nodes to the structure in the tree pane as the user selects or expands a node and can augment node menus with items that will cause your program to be called when those items are selected. Analysis Manager allows multiple custom add-ins to be registered and operating at the same time.

Some ideas for custom add-ins are:

- **Reporting tools**
  
  You can use custom add-ins to create reports on multidimensional meta data or usage-based analysis.

- **Scheduling tools**
  
  You can use custom add-ins to create scheduling tools that handle automatic routine administrative activities.

- **Maintenance tools**
  
  You can create custom add-ins to back up data and meta data.

- **Copying and transferring**
  
  You can create custom add-ins to transfer data from a multidimensional data source to a spreadsheet or other analysis tool.
Analysis Services Programming
Building Add-ins

To create a custom add-in, add Microsoft OLAP Services Add-ins Manager to the available references for your Microsoft® Visual Basic® project. This library contains the classes, objects, methods, properties, enumerations, and collections you can use in your program. For more information, see Tutorial - Creating a Sample Add-in and Programmer’s Reference (Add-ins).

Decision Support Objects (DSO) can also be used in a custom add-in. To use DSO in an add-in, add Microsoft Decision Support Objects to the available references for your project. For more information, see Using Decision Support Objects.

Your add-in can add nodes to the structure in the tree pane as the user selects or expands a node and can augment node menus with items that will cause your program to be called when those items are selected.

The Microsoft OLAP Services Add-Ins Manager library calls your custom add-in to display the objects in the Analysis Manager user interface and to respond to user activity. If your program does not implement the required IOLapAddIn interface as specified here, Analysis Manager may fail to operate as designed.

To create a Microsoft SQL Server™ 2000 Analysis Services add-in, you must:

- Create a Microsoft ActiveX® DLL project in Visual Basic, using the name of your custom add-in as the project name.

- Create one publicly exposed class that implements the IOLapAddIn interface.

- Provide your own implementation of each of the IOLapAddIn methods.

- Register your custom add-in in the registry.

See Also
Decision Support Objects

IOLapAddIn Interface
Analysis Services Programming
Tutorial - Creating a Sample Add-in

Step through the topics in this section to create a sample user interface add-in. If you work through these exercises in order, you will create a project in Microsoft® Visual Basic® that performs the following functions:

- Registers the custom add-in.
- Initializes a Visual Basic project.
- Adds code to implement a form and place new tree nodes within the Analysis Manager tree pane.
- Adds menu items to the new tree nodes.

Registering a Custom Add-in

Add-ins are registered in the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\OLAP Server\O
```

Each custom add-in contains a string value entry in this registry key and its own key in the registry as well. The string value in the `Addins` key and the `Addins` key itself must have the same name.

**CAUTION** The registry keys `DSOInfo` and `MoveRepository` are default keys created when you install Microsoft SQL Server™ 2000 Analysis Services. They should not be modified or deleted. Doing so will have adverse affects on the intended operation of Analysis Manager and may result in the loss of data.

Creating an Addins Key and Key Values

First, in Registry Editor, navigate to the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\OLAP Server\O
```
Second, create a new **String Value** entry for the **Addins** key named **OlapSampleAddIn**. Set the value of this new entry to True. This instructs the Microsoft OLAP Services Add-Ins Manager library to automatically load the add-in when Microsoft Management Console (MMC) is started. Leaving this value blank or setting it to False will prevent the add-in from loading.

Third, create a new registry key for the **OlapSampleAddIn** string value. This new key must have the same name as the registry key, that is, **OlapSampleAddIn**. You should now have a string value in the **Addins** key and a new registry key with the same name.

Fourth, select the new **OlapSampleAddIn** key and add four new string values to this key. They are listed in the following table.

<table>
<thead>
<tr>
<th>Key string value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassName</td>
<td>The project and the exposed interface class</td>
</tr>
<tr>
<td>Name</td>
<td>The name displayed on the <strong>Add-ins</strong> tab of the <strong>Properties</strong> dialog box in Analysis Manager</td>
</tr>
<tr>
<td>Description</td>
<td>The description displayed on the <strong>Add-ins</strong> tab of the <strong>Properties</strong> dialog box in Analysis Manager</td>
</tr>
<tr>
<td>Priority</td>
<td>The loading priority that the Microsoft OLAP Services Add-Ins Manager library uses to load add-ins</td>
</tr>
</tbody>
</table>

The values for each key string are listed in the following table.

<table>
<thead>
<tr>
<th>Key string value</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassName</td>
<td>OlapSampleAddIn.MyAddIn</td>
</tr>
<tr>
<td>Name</td>
<td>Sample AddIn</td>
</tr>
<tr>
<td>Description</td>
<td>Sample OLAP Manager AddIn</td>
</tr>
<tr>
<td>Priority</td>
<td>2</td>
</tr>
</tbody>
</table>

**Note** If you are providing custom add-ins to others, you will need to provide an installation procedure that describes or automatically creates the required registry entries before your add-in will function.
Creating the MyAddIn Project in Visual Basic

First, create a Microsoft ActiveX® DLL project in Visual Basic and set the name of the project to **OlapSampleAddIn**. Change the name of the publicly exposed class to **MyAddIn**. (This corresponds to the **ClassName** registry key value.)

Next, set Instancing to **5 - MultiUse** in the Properties window for the class.

Then, add **Microsoft OLAP Services Add-ins Manager** to the project references. (You must have previously installed Microsoft SQL Server™ 2000 Analysis Services for this reference to be available.)

Next, add a form to the project named **SampleForm**. Add a command button named **cmdClose** to this form.

Lastly, add the following code to the **Form** class:

```vbnet
Public Index As Integer
Private Sub cmdClose_Click()
    Me.Hide
End Sub
```

Implementing IOlapAddIn

The example code in this topic implements the **IOlapAddIn** interface. Place the following code in the Declarations section of the **MyAddIn** class:

```vbnet
Option Explicit
Implements IOlapAddIn

Const ThisAddInName = "My Sample AddIn"
Private m_SampleForms As Collection

Private Enum MenuActions
    mnuactRename = 1
    mnuactAddNewForm
    mnuactRefreshList
    mnuactDeleteSampleForm
    mnuactShowSampleForm
```
mnuactShowTop
mnuactShowCenter
mnuactShowBottom
End Enum

Private Enum SampleIcons
    icoForms = 1
    icoForm
End Enum

In the **Objects** box, click **IOLapAddIn**.

In the **Procedures** box, select each method that the **IOLapAddIn** interface provides. This creates an implementation for each method within your class.

Add the following code to the **Class_Initialize** method:

Private Sub Class_Initialize()
    On Error GoTo Initialize_Err

    Set m_SampleForms = New Collection
    Dim frmSample As New SampleForm

    frmSample.Caption = "Sample Form 1"
    frmSample.Index = 1
    m_SampleForms.Add frmSample, "Sample Form 1"

    Exit Sub

Initialize_Err:
    Debug.Assert False
    MsgBox "An Error Occurred in Class_Initialize"
    Err.Clear
Add the following code to the **IOlapAddIn_Name** property method:

```vbnet
Private Property Get IOlapAddIn_Name() As String
    IOlapAddIn_Name = ThisAddInName
End Property
```

**Adding New Tree Nodes**

The example code in this topic initializes your custom add-in to add new nodes to the tree node display. You must implement this code before the examples later in this section will work.

Add the following code to the **IOlapAddIn_ProvideChildNodes** method:

```vbnet
Private Sub IOlapAddIn_ProvideChildNodes( _
    ParentNode As DSSAddInsManager.OlapTreeNode, _
    OlapTreeNodes As DSSAddInsManager.OlapTreeNodes)
On Error GoTo IOlapAddIn_ProvideChildNodes_Err
If ParentNode.Caption = "Analysis Servers" Then
    OlapTreeNodes.Add "Sample Forms", icoForms
ElseIf ParentNode.Caption = "Sample Forms" Then
    Dim frm As Form
    For Each frm In m_SampleForms
        OlapTreeNodes.Add frm.Caption, icoForm
    Next
End If
Exit Sub
IOlapAddIn_ProvideChildNodes_Err:
    Debug.Assert False
    MsgBox "Provide Child Nodes Failed"
    Err.Clear
Exit Sub
```
End Sub

Run the application.

With the Visual Basic project executing, start Analysis Manager, and then browse the tree pane. You should see the new tree nodes added to the bottom of the tree.

**Adding New Menu Items**

The example code in this step adds new menu items to the tree nodes that were added in the previous exercise. You must implement this code before the examples later in this section will work.

Add the following code to the `IOlapAddIn_ProvideMenuItems` method:

```vbnet
Private Sub IOlapAddIn_ProvideMenuItems( _
   CurrentNode As DSSAddInsManager.OlapTreeNode, _
   MenuItems As DSSAddInsManager.OlapMenuItems)
On Error GoTo IOlapAddIn_ProvideMenuItems_Err
Dim iFlags As OlapMenuFlags

' The Microsoft OLAP Services Add-Ins Manager calls this method
' each time a tree node is accessed.
' Because multiple add-ins are supported,
' verify that this is the correct tree node to respond to.
If CurrentNode.OwnerAddInName <> ThisAddInName Then Exit Sub

Select Case CurrentNode.Caption
Case "Sample Forms"
   MenuItems.Add mnuStandard, "&Form", _
       mnuactAddNewForm, , mnuflagNew
   MenuItems.Add mnuStandard, "&Refresh", _
       mnuactRefreshList, , mnuflagNew

Case Else
```

```vbnet```
Run the application.

With the Visual Basic project executing, start Analysis Manager, and then browse the tree pane. Right-click one of the new tree nodes, and then examine the added menu items.

**Responding to Menu Item Selection**

The example code in this topic adds code to respond to user selection of the menu items added in the previous example.
Add the following method to the MyAddIn class:

Private Function IsNameUsed(szName As String, _
    col As Collection) As Boolean
    On Error GoTo IsNameUsed_Err
    Dim vTmp As Variant
    Set vTmp = col(szName)
    IsNameUsed = True
    Exit Function
IsNameUsed_Err:
    IsNameUsed = False
    Err.Clear
    Exit Function
End Function

Add the following code to the IOlapAddIn_ExecuteMenuItems method:

Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItem As DSSAddInsManager.OlapMenuItem) _
    As DSSAddInsManager.RefreshTreeTypes
    On Error GoTo IOlapAddIn_ExecuteMenuItem_Err
    Dim frmSample As SampleForm
    Dim szFormCaption As String
    Dim szNodeCaption As String
    Dim iFormIndex As Integer

    szNodeCaption = CurrentNode.Caption

    Select Case MenuItem.Key
        Case mnuactRename
            Dim szName As String
            Dim tmpForm As SampleForm
            Set tmpForm = m_SampleForms(szNodeCaption)
            tmpForm.Name = szName
    End Select

    Exit Function
IOlapAddIn_ExecuteMenuItem_Err:
    Err.Clear
    Exit Function
End Function
m_SampleForms.Remove szNodeCaption

Do
    szName = InputBox("Please enter the new name:", _
        "Rename a Form", szNodeCaption)
    If Len(szName) = 0 Then
        MsgBox _
            "The name must not be a zero length string", _
                vbExclamation, "Invalid Name"
    Else
        Exit Do
    End If
Loop

tmpForm.Caption = szName

If tmpForm.Index <= m_SampleForms.Count Then
    m_SampleForms.Add tmpForm, szName, tmpForm.Index
Else
    ' This is the only item in the list
    ' or it was at the end of the list.
    ' No need to specify a before value
    m_SampleForms.Add tmpForm, szName
End If

' Manually tell the Microsoft OLAP Services Add-Ins Manager to
' refresh the tree
IOlapAddIn_ExecuteMenuItem = refTreeCurrentAndBelow

Case mnuactShowTop
    Set frmSample = m_SampleForms(szNodeCaption)
    frmSample.Move (Screen.Width - frmSample.Width) / 2, 0
frmSample.Show vbModal

Case mnuactShowCenter
    Set frmSample = m_SampleForms(szNodeCaption)
    frmSample.Move (Screen.Width - frmSample.Width) / 2, _
                   (Screen.Height - frmSample.Height) / 2
    frmSample.Show vbModal

Case mnuactShowBottom
    Set frmSample = m_SampleForms(szNodeCaption)
    frmSample.Move (Screen.Width - frmSample.Width) / 2, _
                    Screen.Height - frmSample.Height
    frmSample.Show vbModal

Case mnuactAddNewForm
    Set frmSample = New SampleForm
    iFormIndex = m_SampleForms.Count

    Do
        iFormIndex = iFormIndex + 1
        szFormCaption = "Sample Form " & iFormIndex
    Loop While IsNameUsed(szFormCaption, m_SampleForms)

    frmSample.Caption = szFormCaption
    frmSample.Index = iFormIndex

    m_SampleForms.Add frmSample, szFormCaption

    Set frmSample = Nothing

    ' Tell Microsoft OLAP Services Add-Ins Manager to refresh the tre
    IOlapAddIn_ExecuteMenuItem = reftreeCurrentAndBelow
Case mnuactDeleteSampleForm
    m_SampleForms.Remove szNodeCaption

Case mnuactRefreshList
    IOlapAddIn_ExecuteMenuItem = refTreeCurrentAndBelow

    End Select
    Exit Function

IOlapAddIn_ExecuteMenuItem_Err:
    Debug.Assert False
    MsgBox "Execute Menu Item Failed"
    Err.Clear
    Exit Function
    End Function

Run the application.

With the Visual Basic project executing, start Analysis Manager, and then browse the tree pane. Right-click one of the new tree nodes, and then click a menu item.
Analysis Services Programming
Example - Report Add-in

The following example shows how to create an add-in that incorporates Decision Support Objects (DSO) functionality. A pop-up menu item named List is added to the server tree node with options to list Database, Cube, Dimension, and Level objects. Before you can use this example code, perform the following steps:

1. In Microsoft® Visual Basic®, create a Microsoft ActiveX® DLL project. Name the project ReportAddIn and the publicly exposed class ReportClass. Ensure that references have been added for Microsoft OLAP Services Add-Ins Manager and Microsoft Decision Support Objects.

2. Register the add-in.

3. Add a form to the project and name it ReportForm. Include a ListBox control named ObjectList.

For more information, see Tutorial - Creating a Sample Add-in.

Place the following code into the ReportClass class:

```
Option Explicit
Implements IOlapAddIn

Private dsoServer As DSO.Server 'DSO Server object
Private frmReport As ReportForm
Const OLAPManagerName = "OLAP Manager"
Const ThisAddInName = "ReportAddIn"

Private Enum MenuItems
    mnuParentMenuItem = 1
    mnuListDatabase
    mnuListCube
```
Private Sub Class_Initialize()
    Set frmReport = New ReportForm
End Sub

Private Function IOlapAddIn_ExecuteMenuItem(_
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItem As DSSAddInsManager.OlapMenuItem) _
    As DSSAddInsManager.RefreshTreeTypes
Dim dsoDB As DSO.MDStore   'Database
Dim dsoCube As MDStore     'Cube
Dim dsoDim As DSO.Dimension 'Dimension
Dim dsoLev As DSO.Level    'Level
Dim DBCounter As Integer
Dim CubeCounter As Integer
Dim DimCounter As Integer
Dim LevCounter As Integer

Select Case MenuItem.Key
    Case mnuListDatabase    'List database objects
        frmReport.Caption = "Database Objects"
        For DBCounter = 1 To dsoServer.MDStores.Count
            Set dsoDB = dsoServer.MDStores(DBCounter)
            frmReport.ObjectList.AddItem dsoDB.Name
        Next DBCounter

    Case mnuListCube       'List cube objects
        frmReport.Caption = "Cube Objects"
        For DBCounter = 1 To dsoServer.MDStores.Count
Set dsoDB = dsoServer.MDStores(DBCounter)
frmReport.ObjectList.AddItem dsoDB.Name
For CubeCounter = 1 To dsoDB.MDStores.Count
  Set dsoCube = dsoDB.MDStores(CubeCounter)
  frmReport.ObjectList.AddItem " " & dsoCube.Name
Next CubeCounter
Next DBCounter

Case mnuListDimension  'List dimension objects
  frmReport.Caption = "Dimension Objects"
  For DBCounter = 1 To dsoServer.MDStores.Count
    Set dsoDB = dsoServer.MDStores(DBCounter)
    frmReport.ObjectList.AddItem dsoDB.Name
    For CubeCounter = 1 To dsoDB.MDStores.Count
      Set dsoCube = dsoDB.MDStores(CubeCounter)
      frmReport.ObjectList.AddItem " " & dsoCube.Name
      For DimCounter = 1 To dsoCube.Dimensions.Count
        Set dsoDim = dsoCube.Dimensions(DimCounter)
        frmReport.ObjectList.AddItem " " & dsoDim.Name
      Next DimCounter
    Next CubeCounter
  Next DBCounter

Case mnuListLevel     'List level objects
  frmReport.Caption = "Level Objects"
  For DBCounter = 1 To dsoServer.MDStores.Count
    Set dsoDB = dsoServer.MDStores(DBCounter)
    frmReport.ObjectList.AddItem dsoDB.Name
    For CubeCounter = 1 To dsoDB.MDStores.Count
      Set dsoCube = dsoDB.MDStores(CubeCounter)
      frmReport.ObjectList.AddItem " " & dsoCube.Name
      For DimCounter = 1 To dsoCube.Dimensions.Count
        Set dsoDim = dsoCube.Dimensions(DimCounter)
        frmReport.ObjectList.AddItem " " & dsoDim.Name
      Next DimCounter
    Next CubeCounter
  Next DBCounter
Set dsoDim = dsoCube.Dimensions(DimCounter)
frmReport.ObjectList.AddItem " " & _
    dsoDim.Name
For LevCounter = 1 To dsoDim.Levels.Count
    Set dsoLev = dsoDim.Levels(LevCounter)
    frmReport.ObjectList.AddItem _
    " " & dsoLev.Name
Next LevCounter
Next DimCounter
Next CubeCounter
Next DBCounter

End Select

'Display the form
frmReport.Show

End Function

Private Function IOlapAddIn_GetObject(_
    LinkedNode As DSSAddInsManager.OlapTreeNode) As Object

End Function

Private Property Get IOlapAddIn_Name() As String
    IOlapAddIn_Name = ThisAddInName
End Property

Private Sub IOlapAddIn_ProvideChildNodes(_
    ParentNode As DSSAddInsManager.OlapTreeNode, _
    OlapTreeNodes As DSSAddInsManager.OlapTreeNodes)
    'No child nodes needed
End Sub
Private Sub IOlapAddIn_ProvideHTML(_
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    CurrentURL As String)
' If custom HTML pages are needed -
' CurrentURL = '{custom.htm}'
End Sub

Private Function IOlapAddIn_ProvideIcon(Index As Integer) _
    As stdole.OLE_HANDLE
' No icons needed
End Function

Private Sub IOlapAddIn_ProvideMenuItems(_
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItems As DSSAddInsManager.OlapMenuItems)
Dim iFlags As OlapMenuFlags

If CurrentNode.OwnerAddInName = OLAPManagerName Then
    ' Do we have as server?
    If CurrentNode.LinkedObject.ClassType = clsServer Then
        Set dsoServer = CurrentNode.LinkedObject

        ' If not connected to server, disable menu item
        If dsoServer.State = stateConnected Then
            iFlags = mnuflagRegular + mnuflagPopup
        Else
            iFlags = mnuflagGrayed + mnuflagPopup
        End If

        ' Add popup menu item
        MenuItems.Add mnuSeparator
        MenuItems.Add mnuStandard, "&List", _
    End If
End Sub
mnuParentMenuItem, , iFlags

'Add popup menu child menu items
MenuItems.Add mnuStandard, "&Database", _
    mnuListDatabase, mnuParentMenuitem, mnuflagSubmenu
MenuItems.Add mnuStandard, "&Cube", _
    mnuListCube, mnuParentMenuitem, mnuflagSubmenu
MenuItems.Add mnuStandard, "&Dimension", _
    mnuListDimension, mnuParentMenuitem, mnuflagSubmenu
MenuItems.Add mnuStandard, "&Level", _
    mnuListLevel, mnuParentMenuitem, mnuflagSubmenu
    End If
    End If
    End Sub
Analysis Services Programming
Programmer's Reference (Add-ins)

The Microsoft OLAP Services Add-Ins Manager library, Msmdadin.dll, contains the classes, objects, methods, properties, enumerations, and collections you can use in your program.

**CAUTION** It is important that your program use only those elements of the Microsoft OLAP Services Add-Ins Manager library that are documented here. The use of undocumented library elements that may be exposed in the Object Browser can cause indeterminate results and possible loss of data.

The Microsoft OLAP Services Add-Ins Manager library calls your custom add-in as it displays objects, such as tree nodes and menu items, in the Analysis Manager user interface, and in response to user activity with any object, including objects your custom add-in did not create, in Analysis Manager. Therefore, your custom add-in must implement the required `IOlapAddIn` interface, as specified in this section, or Analysis Manager may fail to operate as designed.

To create a custom add-in, add **Microsoft OLAP Services Add-Ins Manager** to the available references for your Microsoft® Visual Basic® project.

The following topics further detail the elements of the Microsoft OLAP Services Add-Ins Manager library.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
<td>Discusses the <code>IOlapAddIn</code> interface, including its properties and methods</td>
</tr>
<tr>
<td><strong>Objects</strong></td>
<td>Covers the <code>OlapMenuItem</code> and <code>OlapTreeNode</code> objects, including their properties</td>
</tr>
<tr>
<td><strong>Enumerations</strong></td>
<td>Details several enumerations used by the rest of the library</td>
</tr>
<tr>
<td><strong>Collections</strong></td>
<td>Provides information on the properties and methods of the <code>OlapMenuItems</code> and <code>OlapTreeNodes</code> collections</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Interfaces

Although the Microsoft OLAP Services Add-Ins Manager library includes several interfaces, only one needs to be implemented in order for you to take advantage of the functionality offered by the library.

The `IOlapAddIn` interface is implemented by all custom applications for Analysis Manager. Many of the methods for the `IOlapAddIn` interface involving menu items and tree nodes use the `OlapMenuItem` objects to supply access to the properties associated with a menu item and the `OlapTreeNode` object to supply access to the properties associated with a tree node in Analysis Manager.

See Also

`IOlapAddIn Interface`
IOlapAddIn Interface

The IOlapAddIn interface supports the integration of custom add-in applications with Analysis Manager. You can use this interface in conjunction with Decision Support Objects (DSO) to create customized applications for managing DSO objects and controlling the server. This interface requires methods and properties. There are no collections exposed by this interface, although several methods employ collections as arguments.

**CAUTION**  This interface must be implemented as specified or your add-in (and other add-ins, including Analysis Manager) may not operate correctly. It is possible for data to be corrupted or lost as a result of incorrect implementation of the IOlapAddIn interface.

The **Class_Initialize** subroutine of the class module in which you have implemented the IOlapAddIn interface is called before any methods of your program are called. After initialization, the Microsoft OLAP Services Add-Ins Manager library can call the methods of your implementation of the IOlapAddIn interface any number of times and in any sequence.

See Also

[Methods, IOlapAddIn Interface](#)

[Properties, IOlapAddIn Interface](#)
Methods, IOlapAddIn Interface

The IOlapAddIn interface requires you to implement the following methods.

**Note** The syntax descriptions for the methods of the IOlapAddIn interface are shown from the viewpoint of the interface, not from the viewpoint of a class module implementing the interface. All code examples, however, are shown from the viewpoint of a class module implementing the IOlapAddIn interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteMenuItem</td>
<td>Carries out a command in response to a user action</td>
</tr>
<tr>
<td>GetObject</td>
<td>Returns a reference to the object that is represented by an OlapTreeNode</td>
</tr>
<tr>
<td>ProvideChildNodes</td>
<td>Populates an OlapTreeNodes collection so that these nodes can be displayed in the Analysis Manager tree pane</td>
</tr>
<tr>
<td>ProvideHTML</td>
<td>Provides the URL for the HTML pane when the user clicks a new node in the tree pane</td>
</tr>
<tr>
<td>ProvideIcon</td>
<td>Specifies the numeric ID for the icons to display when the user selects a node</td>
</tr>
<tr>
<td>ProvideMenuItems</td>
<td>Populates a collection of OlapMenuItems for a node in the tree pane</td>
</tr>
</tbody>
</table>

**See Also**

IOlapAddIn Interface
ExecuteMenuItem (IOlapAddIn Interface)

The **ExecuteMenuItem** method of the **IOlapAddIn** interface is called when the user clicks a menu item. Provide code in this method to respond to the user's actions.

**Syntax**

Function **ExecuteMenuItem**(*CurrentNode* As **OlapTreeNode**, *MenuItem* As **OlapMenuItem**) As **RefreshTreeTypes**

*CurrentNode*

The node that is currently selected in the Analysis Manager tree pane.

*MenuItem*

The menu item that the user clicked.

**Remarks**

By querying the properties of the **CurrentNode** and **MenuItem** objects, your application can determine which menu item the user clicked and respond accordingly. The function returns a constant from the **RefreshTreeTypes** enumeration.

**Example**

The following example shows how to execute a menu item based upon the caption of the node that is currently selected in the tree view:

```vba
Private Function IOlapAddIn_ExecuteMenuItem(_
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItem As DSSAddInsManager.OlapMenuItem) As DSSAddInsManager.RefreshTreeTypes
On Error GoTo ExecuteMenuItem_Err 'Handle errors
Select Case CurrentNode.Caption
```
Case "Add"
   Select Case MenuItem.Key
      Case mnuactAddItem1
         'Code to add item 1
      Case mnuactAddItem2
         'Code to add item 2
      Case mnuactAddItem3
         'Code to add item 3
   End Select

Case "Edit"
   Select Case MenuItem.Key
      Case mnuactEditItem1
         'Code to edit item 1
      Case mnuactEditItem1
         'Code to edit item 2
      Case mnuactEditItem1
         'Code to edit item 3
   End Select
   End Select

   Exit Function
ExecuteMenuItem_Err:
   MsgBox "ExecuteMenuItem Failed"
   Err.Clear
End Function

See Also

IOLapAddIn Interface
RefreshTreeTypes Enumeration
GetObject (IOlapAddIn Interface)

The **GetObject** method of the **IOlapAddIn** interface responds to a request that your add-in return a reference to the object that is represented by an **OlapTreeNode** object.

**Syntax**

Function **GetObject(LinkedNode As OlapTreeNode) As Object**

*LinkedNode*

The **OlapTreeNode** object to be linked with an object.

**Remarks**

An **OlapTreeNode** object can have other objects linked to it. The **GetLinkedObject** method of the **OlapTreeNode** object can be used to retrieve these linked objects.

**Example**

The following example retrieves an item from the **OlapTreeNode** collection:

'm_MyObjects is a collection of objects that are represented
'in the Analysis Manager tree pane as members of the OlapTreeNodes
'collection. Your add-in will need to populate this
'collection with objects.

Private m_MyObjects As New Collection
Private Function IOlapAddIn_GetObject(_
    LinkedNode As DSSAddInsManager.OlapTreeNode) As Object
On Error Resume Next  ' Handle error when it happens
Set IOlapAddIn_GetObject = m_MyObjects(LinkedNode.Caption)
If Err Then Err.Clear 'Item was not found in the collection
End Function

See Also

IOLapAddIn Interface
**ProvideChildNodes (IOlapAddIn Interface)**

The **ProvideChildNodes** method of the **IOlapAddIn** interface adds nodes under existing nodes to the Analysis Manager tree pane.

**Syntax**

Sub **ProvideChildNodes**(*ParentNode As OlapTreeNode, OlapTreeNodes As OlapTreeNodes*)

*ParentNode*

The **OlapTreeNode** object that is currently selected or being expanded in the tree pane display.

*OlapTreeNodes*

An empty collection of **OlapTreeNode** objects. The add-in populates this collection with the child **OlapTreeNode** objects to be created under *ParentNode*.

**Remarks**

This method responds to a request that the add-in populate an **OlapTreeNodes** collection. This collection is made of nodes that belong to the **OlapTreeNode** object that is currently selected or being expanded. Your add-in uses this method to add nodes to the tree pane. An add-in uses this method only if it needs to add **OlapTreeNode** objects to the tree pane.

**Example**

The following code illustrates how to use this method to add a node to the **OlapTreeNodes** collection based upon the caption of the parent node:

'**Declarations**

Private Enum SampleIcons 'Icons for tree nodes
  icoForm1 = 1
Private Sub IOlapAddIn_ProvideChildNodes( _
    ParentNode As DSSAddInsManager.IOlapTreeNode, _
    TreeNodes As DSSAddInsManager.OlapTreeNodes)
On Error GoTo pc_Err 'Handle errors
If ParentNode.Caption = "Analysis Servers" Then
    TreeNodes.Add "Sample Form1", icoForm1
Else
    TreeNodes.Add "Sample Form2", icoForm2
End If
Exit Sub
pc_Err:
    MsgBox "ProvideChildNodes Failed"
    Err.Clear
End Sub

See Also

IOlapAddIn Interface
ProvideHTML (IOlapAddIn Interface)

The **ProvideHTML** method of the **IOlapAddin** interface provides the URL for the HTML pane in Analysis Manager when the user selects a new node in the tree pane.

**Syntax**

Sub **ProvideHTML** (*CurrentNode* As **OlapTreeNode**, *CurrentURL* As **String**)

*CurrentNode*

The **OlapTreeNode** that is currently selected

*CurrentURL*

The source URL

**Remarks**

The *CurrentURL* variable initially contains the URL for the HTML file that is currently displayed. If there is no need to display a different HTML file, the method can exit. Otherwise, set the *CurrentURL* parameter to the URL for Analysis Manager to display.

**Example**

The following example uses this method to display the contents of a URL if the parent node's caption is Sample Forms:

Private Sub IOlapAddIn_ProvideHTML(*CurrentNode* As DSSAddIns!
           On Error GoTo IOlapAddIn_ProvideHTML_Err

    'Check to see whether the provided node is owned by another add-in
    If *CurrentNode*.OwnerAddInName <> ThisAddInName Then
        'Work with node owned by another add-in
        Exit Sub

    'Set *CurrentURL* to the URL for Analysis Manager to display
    *CurrentURL* = "http://example.com/analysis.html"
End If
' This add-in owns the node

' Assume that the files form1.htm and form2.htm exist
If CurrentNode.Caption = "Sample Forms" Then
    CurrentURL = App.Path & ":\form1.htm"
Else
    CurrentURL = App.Path & ":\form2.htm"
End If

Exit Sub
IOlapAddIn_ProvideHTML_Err:
    Debug.Assert False
    MsgBox "ProvideHTML method failed."
    Err.Clear
    Exit Sub
End Sub

See Also

IOlapAddIn Interface
**ProvideIcon (IOlapAddIn Interface)**

The **ProvideIcon** method of the **IOlapAddIn** interface is called by the Microsoft OLAP Services Add-Ins Manager library to load an available icon from a resource file.

**Syntax**

Function **ProvideIcon(Index As Integer) As OLE_HANDLE**

**Index**

The requested index for the icon resource.

**Remarks**

This method responds to a request for an OLE_HANDLE that contains the handle of an icon to be used for a tree node in Analysis Manager. If an icon handle is supplied, the icon is then loaded for use in the tree pane. If no icon handle is supplied, the icon is then loaded from a default resource file.

**Example**

The following code adds an icon to the tree pane:

Private Const FirstIconID = 1
Private Const LastIconID = 4
'Other code

Private Function IOlapAddIn_ProvideIcon(Index As Integer) _
As stdole.OLE_HANDLE
On Error GoTo ProvideIcon_Err 'Handle errors
If Index >= FirstIconID Or Index <= LastIconID Then
    IOlapAddIn_ProvideIcon = LoadResPicture(Index, vbResIcon)
End If
Exit Function
ProvideIcon_Err:
    MsgBox "ProvideIcon failed"
    Err.Clear
End Function

See Also

IOLapAddIn Interface
ProvideMenuItems (IOlapAddIn Interface)

The **ProvideMenuItems** method of the **IOlapAddIn** interface enables default menu items and adds new menu items to the current tree node.

**Syntax**

Sub **ProvideMenuItems**\(\text{CurrentNode} \text{ As } \text{OlapTreeNode}, \text{MenuItems} \text{ As } \text{OlapMenuItems}\)

**CurrentNode**

The **OlapTreeNode** object that is selected for menu display in the tree pane when users right-click.

**MenuItems**

A collection of **OlapMenuItem** objects.

**Remarks**

This method responds when the calling subroutine sends a request for default menu items to be enabled or for new menu items to be added. When your add-in enables default menu items, it should also provide associated child menu items.

**Note**  You should initialize the enumerations provided for menu items and menu actions added to **MenuItems** with a positive, nonzero value. Enumerations initialized to zero may cause unpredictable results when this method provides menu items for a custom add-in.

**Example**

The following example enables a new default menu item:

Private Enum MenuActions
    mnuActTop = 1
    mnuActMid
    mnuActBtm
End Enum

Private Sub IOlapAddIn_ProvideMenuItems(CurrentNode As _
    DSSAddInsManager.OlapTreeNode, MenuItems As _
    DSSAddInsManager.OlapMenuItems)
On Error GoTo ProvideMenuItems_Err 'Handle errors
If CurrentNode.Caption = "Node 1" Then
    'Enable default new menu item and add child menu items
    MenuItems.Add mnuStandard, "&Top", mnuActTop, , mnuflagNew
    MenuItems.Add mnuStandard, "&Mid", mnuActMid, , mnuflagNew
    MenuItems.Add mnuStandard, "&Btm", mnuActBtm, , mnuflagNew
    'Add regular menu item to root menu
    MenuItems.Add mnuStandard, "&Special", mnuActSpc, , mnuflagR
End If
Exit Sub
ProvideMenuItems_Err:
    MsgBox "ProvideMenuItems failed"
    Err.Clear
End Sub

See Also

IOlapAddIn Interface
Properties, IOlapAddIn Interface

The IOlapAddIn interface requires you to provide one property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the add-in</td>
</tr>
</tbody>
</table>

See Also

IOlapAddIn Interface
Analysis Services Programming

Name (IOlapAddIn Interface)

The **Name** property of the **IOlapAddIn** interface returns the name of your add-in to the calling program. The value of this property is used to identify the program that provides objects to the Microsoft OLAP Services Add-Ins Manager library.

**Data Type**

String

**Example**

The following example returns the name of the custom add-in:

Private ThisAddInName = "MySampleAddIn"
'More code

Private Property Get IOlapAddIn_Name() As String
   On Error Resume Next 'Defer errors
   IOlapAddIn_Name = ThisAddInName
   Err.Clear 'Clears errors if any occurred
End Property
Analysis Services Programming
Objects

The Microsoft OLAP Services Add-Ins Manager library includes several objects; only two are directly exposed by the IOlapAddIn interface.

The OlapMenuItem object is used by several methods in the IOlapAddIn interface to provide access to the properties of menu items in Analysis Manager. Similarly, the OlapTreeNode object is also used by several methods in the IOlapAddIn interface to provide access to the properties of tree nodes in Analysis Manager.

The following topics detail the properties exposed by these two objects.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlapMenuItem</td>
<td>Details the properties available to the OlapMenuItem object</td>
</tr>
<tr>
<td>OlapTreeNode</td>
<td>Details the properties and events available to the OlapTreeNode object</td>
</tr>
</tbody>
</table>

See Also

IOlapAddIn Interface
Analysis Services Programming

**OlapMenuItem**

The **OlapMenuItem** object contains the properties of a menu item in Analysis Manager. There are no collections or methods associated with this object.

**See Also**

Properties, OlapMenuItem
Properties, OlapMenuItem

An **OlapMenuItem** object contains the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Caption</strong></td>
<td>The menu item caption.</td>
</tr>
<tr>
<td><strong>Disabled</strong></td>
<td>Indicates whether the menu item is disabled.</td>
</tr>
<tr>
<td><strong>Flags</strong></td>
<td>Flags that describe the actual state of the <strong>OlapMenuItem</strong> object.</td>
</tr>
<tr>
<td><strong>HelpContextId</strong></td>
<td>The Help context ID (optional).</td>
</tr>
<tr>
<td><strong>HelpFileName</strong></td>
<td>The Help file name (optional).</td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td>The user-defined value assigned to the menu item.</td>
</tr>
<tr>
<td><strong>OwnerAddInName</strong></td>
<td>The name of the add-in associated with the menu item. It is set automatically by the Microsoft OLAP Services Add-Ins Manager library.</td>
</tr>
<tr>
<td><strong>OwnerAddInProgId</strong></td>
<td>The program ID of the add-in associated with the menu item. It is set automatically by the Microsoft OLAP Services Add-Ins Manager library.</td>
</tr>
<tr>
<td><strong>ParentKey</strong></td>
<td>The parent key value used to associate a child menu item with a parent or owner pop-up menu item.</td>
</tr>
</tbody>
</table>
Caption (OlapMenuItem)

The Caption property of an OlapMenuItem object contains the caption exposed in the menu for the node in the tree pane.

Data Type

String

Access

Read/write

Example

The following example adds a Form menu item and tests the caption property of a selected item to determine whether it is this same Form menu item:

'Custom add-in
Private Enum MenuActions
    mnuActAddNewForm
    mnuActRefreshForm
    mnuActDeleteForm
End Enum

Private Sub IOlapAddIn_ProvideMenuItems( _
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItems As DSSAddInsManager.OlapMenuItems)
On Error Resume Next 'Handle errors
'MenuItems is a collection for MenuItem objects
'Some more code...
'Menu item Form is added as a child of the standard New menu item
MenuItems.Add mnuStandard, "&Form", mnuActAddNewForm,,mnu
End Sub
'Later
Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.IOlapTreeNode, _
    MenuItem As DSSAddInsManager.IOlapmenuItem) _
    As DSSAddInsManager.RefreshTreeTypes
On Error GoTo ExecuteMenuItem_Err 'Handle errors
'Some code
If MenuItem.Caption = "Form" Then
    'Code to handle New/Form menu selection
End If
Exit Function
ExecuteMenuItem_Err:
    MsgBox "ExecuteMenuItem Failed"
    Err.Clear
End Function

See Also
OlapMenuItem
Analysis Services Programming

**Disabled (OlapMenuItem)**

The **Disabled** property of an **OlapMenuItem** object determines whether the menu item is enabled or disabled.

**Data Type**

Boolean

**Access**

Read/write

**Remarks**

This property determines the visible state of the tree node menu item. If disabled, the menu item appears dimmed.

**Example**

The following example displays a message box if the selected menu item is disabled:

```vba
Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.IOlapTreeNode, _
    MenuItem As DSSAddInsManager.IOlapmenuItem) As DSSAddInsManager.RefreshTreeTypes
On Error GoTo ExecuteMenuItem_Err 'Handle errors
'Some more code
If MenuItem.Disabled = TRUE Then
    MsgBox "Disabled: Cannot execute at this time"
End If
Exit Function
_ExecuteMenuItem_Err:
MsgBox "ExecuteMenuItem failed"
```
Err.Clear
End Function

See Also

OlapMenuItem
Flags (OlapMenuItem)

The Flags property of an OlapMenuItem object determines the state of the menu item, such as whether the item is enabled, the place of the item in the menu hierarchy, and so on.

Data Type

Integer representing a bitmask of constants in the OlapMenuFlags enumeration.

Access

Read/write

Remarks

Values can be combined from the OlapMenuFlags enumeration using a bitwise OR operator.

Example

The following example adds a menu item, marks it as new, and ensures that it is unchecked:

'Custom add-in
Private Enum MenuActions
  mnuActAddNewForm
  mnuActRefreshForm
  mnuActDeleteForm
End Enum

Private Sub IOlapAddIn_ProvideMenuItems(_
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItems As DSSAddInsManager.OlapMenuItems) On Error GoTo ProvideMenuItems_Err 'Handle errors
'MenuItems is a collection for MenuItem objects
'Some more code
'Menu item Form is added as a child of the standard New menu item
MenuItems.Add mnuStandard, "&Form", mnuActAddNewForm, , _
    mnuflagNew + NOT mnuflagChecked
Exit Sub
ProvideMenuItems_Err:
    MsgBox "ProvideMenuItems Failed"
    Err.Clear
End Sub

'Later
Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.IOlapTreeNode, _
    MenuItem As DSSAddInsManager.IOlapmenuItem) As DSSAddInsManager.refreshTreeTypes
On Error GoTo ExecuteMenuItem_Err 'Handle errors
If MenuItem.Flag AND NOT mnuflagChecked Then
    'menu item unchecked, execute code
    Exit Function
End If
    Exit Function
ExecuteMenuItem_Err:
    MsgBox "ExecutemenuItem Failed"
    Err.Clear
End Function

See Also

OlapMenuFlags
OlapMenuItem
HelpContextId (OlapMenuItem)

The HelpContextId property of an OlapMenuItem object contains the Help context ID number associated with the item.

Data Type

Long

Access

Read/write

Remarks

This property returns or sets an associated context ID number for an object. It is used to provide context-sensitive Help for your application. You must provide the Help file and identify it using the HelpFileName property. For more information about the HelpContextID property, see the Microsoft® Visual Basic® documentation.

Example

Private Enum MenuActions
  mnuActTop
  mnuActMid
  mnuActBtm
  mnuActSpc
End Enum

Private Sub IOlapAddIn_ProvideMenuItems(CurrentNode _
  As DSSAddInsManager.OlapTreeNode, MenuItems _
  As DSSAddInsManager.OlapMenuItems)
On Error GoTo ProvideMenuItems_Err 'Handle errors
If CurrentNode.Caption = "Node 1" Then
'Enable default New menu item and add child menu items
MenuItems.Add mnuStandard, "&Top", mnuActTop, , mnuflagNew
MenuItems.Add mnuStandard, "&Mid", mnuActMid, , mnuflagNew
MenuItems.Add mnuStandard, "&Btm", mnuActBtm, , mnuflagNew
'Add regular menu item to root menu
MenuItems.Add mnuStandard, "&Special", mnuActSpc, , _
    mnuflagRegular
MenuItems.HelpContextID = 100
End If
Exit Sub

ProvideMenuItems_Err:
    MsgBox "ProvideIcon failed"
    Err.Clear
End Function

See Also

HelpFileName
OlapMenuItem
HelpFileName (OlapMenuItem)

The HelpFileName property of an OlapMenuItem object specifies the name of the Help file that contains the Help topic identified by the HelpContextID property. You must provide the Help file.

Data Type

String

Access

Read/write

See Also

HelpContextID

OlapMenuItem
Key (OlapMenuItem)

The **Key** property of an **OlapMenuItem** object contains a value specified by your add-in that uniquely identifies the menu item.

**Data Type**

Long

**Access**

Read/write

**Remarks**

This property is a user-defined value assigned by the add-in.

**Example**

Private Enum MenuActions
    mnuActAddNewForm
    mnuActRefreshForm
    mnuActDeleteForm
End Enum

Private Sub IOlapAddIn_ProvideMenuItems(_
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItems As DSSAddInsManager.OlapMenuItems)
On Error GoTo ProvideMenuItems_Err 'Handle errors
'MenuItems is a collection for MenuItem objects
'Some more code
'Menu item Form is added as a child of the standard New menu item
'mnuActAddNewForm is our Key value
MenuItems.Add mnuStandard, "&Form", mnuActAddNewForm,,mnu
Exit Sub
ProvideMenuItems_Err:
    MsgBox "ProvideMenuItems Failed"
    Err.Clear
End Sub

'Later
Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.IOlapTreeNode, _
    MenuItem As DSSAddInsManager.IOlapmenuItem) _
    As DSSAddInsManager.RefreshTreeTypes
On Error GoTo ExecuteMenuItem_Err 'Handle errors
'some code
'Check the Key value
If MenuItem.Key = mnuActAddNewForm Then
    'code to add a new form
End If
Exit Function
ExecuteMenuItem_Err:
    MsgBox "ExecuteMenuItem Failed"
    Err.Clear
End Function

See Also
OlapMenuItem
OwnerAddInName (OlapMenuItem)

The OwnerAddInName property of an OlapMenuItem object contains the name of the add-in that owns the menu item.

Data Type

String

Access

Read/write

Remarks

This property is set automatically by the Microsoft OLAP Services Add-ins Manager library. You can use this property to identify the items your program owns and the programs that own other items.

Example

The following example prints OwnerAddInName property in a message box:

Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItem As DSSAddInsManager.OlapMenuItem) _
As DSSAddInsManager.RefreshTreeTypes
On Error Resume Next 'Handle errors
If MenuItem.Caption = "&Special" Then
    MsgBox "OwnerAddInName is: " & MenuItem.OwnerAddInName
End If
End Function

See Also
OlapMenuItem
OwnerAddInProgID (OlapMenuItem)

The *OwnerAddInProgID* property of an *OlapMenuItem* object contains the program ID of the add-in that owns the menu item.

**Data Type**

String

**Access**

Read/write

**Remarks**

This property is set automatically by the Microsoft OLAP Services Add-ins Manager library. The program ID is set in the registry when the add-in is registered, and it consists of the program name of the add-in and the name of the class used to instantiate the add-in. For example, the Report Add-In example uses the following program ID:

ReportAddIn.ReportClass

**Example**

The following example prints the *OwnerAddInProgID* property in a message box:

```vba
Private Function IOlapAddIn_ExecuteMenuItem( _
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItem As DSSAddInsManager.OlapMenuItem) _
As DSSAddInsManager.RefreshTreeTypes
On Error Resume Next 'Handle errors
If MenuItem.Caption = "&Special" Then
    MsgBox "OwnerAddInProgID is: " & MenuItem.OwnerAddInProgI
```
End If
End Function

See Also

OlapMenuItem
ParentKey (OlapMenuItem)

The **ParentKey** property of an **OlapMenuItem** object contains the **Key** property of the parent of the menu item.

**Data Type**

Long

**Access**

Read/write

**Remarks**

The **ParentKey** property is used to associate a child menu item with a parent (or owner) pop-up menu item.

**Example**

The following example builds a menu item with child menu items underneath it:

Private Enum PopupItems
   mnuPopSpecial
   mnuPopAverage
   mnuPopPoor
End Enum

Private Enum MenuActions
   mnuAddNew
   mnuRefresh
   mnuDelete
End Enum
Private Sub IOlapAddIn_ProvideMenuItems( _
    CurrentNode As DSSAddInsManager.OlapTreeNode, _
    MenuItems As DSSAddInsManager.OlapMenuItems)
On Error GoTo ProvideMenuItems_Err 'Handle errors
'MenuItems is a collection for MenuItem objects
'Some more code
"Special" is displayed on the root menu as a pop-up
"New", "Refresh", and "Delete" are items on Special's child menu
'flagged by the mnuPopSpecial enumeration
MenuItems.Add mnuStandard, "&Special", mnuPopSpecial, , _
    mnuflagRegular + mnuflagPopup
MenuItems.Add mnuStandard, _
    "&New", mnuAddNew, mnuPopSpecial, mnuflagSubmenu
MenuItems.Add mnuStandard, _
    "&Refresh", mnuRefresh, mnuPopSpecial, mnuflagSubmenu
MenuItems.Add mnuStandard, _
    "&Delete", mnuDelete, mnuPopSpecial, mnuflagSubmenu
Exit Sub
ProvideMenuItems_Err:
    MsgBox "ProvideMenuItems Failed"
    Err.Clear
End Sub

See Also

OlapMenuItem
OlapTreeNode

The **OlapTreeNode** object provides the properties associated with a tree node in Analysis Manager. This object provides no collections or methods.

**Note**  Events for the **OlapTreeNode** object exposed in the Microsoft® Visual Basic® Object Browser are not appropriate for use by add-ins.

**See Also**

Properties, OlapTreeNode
## Properties, OlapTreeNode

The `OlapTreeNode` object contains the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Caption</strong></td>
<td>The <code>OlapTreeNode</code> caption.</td>
</tr>
<tr>
<td><strong>HelpContextId</strong></td>
<td>The Help context ID (optional).</td>
</tr>
<tr>
<td><strong>IconClosed</strong></td>
<td>The icon to display when the tree node is collapsed.</td>
</tr>
<tr>
<td><strong>IconOpen</strong></td>
<td>The icon to display when the tree node is expanded.</td>
</tr>
<tr>
<td><strong>LinkedObject</strong></td>
<td>The object linked to the <code>OlapTreeNode</code> object.</td>
</tr>
<tr>
<td><strong>OwnerAddInName</strong></td>
<td>The name of the add-in associated with the tree node. It is set automatically by the Microsoft OLAP Services Add-ins Manager library.</td>
</tr>
<tr>
<td><strong>OwnerAddInProgId</strong></td>
<td>The program ID of the add-in associated with the tree node. It is set automatically by the Microsoft OLAP Services Add-ins Manager library.</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>The parent <code>OlapTreeNode</code> object of the current node.</td>
</tr>
</tbody>
</table>

### See Also

- [OlapTreeNode](#)
Caption (OlapTreeNode)

The Caption property of an OlapTreeNode object contains the value displayed in the tree node.

Data Type
String

Access
Read/write

See Also
OlapTreeNode
HelpContextId (OlapTreeNode)

The `HelpContextId` property of an `OlapTreeNode` object contains the Help context ID number associated with the item.

Data Type
Long

Access
Read/write

See Also
`OlapTreeNode`
IconClosed (OlapTreeNode)

The `IconClosed` property of an `OlapTreeNode` object specifies the index of the icon to display when the node is collapsed.

**Data Type**

Long

**Access**

Read/write

**Remarks**

The `ProvideIcon` method of the `IOlapAddIn` interface is called when this tree node displays the icon used when the node is collapsed. The value of the `IconClosed` property of the `OlapTreeNode` is used in the `ProvideIcon` method to indicate which icon is to be referenced.

**See Also**

- [OlapTreeNode](#)
- [ProvideIcon (IOlapAddIn Interface)](#)
IconOpen (OlapTreeNode)

The IconOpen property of an OlapTreeNode object specifies the icon to display when the node is expanded.

Data Type
Long

Access
Read/write

Remarks
The ProvideIcon method of the IOlapAddIn interface is called when this tree node displays the icon used when the node is expanded. The value of the IconOpen property of the OlapTreeNode is used in the ProvideIcon method to indicate which icon is to be referenced.

See Also
OlapTreeNode
ProvideIcon (IOlapAddIn Interface)
**LinkedObject (OlapTreeNode)**

The **LinkedObject** property of an **OlapTreeNode** object contains a reference to the object linked to the node. The linked object is supplied by the add-in through the **GetObject** method of the **IOlapAddIn** interface, and it is used to store object references associated with a tree node in Analysis Manager.

**Data Type**

Object

**Access**

Read-only

**Remarks**

This property is useful when combined with another object-based library, such as Decision Support Objects (DSO). Combining the two allows this property to store object references that are relevant to the add-in for a given tree node in Analysis Manager.

In addition, tree nodes owned by Analysis Manager have DSO references to the appropriate object represented by the tree node. For example, a tree node in Analysis Manager for an Analysis server has a DSO **Server** object associated with it.

The **GetObject** method of the **IOlapAddIn** interface is used to provide object references for other tree nodes in Analysis Manager.

**Example**

The following example checks the **LinkedObject** property of the **OlapTreeNode** supplied by **LinkedNode** in the **GetObject** method of the **IOlapAddIn** interface:

Private Function IOlapAddIn_GetObject(LinkedNode As _
DSSAddInsManager.OlapTreeNode) As Object

On Error Resume Next

If Not (LinkedNode.LinkedObject Is Nothing) Then
    ' It already has a linked object
    Beep
End If

End Function

See Also

OlapTreeNode
GetObject (IOlapAddIn Interface)
OwnerAddInName (OlapTreeNode)

The **OwnerAddInName** property of an **OlapTreeNode** object contains the name of the add-in that owns the object.

**Data Type**

String

**Access**

Read/write

**Remarks**

The add-in name for the current **OlapTreeNode** is set automatically by the Microsoft OLAP Services Add-ins Manager library.

**See Also**

[OlapTreeNode](#)
OwnerAddInProgID (OlapTreeNode)

The OwnerAddInProgID property of an OlapTreeNode object contains the program ID of the add-in that owns the tree node.

Data Type

String

Access

Read/write

Remarks

This property is set automatically by the Microsoft OLAP Services Add-ins Manager library. The program ID is set in the registry when the add-in is registered, and it consists of the program name of the add-in and the name of the class used to instantiate the add-in. For example, the Report Add-In example uses the following program ID:

ReportAddIn.ReportClass

Example

The following example prints the OwnerAddInProgID property in a message box:

Private Function IOlapAddIn_GetObject(LinkedNode As _
DSSAddInsManager.OlapTreeNode) As Object

On Error Resume Next

If LinkedNode.Caption = "Samples Forms" Then
    MsgBox "OwnerAddInProgID is: " & LinkedNode.OwnerAddInF
End If

End Function

See Also

OlapTreeNode
Analysis Services Programming

**Parent (OlapTreeNode)**

The **Parent** property of an **OlapTreeNode** object contains a reference to the parent tree node of the selected tree node.

**Data Type**

**OlapTreeNode**

**Access**

Read-only

**See Also**

**OlapTreeNode**
Analysis Services Programming
Enumerations

The Microsoft OLAP Services Add-ins Manager library includes the following enumerations.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errDSSAddinErrorNumbers</code></td>
<td>Enumerates errors that can be raised by the Microsoft OLAP Services Add-ins Manager library</td>
</tr>
<tr>
<td><code>OlapMenuFlags</code></td>
<td>Enumerates flags that set characteristics of a menu item</td>
</tr>
<tr>
<td><code>OlapMenuTypes</code></td>
<td>Enumerates types of menu items</td>
</tr>
<tr>
<td><code>RefreshTreeTypes</code></td>
<td>Enumerates values for the modes of refreshing the Analysis Manager tree pane</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**errDSSAddinErrorNumbers**

The **errDSSAddinErrorNumbers** enumeration provides the following constants.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errCaptionRequired</td>
<td>An error with this error code is raised if you attempt to use the <strong>Add</strong> method of the <strong>OlapTreeNodes</strong> or <strong>OlapMenuItems</strong> collections with a zero-length caption.</td>
</tr>
<tr>
<td>errInvalidMenuType</td>
<td>An error with this error code is raised if you attempt to use the <strong>Add</strong> method of the <strong>OlapMenuItems</strong> collection with a <strong>MenuType</strong> value other than <strong>mnuStandard</strong> or <strong>mnuSeparator</strong>.</td>
</tr>
</tbody>
</table>
OlapMenuFlags

The **OlapMenuFlags** enumeration provides constants for the modes of menu items in the Analysis Manager tree pane.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mnuflagChecked</td>
<td>Indicates that a check mark is to be displayed next to a menu item.</td>
</tr>
<tr>
<td>mnuflagDeleteKey</td>
<td>Enables the <strong>Delete</strong> menu item. It also enables the <strong>Delete</strong> button on the toolbar and the DELETE key on the keyboard.</td>
</tr>
<tr>
<td>mnuflagDisabled</td>
<td>Disables a menu item. However, it will appear to be enabled. To fully disable a menu item, use the <strong>mnuflagGreyed</strong> flag instead.</td>
</tr>
<tr>
<td>mnuflagDoubleClick</td>
<td>Reserved.</td>
</tr>
<tr>
<td>mnuflagF1</td>
<td>Reserved.</td>
</tr>
<tr>
<td>mnuflagGrayed</td>
<td>Disables a menu item.</td>
</tr>
<tr>
<td>mnuflagInsertKey</td>
<td>Reserved.</td>
</tr>
<tr>
<td>mnuflagNew</td>
<td>Enables the <strong>New</strong> menu item as a pop-up menu. Menu items added with this flag appear as child menu items when <strong>New</strong> is selected from the <strong>OlapTreeNode</strong> parent menu.</td>
</tr>
<tr>
<td>mnuflagPopup</td>
<td>Indicates that a newly defined menu item is a pop-up menu. Child menu items are displayed when the menu item is selected.</td>
</tr>
<tr>
<td>mnuflagRegular</td>
<td>Places an item on the root menu of an <strong>OlapTreeNode</strong>.</td>
</tr>
<tr>
<td>mnuflagSeparator</td>
<td>Indicates that the menu item is a separator bar.</td>
</tr>
<tr>
<td>mnuflagSubmenu</td>
<td>Indicates that the menu item belongs to a child menu of the parent pop-up menu item.</td>
</tr>
<tr>
<td>mnuflagTask</td>
<td>Enables the <strong>Task</strong> menu item as a pop-up menu. Menu items added with this flag appear as child menu items when <strong>Task</strong> is selected from the</td>
</tr>
</tbody>
</table>
OlapTreeNode parent menu.

Remarks

These flags are commonly combined by adding them together using the addition operator or the logical OR operator.

Example

The following code identifies a menu item as a child menu item and then disables it, leaving its appearance unchanged:

mnuflagSubmenu OR mnuflagDisabled
OlapMenuTypes

The **OlapMenuTypes** enumeration provides constants for the types of menu items in the Analysis Manager tree pane.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mnuSeparator</td>
<td>Indicates that the menu item is a separator bar</td>
</tr>
<tr>
<td>mnuStandard</td>
<td>Indicates that the menu item is a standard menu item</td>
</tr>
</tbody>
</table>
RefreshTreeTypes

The RefreshTreeTypes enumeration provides constants for refreshing the Analysis Manager tree pane by means of the pane's Refresh method or an event that invalidates the pane's contents.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refrtreeNoRefresh</td>
<td>Indicates that the tree pane will not be refreshed</td>
</tr>
<tr>
<td>refrtreeParentAndBelow</td>
<td>Indicates that the parent of the current node and all of the children of the parent node will be refreshed</td>
</tr>
<tr>
<td>refrtreeCurrentAndBelow</td>
<td>Indicates that the current node and all of its child nodes will be refreshed</td>
</tr>
<tr>
<td>refrtreeAllTree</td>
<td>Indicates that the entire tree pane is refreshed</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Collections

Two collections, **OlapMenuItems** and **OlapTreeNodes**, are used by several methods of the **IOlapAddIn** interface to provide collections of menu items and tree nodes, respectively, to Analysis Manager.

The following topics detail the properties and methods supplied by these collections.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OlapMenuItems</strong></td>
<td>Describes the properties and methods associated with the <strong>OlapMenuItems</strong> collection</td>
</tr>
<tr>
<td><strong>OlapTreeNodes</strong></td>
<td>Details the properties and methods associated with the <strong>OlapTreeNodes</strong> collection</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**OlapMenuItems**

The *OlapMenuItems* collection contains *OlapMenuItem* objects.

This collection contains methods and properties. There are no collections associated with this object.

**See Also**

[Methods, OlapMenuItems](#)

[OlapMenuItem](#)

[Properties, OlapMenuItems](#)
Analysis Services Programming

Methods, OlapMenuItems

The **OlapMenuItems** collection contains the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds an item to the collection</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes an item from the collection</td>
</tr>
</tbody>
</table>
Add (OlapMenuItems)

The **Add** method of the *OlapMenuItems* collection adds a new *OlapMenuItem* object to the collection. It returns an object of type *OlapMenuItem*.

**Syntax**

```vba
Set vnt = object.Add(MenuType As OlapMenuTypes, [Caption As String], [Key As Long], [ParentKey As Long], [Flags As OlapMenuFlags])
```

* vnt
  An instance of *OlapMenuItem* that receives the instance of the new member.
* object
  An instance of the *OlapMenuItems* collection.
* MenuType
  A constant from the *OlapMenuTypes* enumeration.
* Caption
  The string value to be displayed in the menu.
* Key
  User-defined key value to be used by the add-in.
* ParentKey
  Associates a child menu item with its parent. Used when the parent menu item is defined using the *mnuflagPopup* option in *MenuType*.
* Flags
  A bitmask of values from the *OlapMenuFlags* enumeration.

**Remarks**

The **Add** method is used to populate the *OlapMenuItems* collection with menu
items to be displayed when the user right-clicks a tree node. Call this method for each menu item you want to add.

**Example**

The following example builds a menu with various menu items:

```vba
Private Enum MenuActions
    mnuActTop
    mnuActMid
    mnuActBtm
    mnuActSpc
End Enum

Private Sub IOlapAddIn_ProvideMenuItems(CurrentNode As DSSAddInsManager.OlapTreeNode, MenuItems As DSSAddInsManager.OlapMenuItems)
    On Error GoTo ProvideMenuItems_Err 'Handle errors
    If CurrentNode.Caption = "Node 1" Then
        'Enable default New menu item and add child menu items
        MenuItems.Add mnuStandard, "&Top", mnuActTop, , mnuflagNew
        MenuItems.Add mnuStandard, "&Mid", mnuActMid, , mnuflagNew
        MenuItems.Add mnuStandard, "&Btm", mnuActBtm, , mnuflagNew
        'Add regular menu item to root menu
        MenuItems.Add mnuStandard, "&Special", mnuActSpc, , mnuflagRegular
    End If
    Exit Sub

ProvideMenuItems_Err:
    MsgBox "ProvideIcon failed"
    Err.Clear
End Sub
```

**See Also**

OlapMenuItems
Remove (OlapMenuItems)

The **Remove** method of the **OlapMenuItems** collection removes an **OlapMenuItem** object from the collection.

**Syntax**

```
object.Remove(Index)
```

**object**

An instance of the **OlapMenuItems** collection.

**Index**

Specifies the index of the object to be removed from the collection. If it is an integer, **Index** specifies the ordinal position of the item in the collection. If it is a string, **Index** specifies the key assigned to the item when it was added to the collection.

**See Also**

- [OlapMenuItem](#)
- [OlapMenuItems](#)
Properties, OlapMenuItems

The `OlapMenuItems` collection contains the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>The number of <code>OlapMenuItem</code> objects in the <code>OlapMenuItems</code> collection</td>
</tr>
<tr>
<td>Item</td>
<td>The <code>OlapMenuItem</code> object to be returned from the <code>OlapMenuItems</code> collection</td>
</tr>
</tbody>
</table>

See Also

- `OlapMenuItem`
- `OlapMenuItems`
Count (OlapMenuItems)

The `Count` property of an `OlapMenuItems` collection returns the number of items in the collection.

**Syntax**

`object.Count`

`object`

An instance of the `OlapMenuItems` collection.

**Data Type**

Long

**Remarks**

Use this property to iterate through a collection of menu items.

**See Also**

`OlapMenuItems`
Item (OlapMenuItems)

The Item property of an OlapMenuItems collection retrieves a specified item from the collection.

Syntax

object.Item(Index)

object

An instance of the OlapMenuItems collection.

Index

Specifies the index of the object to be retrieved from the collection. If it is an integer, Index specifies the ordinal position of the item in the collection. If it is a string, Index specifies the key assigned to the item when it was added to the collection.

Data Type

OlapMenuItem

Remarks

Use this method to retrieve a specific OlapMenuItem reference from the collection.

See Also

OlapMenuItems
Analysis Services Programming

**OlapTreeNodes**

The `OlapTreeNodes` collection contains `OlapTreeNode` objects. This collection contains methods and properties.

**See Also**

- [Methods, OlapTreeNodes](#)
- [OlapTreeNode](#)
- [Properties, OlapTreeNodes](#)
Analysis Services Programming

Methods, OlapTreeNode

The `OlapTreeNode` collection contains the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds an item to the collection</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes an item from the collection</td>
</tr>
</tbody>
</table>

See Also

`OlapTreeNode` Collection
Add (OlapTreeNodes)

The Add method of the OlapTreeNodes collection adds a new OlapTreeNode object to the collection. This method returns an object of type OlapTreeNode.

Syntax

Set vnt = object.Add(Caption As String, IconClosed As Integer, [IconOpen As Integer])

vnt

An instance of OlapTreeNode that receives the instance of the new member.

object

An instance of the OlapTreeNodes collection.

Caption

The string value to be displayed for the tree node.

IconClosed

The index of the icon resource displayed when the tree node is collapsed.

IconOpen

(Optional) The index of the icon resource displayed when the tree node is expanded.

Remarks

The Add method is used to populate the OlapTreeNodes collection with child nodes that are displayed in the tree. Call this method for each OlapTreeNode object you want to add.

Example

Private Enum NodeIcons
icoNode1 = 1
icoNode2
icoNode3
End Enum

Private Sub IOlapAddIn_ProvideChildNodes(_
    ParentNode As DSSAddInsManager.OlapTreeNode, _
    OlapTreeNodes As DSSAddInsManager.OlapTreeNodes)
On Error GoTo ProvideChildNodes_Err 'Handle errors

If ParentNode.Caption = "Analysis Servers" Then
    OlapTreeNodes.Add "Node 1", icoNode1
    OlapTreeNodes.Add "Node 2", icoNode2
    OlapTreeNodes.Add "Node 3", icoNode3
End If

Exit Sub
ProvideChildNodes_Err:
    Debug.Assert False
    MsgBox "ProvideChildNodes Failed"
    Err.Clear
End Sub

See Also

OlapTreeNodes Collection
Remove (OlapTreeNodes)

The Remove method of the OlapTreeNodes collection removes an existing OlapTreeNode object from the collection.

Syntax

object.Remove(Index)

object

An instance of the OlapTreeNodes collection.

Index

Specifies the index of the object to be removed from the collection. If it is an integer, Index specifies the ordinal position of the item in the collection. If it is a string, Index specifies the key assigned to the item when it was added to the collection.

See Also

OlapTreeNode

OlapTreeNodes Collection
Properties, OlapTreeNodes

The **OlapTreeNodes** collection provides the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Count</strong></td>
<td>The number of <strong>OlapTreeNode</strong> objects in the <strong>OlapTreeNodes</strong> collection</td>
</tr>
<tr>
<td><strong>Item</strong></td>
<td>A specified <strong>OlapTreeNode</strong> object from the <strong>OlapTreeNodes</strong> collection</td>
</tr>
</tbody>
</table>

See Also

- [OlapTreeNode](#)
- [OlapTreeNodes Collection](#)
Count (OlapTreeNodes)

The Count property of an OlapTreeNodes collection returns the number of objects in the collection.

Syntax

object.Count

object

An instance of the OlapTreeNodes collection.

Data Type

Long

Remarks

Use this property to iterate through the OlapTreeNodes collection.

See Also

OlapTreeNodes Collection
Item (OlapTreeNodes)

The Item property of an `OlapTreeNodes` collection retrieves a specified item from the collection.

Syntax

```
object.Item(Index)
```

- `object`  
  An instance of the `OlapTreeNodes` collection.

- `Index`  
  Specifies the index of the object to be retrieved from the collection. If it is an integer, `Index` specifies the ordinal position of the item in the collection. If it is a string, `Index` specifies the key assigned to the item when it was added to the collection.

Data Type

`OlapTreeNode`

Remarks

Specifies the `OlapTreeNode` object when retrieving objects from this collection. If it is an integer, `Index` specifies the ordinal position of the item in the collection. If it is a string, `Index` specifies the key assigned to the item when it was added to the collection.

See Also

`OlapTreeNodes`
Analysis Services Programming
PivotTable Service

PivotTable® Service is the primary interface for applications interacting with Microsoft® SQL Server™ 2000 Analysis Services. It is used to build client applications that interact with multidimensional data. PivotTable Service also provides methods for online and offline data mining analysis of multidimensional data and relational data. PivotTable Service is included as part of Analysis Services, and it can be redistributed by third-party client applications.

PivotTable Service is the primary method for interacting with Analysis Services in order to accomplish such tasks as connecting to a cube or data mining model, querying a cube or data mining model, and retrieving schema information.

As a stand-alone provider, PivotTable Service provides client applications with the ability to create local cube files and mining models from relational and multidimensional sources. Client applications can connect to a local cube and execute queries using Multidimensional Expressions (MDX) without interacting with the full-scale Analysis server.

PivotTable Service can be used in a variety of development environments. Both Microsoft Visual Basic® and Visual C++® developers can use either the Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD) object library or the OLE DB for OLAP Component Object Model (COM) interfaces to create client applications.

Intended Audience

This document is intended for developers who are interested in developing client applications that work with Analysis Services. Readers should be familiar with online analytical processing (OLAP) and the structure of multidimensional data (cubes), and MDX. Knowledge of data mining theory and practice is also helpful. Additionally, knowledge of either C++ or a COM Automation language, such as Microsoft Visual Basic, is required.

COM Automation programmers should have some knowledge of programming using ActiveX components and ADO. Programmers using C++ should be well versed in OLE DB and COM. All programmers should be able to work with
SQL, including the data definition language (DDL) and data manipulation language (DML) extensions defined by Transact-SQL.

Readers developing data mining applications should be familiar with the OLE DB for Data Mining specification, which includes detailed information about standard mining models, OLE DB for Data Mining grammar, query syntax, schema rowsets, prediction functions, special histogram nested table columns, and mining model XML format (PMML). The OLE DB for Data Mining specification is available for download from the Microsoft OLE DB Web page at the Microsoft Web site.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview of PivotTable Service</strong></td>
<td>Overview of PivotTable Service capabilities and uses</td>
</tr>
<tr>
<td><strong>Key Concepts in PivotTable Service</strong></td>
<td>Information about installing, setting up, and distributing PivotTable Service with custom client applications</td>
</tr>
<tr>
<td><strong>Client Operations in PivotTable Service</strong></td>
<td>Guide to common operations involving PivotTable Service</td>
</tr>
<tr>
<td><strong>PivotTable Service Programmer's Reference</strong></td>
<td>Reference material for PivotTable Service, including properties and DDL</td>
</tr>
</tbody>
</table>

For more information about using ADO, see the ADO documentation.

For more information about using OLE DB in Microsoft Visual C++, see the OLE DB documentation.

For more information about Transact-SQL, see Transact-SQL Overview.

For more information about MDX, see MDX.
Analysis Services Programming
Overview of PivotTable Service

PivotTable® Service is an OLE DB provider for multidimensional data and data mining operations. This means that it provides OLE DB functionality for applications that need access to multidimensional data and data mining services. By providing support for a subset of SQL and Multidimensional Expressions (MDX), PivotTable Service enables applications to retrieve tabular and multidimensional data. The data can be displayed, included in a local cube, analyzed using sophisticated data mining algorithms, or updated.

The execution speed of these complex operations makes it possible to perform sophisticated analyses on the client computer itself. This, in turn, allows remote client applications to function independent of a high-speed network, intranet connection, or physical presence at a geographical location. For example, a sales representative can forecast profit margins at a potential customer's site based on proposals by that customer even if her computer is not connected to her own company's network. She can also repopulate her local cube with new data whenever she needs to do so, by using the Internet to connect to her corporate Analysis server and refreshing her local data with any new or updated data.

PivotTable Service also supports data definition language (DDL) in the connection string of the client application so that offline clients can create and modify local cubes at run time and define temporary multidimensional objects for use in analysis. This ability allows client applications to be extensible in the face of changing business practices and across business boundaries. The data for local cubes can be derived from either a multidimensional data source or a relational one such as Microsoft® SQL Server™, Microsoft Access, or Oracle.

For more information about using PivotTable Service in various development environments, see Development Environments.

You can develop client applications that use PivotTable Service using a variety of techniques and environments. You can use Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD) to implement client applications in any Component Object Model (COM) Automation language, such as Microsoft Visual Basic®, or as Active Server Pages (ASP) on a Web site. C++ programmers can use PivotTable Service with COM and OLE DB to implement
highly specialized custom applications.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>What's New in PivotTable Service</td>
<td>Describes new features for this release</td>
</tr>
<tr>
<td>Redistributing Components</td>
<td>Contains information about redistributing PivotTable components with third party applications</td>
</tr>
<tr>
<td>Developing Client Applications</td>
<td>Describes building client applications using PivotTable Service</td>
</tr>
<tr>
<td>Development Environments</td>
<td>Contains information about using PivotTable Service in different development environments</td>
</tr>
</tbody>
</table>
Analysis Services Programming
What's New in PivotTable Service

PivotTable® Service supports the advanced data mining and analysis techniques that are introduced in Microsoft® SQL Server™ 2000.

This release of PivotTable Service also includes new features that enhance communication with the Analysis server and the management of local cubes.

Client applications can communicate with the Analysis server through Microsoft Internet Information Services (IIS) using HTTP; clients do not need physical proximity to the Analysis server to take advantage of common Internet services such as Domain Name System (DNS). The Analysis server also supports security features such as cube roles, member security, and cell security.

The ability to create and manage local cubes from the client application has been improved. Client applications can now alter the structure of a cube, define default members, and sort by member properties by using the ALTER CUBE statement. The UPDATE CUBE command supports writeback to members higher than those at the leaf level of a hierarchy.

Advanced Data Mining and Analysis

PivotTable Service supports data mining and analysis with the addition of the CREATE MINING MODEL statement and extensions to the CREATE VIRTUAL CUBE statement. Two algorithms are included: Microsoft Decision Trees and Microsoft Clustering.

Data mining models can be created on the server, using Decision Support Objects (DSO), or locally, using the CREATE MINING MODEL statement. The models can then be trained and used to produce predictions based on trends identified in the training data.

The syntax for virtual cubes has also been extended to allow the inclusion of mining models.

For more information, see Data Mining Models, Advanced Data Mining and Analysis, CREATE MINING MODEL Statement, and Decision Support Objects.

Security Enhancements in PivotTable Service
SQL Server 2000 Analysis Services provides three security models with which PivotTable Service interacts.

**Cube security**

Read and read/write permissions for databases and cubes in a database.

**Member security**

Secured members are visible only to users who have been granted permission to read them. For example, a user may query a States level that contains the members Washington, Oregon, and California. If that user has at least read permission on the States level and all three of its members, all three states are returned. If the user does not have at least read permission on California, California is not returned in the query. No error is raised, and no placeholder is generated.

**Cell security**

Queries that involve a secured cell or set of cells return an error code or value, depending on the value of the Secured Cell Value property of the connection string.

For more information, see Security in PivotTable Service, Data Source Property, Secured Cell Value Property, and SSPI Property.

**Connection Enhancements in PivotTable Service**

The following enhancements have been made to the way that PivotTable Service connects to a data source.

**Connecting to a server over the Internet**

Client applications can connect to the Analysis server through IIS by using a URL for the server name in the Data Source property of the connection string. For more information, see Connecting Using HTTP.

**Using the security services provider interface**

You can use the SSPI property to specify a third-party security provider when connecting to a data source using PivotTable Service.

**Connecting using a different user role**
When connecting to a database, the client developer can specify a different role than the current one using the **Roles** property. This allows developers and database administrators (DBAs) to test security permission settings for different roles without having to log in again under those roles. The developer or DBA must be a member of the role under which he or she is connecting.

**Data Source Property**, **Connecting Using HTTP**, **SSPI Property**, and **Roles Property**.

**Cube Enhancements**

PivotTable Service includes the following enhancements to the client application's interaction with server cubes and local cubes.

**Ragged hierarchies**

PivotTable Service supports a variety of hierarchy structures in dimensions. Some hierarchies that are based on levels, such as geographical dimensions, may have branches where there is no member for a particular level. A geographical hierarchy that includes Country, State_province, and City levels might include a country that does not have states or provinces. Such hierarchies are known as ragged hierarchies. For example, a geography dimension might include Washington, D.C. Unlike other cities in the United States, this city is not a child member of any state. Its parent is the [USA] member itself.

**Unbalanced hierarchies**

Some hierarchies are based less on the concept of levels and more on the hierarchical relationship between the dimension members. Organization charts and part manufacturing structures are examples of such hierarchies. These hierarchies, known as unbalanced hierarchies, are created in Analysis Services from a table that has a column of keys that refer to primary keys in the same table. In an organization chart table this column is often called **ManagedBy** or **Manager**. The concept of levels is often less significant than the relationship hierarchy in these dimensions. For example, one executive assistant may report directly to a vice president, whereas another may report to a manager.
Custom rollups and calculated members

You can define custom rollup functions in addition to the standard aggregate functions (Distinct Count, Sum, Min, Max, and Count). For example, you can use a rollup function to define a weekly average. For more information, see CREATE MEMBER Statement.

PivotTable Service supports calculated members for server and local cubes. For more information, see Defining Calculated Members.

Allocations

Allocations are now supported by the inclusion of the UPDATE CUBE statement in data manipulation language (DML). Aggregate members (that is, members that are higher than the leaf level) can have a new value assigned to them. The component members of the aggregate member are updated according to the allocations described by the parameters of the UPDATE CUBE statement.

For more information, see Writing a Value Back to a Cell and UPDATE CUBE Statement.

Altering the structure of a cube

PivotTable Service now supports the ALTER CUBE statement of the data definition language (DDL). You can modify existing cubes by:

- Adding a dimension member.
- Removing a dimension member.
- Moving a dimension member.
- Modifying the properties of a dimension member.

For more information, see ALTER CUBE Statement.

Defining default hierarchies and members
You can define members and hierarchies as the default member or hierarchy for a dimension using the ALTER CUBE statement. For example, you can define the member USA as the default member of the Geography dimension (which only has one hierarchy) in a cube that contains data for the USA only. Alternatively, you can define the default hierarchy of the Time dimension as the Year-Month-Day hierarchy or the Year-Quarter hierarchy. For more information, see CREATE CUBE Statement.

New schema rowsets

The following schema rowsets are now supported.

<table>
<thead>
<tr>
<th>Schema rowset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDSHEMA_FUNCTIONS</td>
<td>Describes the functions that are available to client applications connected to the database</td>
</tr>
<tr>
<td>MDSHEMA_PROPERTIES</td>
<td>Describes the properties of members contained within a database</td>
</tr>
<tr>
<td>MDSHEMA_SETS</td>
<td>Describes any sets that are currently defined</td>
</tr>
<tr>
<td>MDSHEMA_ACTIONS</td>
<td>Describes the actions that may be available to client application</td>
</tr>
<tr>
<td>MDSHEMA_CELL_FORMULAS</td>
<td>Describes the calculated cells that may be contained within a database</td>
</tr>
<tr>
<td>MINING_MODELS</td>
<td>Exposes data mining models</td>
</tr>
<tr>
<td>MINING_COLUMNS</td>
<td>Describes the individual columns of all defined data mining models known to the provider</td>
</tr>
<tr>
<td>MINING_MODEL_CONTENT</td>
<td>Allows browsing of the content of a data mining model</td>
</tr>
<tr>
<td>MINING_SERVICES</td>
<td>Provides a description of each data mining algorithm that is supported by that provider</td>
</tr>
<tr>
<td>MINING_SERVICE_PARAMETERS</td>
<td>Provides a list of parameters that can be supplied when generating a mining model using the CREATE MINING MODEL statement</td>
</tr>
</tbody>
</table>
MINING MODEL CONTENT PMML | Stores the Predictive Model Markup Language (PMML) standard Extensible Markup Language (XML) representation of the mining model

For more information, see Schema Rowsets and the OLE DB documentation.

**Sorting by member properties**

When building a local cube file, you can sort members in a dimension using member properties. For example, assume that a Geography dimension has a member property Latitude of type **string**. When you build a cube that contains the Geography dimension, you can sort the members of the geography dimension by the member property Latitude instead of by the Name or Key properties. For more information, see CREATE CUBE Statement.

**Actions**

Each member of a cube can have an action associated with it. Actions are context-sensitive operations that allow a client application to trigger a custom, data-sensitive operation on the members. For example, you can define an action for members of the Customers dimension that retrieves a rowset describing the details for that customer. The client application may allow the user to right-click the customer entry to trigger the display of that rowset. For more information, see Creating Actions and CREATE ACTION Statement.

**Drilling through to source data**

The source data that make up a cell's value can be obtained by using the DRILLTHROUGH statement. For example, a user who is interested in the cell {"Sales","Pearl Light Beer","Orgeon","1998"} can obtain the rows from the fact table that were used to compute that cell's value. For more information, see DRILLTHROUGH Statement.
Analysis Services Programming
Redistributing Components

PivotTable® Service includes a number of dynamic-link libraries (DLLs) that you may need to ship with a client application. Any individual client application may need a combination of these components, depending on the PivotTable Service features it uses.

<table>
<thead>
<tr>
<th>File set</th>
<th>Component files</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Msolap80.dll, Msolui80.dll, Msolap80.rll, Olapuir.rll, and Microsoft® Data Access Components (MDAC)</td>
</tr>
<tr>
<td>2</td>
<td>File Set 1 plus Msmdcb80.dll, Msmdgd80.dll, and an appropriate OLE DB tabular data provider</td>
</tr>
<tr>
<td>3</td>
<td>File Set 1 plus Msdmine.dll, Msmdun80.dll, Msdmine.rll, and Msdmeng.dll</td>
</tr>
</tbody>
</table>

The following table shows which file set to use based on which tasks you want your client application to perform.

<table>
<thead>
<tr>
<th>Task</th>
<th>File set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicate with the Analysis server using TCP/IP or HTTP and read local cube files</td>
<td>1</td>
</tr>
<tr>
<td>Create and refresh local cubes</td>
<td>2</td>
</tr>
<tr>
<td>Read OLAP and relational data mining models</td>
<td>3</td>
</tr>
</tbody>
</table>

You must install MDAC before you install PivotTable Service. File sets 2 and 3 can be combined if the entire suite of components is desired. If you install PivotTable Service with Microsoft Windows® 95, you must install distributed COM (DCOM) before you install MDAC. For more information about distributing and installing MDAC, see the MSDN® Library at the Microsoft Web site.

**Installing and Registering Components**

You can create your own setup program to install and register the redistribution component files and prerequisite Microsoft® Data Access Components (MDAC)
files. Alternatively, you can use one of the setup programs provided on the SQL Server™ 2000 CD-ROM. These setup programs are described in Redistribution Setup Programs later in this topic.

The PivotTable Service files, with the exception of the resource files (files with an extension of .rll), must be installed to the following location:

C:\Program Files\Common Files\System\OLE DB

After the required components have been installed, the following components must also be registered using Regsvr32.exe or their own DLLSelfRegister functions:

- Msolap80.dll
- Msolui80.dll
- Msmdgd80.dll
- Msmdcb80.dll
- Msmdun80.dll
- Msdmine.dll
- Msdmeng.dll

The English Language versions of the resource files, Msolap80.rll, Olapuir.rll and Msdmine.rll, must be installed to the following location:

C:\Program Files\Common Files\System\OLE DB\Resources\1033

If you are shipping a localized product, you must install both the English and the localized versions of these files and place them in their appropriate resource directories. For example, if you are shipping a German version of your product, you must install the German versions of the resource files in the following directory:
When registering DLLs, observe the following dependencies:

- Msolap80.dll depends on Msolap80.rll, Wininet.dll, and Oleaut32.dll version 2.3.0 or later.

- Msolap80.rll is the resource file for Msolap.dll. Oleaut32 is the OLE Automation run-time library, and is also installed in either the C:\Windows\System or C:\Winnt\System32 directories.

- Msolui80.dll depends on Msolap80.dll, Msvbvm60.dll, and Oleaut32.dll version 2.3.0 or later.

Msvbvm60.dll is a Microsoft Visual Basic® run-time library, and Wininet.dll contains the Internet automation run-time library. Both are installed in the C:\Windows\System or C:\Winnt\System32 directory.

Note You can use the Depends.exe utility to see the full set of external dependencies for these DLLs. This utility is available in either the Windows NT® 4.0 Resource Kit or the Windows 2000 Server Resource Kit.

**Installation Registry Settings**

For the purposes of installing and uninstalling, all PivotTable Service files should be considered shared files. Create a registry value for each PivotTable Service file under the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs

If this registry value (known as the reference counter) already exists, it should be incremented by one during installation of the PivotTable Service files. During removal of the PivotTable Service files, the reference counter should be decremented by one. The PivotTable Service files should not be deleted if the corresponding reference counter is greater than zero.

**Redistribution Setup Programs**
Two setup programs, Ptslite.exe and Ptsfull.exe, are provided on the SQL Server 2000 CD-ROM, in the folder \\Msolap\Install\Pts. Ptslite.exe installs the PivotTable Service files only; Ptsfull.exe installs the PivotTable Service files and Microsoft Data Access Components (MDAC).

In addition to two required parameters, both Ptslite.exe and Ptsfull.exe use the same optional command line parameters as the Analysis Services Setup program. The required parameters, which must be the first parameters on the command line, are -s and -a. Optional parameters follow these required parameters. For example, to use Ptslite.exe to perform a silent installation of the PivotTable Service files, the command is:

Ptssite -s -a –s –f1Setup.iss ...

For more information, see Setup Parameters and Silent Installation.

**Ptslite.exe**

Ptslite.exe installs the following PivotTable Service files.

<table>
<thead>
<tr>
<th>atl.dll</th>
<th>msdmeng.dll</th>
<th>msdmine.dll</th>
</tr>
</thead>
<tbody>
<tr>
<td>msmdcb80.dll</td>
<td>mdmdgd80.dll</td>
<td>msolap80.dll</td>
</tr>
<tr>
<td>msolui80.dll</td>
<td>msmdcube.dll</td>
<td>msmdgdrv.dll</td>
</tr>
<tr>
<td>msolap.dll</td>
<td>msolapui.dll</td>
<td>msdmine.rll</td>
</tr>
<tr>
<td>msolap80.rll</td>
<td>olapui.rll</td>
<td>msvbvm60.dll</td>
</tr>
<tr>
<td>msmdun80.dll</td>
<td>msolap.rdll</td>
<td></td>
</tr>
</tbody>
</table>

**Ptsfull.exe**

Ptsfull.exe installs the same files as Ptslite.exe, and also the Microsoft Data Access Components (MDAC).
Analysis Services Programming
Developing Client Applications

PivotTable® Service is the primary method of communication between a client application and a multidimensional data source or data mining model, such as Microsoft® SQL Server™ 2000 Analysis Services. It is used by applications provided by Microsoft (such as the Microsoft Excel PivotTable and PivotChart® features) and by applications provided by third-party vendors. Programmers who want to develop custom client applications for Analysis Services must use PivotTable Service.

PivotTable Service client applications can retrieve, display and manipulate data from multidimensional sources, such as OLAP cubes, or from data mining models. PivotTable Service client applications can also create cube files and data mining models on the local computer, and populate them with data derived from an OLE DB tabular provider such as SQL Server or an OLAP cube. After a cube or mining model is created, client applications can use PivotTable Service to browse and analyze the data contained in it. In the case of OLAP cubes, client applications can conduct what-if analyses using writeback and cell allocations. In the case of local multidimensional OLAP (MOLAP) cubes, such analyses can be conducted even if the client application is not connected to the original data source. In this case, PivotTable Service takes the place of the Analysis server by providing many of the functions of the server on the local computer.

The data source for creating a local cube file or mining model can be any OLE DB data provider, such as SQL Server or Analysis Services.

Overview of Client Applications

Client applications that need to access OLAP data can be implemented either in a Component Object Model (COM) Automation language, such as Microsoft Visual Basic®, or in Microsoft Visual C++®. Client applications that are implemented in a COM Automation language or in C++ may access PivotTable service using Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD), or they can use the COM interfaces provided by OLE DB for OLAP. For more information, see the OLE DB documentation.

Microsoft Office
Microsoft Excel and other Microsoft Office products work with PivotTable Service to provide access to OLAP cubes for creating reports and charts. For more information, see the Excel documentation.
Analysis Services Programming
Development Environments

You can use PivotTable® Service using either Microsoft® ActiveX® Data Objects (ADO) and a COM Automation language, such as Microsoft Visual Basic®, or the COM interfaces provided by OLE DB for OLAP using Microsoft Visual C++®.

Microsoft SQL Server™ 2000 Analysis Services and PivotTable Service are fully compliant with OLE DB for OLAP 2.0. For more information, see the OLE DB documentation.
Using Visual C++

In a COM and Visual C++® environment, you can interact with PivotTable® Service using OLE DB for OLAP. This extension to OLE DB contains interfaces for issuing and retrieving Multidimensional Expressions (MDX) queries. The primary interface for retrieving multidimensional data is **IMDDataset**, which returns cell values and column information. The OLE DB **IDBSchemaRowset** interface is used to retrieve multidimensional schema information. OLE DB for OLAP defines the additional schema rowsets you need to retrieve this information. For more information, see the OLE DB documentation.

In a COM and C++ environment, you can also use the Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD) libraries with late binding.

For more information, see [Visual C++ and ADO](#).
Using Visual Basic

In a COM Automation environment, you can interact with PivotTable® Service using the Microsoft® ActiveX® Data Objects (ADO) library and the Microsoft ActiveX Data Objects (Multidimensional) (ADO MD) library.

These libraries can be used together or independently of each other.

See Also

Using ADO
Using ADO MD
Using Active Server Pages
Analysis Services Programming

Using ADO MD

The Microsoft® ActiveX® Data Objects (Multidimensional) (ADO MD) library contains a number of objects you can use with PivotTable® Service. The **Cellset** object allows developers to issue Multidimensional Expressions (MDX) queries against cubes on the Analysis server or local cube files. The **CubeDef** object enables you to retrieve multidimensional schema information.

To use the ADO MD library independently of the ADO library, use the **ActiveConnection** property of the **Catalog** or **Cellset** objects. For more information, see the ADO MD documentation.

**Example**

The following code creates a **Cellset** object and sets the **ActiveConnection** property to the **FoodMart 2000** sample database. It then executes a query against the **Sales** cube.

```vbscript
Dim MyCellSet As New ADOMD.Cellset
MyCellSet.ActiveConnection = "Provider=msolap; Data Source=Local\FoodMart 2000; Initial Catalog=FoodMart 2000;"
MyCellSet.Source = "select {[Measures].[Unit Sales]} on columns," & 
    "order(except([Promotion Media].[Media Type].members," & 
    "'[Promotion Media].[Media Type].[No Media]}),[Measures].[Unit Sales]"
MyCellSet.Open
```

**See Also**

[Working with OLAP Data](#)

[Using ADO](#)
Using ADO

This library contains objects that may be used to develop client applications that use PivotTable® Service. This library provides objects such as the Connection object, which provides methods such as Open, OpenSchema, and Execute. For more information, see the Microsoft® ActiveX® Data Objects (ADO) documentation.

It is not necessary to use the ActiveX Data Objects (Multidimensional) (ADO MD) library in conjunction with ADO to retrieve data from PivotTable Service. Each library can be used independently of the other. For more information, see Working with OLAP Data.

Example

The following code creates a Command object:

Dim MyCommand As New ADODB.Command
MyCommand.ActiveConnection = "Provider=msolap; Data Source=LocalHost; Initial Catalog=FoodMart 2000;"
MyCommand.CommandText = "select {[Measures].[Unit Sales]} on columns," & 
  "order(except([Promotion Media].[Media Type].members," & 
  "{{[Promotion Media].[Media Type].[No Media]}},[Measures].[Unit Sales])" & 
  "From Sales"
MyCommand.Execute

See Also

Using ADO MD
Analysis Services Programming

**Using Active Server Pages**

PivotTable® Service can be used by Active Server Pages (ASP) Microsoft® Visual Basic® Scripting Edition (VBScript) programmers by using the `CreateObject` method of the ASP Server object. This method can be used to create any needed object from either the Microsoft® ActiveX® Data Objects (ADO) or ActiveX Data Objects (Multidimensional) (ADO MD) object libraries.

**Example**

The following example creates an ADO MD **Cellset** object using VBScript:

```vbscript
Dim MyCellset
Set MyCellset = Server.CreateObject("ADOMD.Cellset")
```

**See Also**

[Using ADO](#)

[Using ADO MD](#)
Analysis Services Programming
Key Concepts in PivotTable Service

The following topics may be helpful in understanding the concepts that are used when developing client applications for Microsoft® SQL Server™ 2000 Analysis Services. Each topic is presented from the point of view of the client application. For more information presented from the point of view of the server, see Analysis Services Architecture.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context of Connections</td>
<td>Describes how the different connection contexts in PivotTable® Service define the features available to the client application</td>
</tr>
<tr>
<td>Organization of Multidimensional Data</td>
<td>Describes how multidimensional data is organized within Analysis Services, and how PivotTable Service uses that organization</td>
</tr>
<tr>
<td>Advanced Data Mining and Analysis</td>
<td>Information about using data mining in client applications</td>
</tr>
<tr>
<td>Working with OLAP Data</td>
<td>Describes how OLAP data can be manipulated and displayed</td>
</tr>
<tr>
<td>Calculated Members</td>
<td>Describes how to create calculated members</td>
</tr>
<tr>
<td>Managing the Client Cache</td>
<td>Information on using the client cache</td>
</tr>
<tr>
<td>Transactions in Analysis Services</td>
<td>Describes how transactions affect features such as writeback and allocations</td>
</tr>
<tr>
<td>Security in PivotTable Service</td>
<td>Describes how PivotTable Service works with the security features in Analysis Services</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Context of Connections

The features and characteristics of PivotTable® Service are defined by the context of the connection that it is managing. There are three connection contexts in PivotTable Service:

- Connected to Microsoft® SQL Server™ 2000 Analysis Services
- Connected to an OLE DB provider
- Connected to a local cube or a local data mining model

When PivotTable Service is connected to an Analysis server, details such as the communication protocol between the client application, server management, and the cache management are hidden from the client application. This simplifies implementation of client applications. It is possible for the client application to interact with any type of cube on the Analysis server, create local cubes based on server cubes (MOLAP cubes), create local cubes based on separate relational databases (that is, ROLAP cubes), create a local data mining model, and interact with existing data mining models.

When connected to a ROLAP cube, PivotTable Service acts as a server for elements that reside in the local cube and as an interface for elements of the cube that reside in the relational database.

When connected to a local MOLAP cube or data mining model, PivotTable Service takes on the role of the Analysis server. PivotTable Service interprets commands from the client application and executes them against the local data source, without need for communication with a separate Analysis server. PivotTable Service formats the results and passes them back to the client application.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected to Analysis Services</td>
<td>Describes how PivotTable Service gets data from Microsoft® SQL Server™ 2000 Analysis Services, builds cube slices based</td>
</tr>
</tbody>
</table>
on cubes on the Analysis server, works with data mining models on the Analysis server, and connects to Analysis Services over the Internet

| **Connected to an OLE DB Provider** | Describes how PivotTable Service retrieves data from a local ROLAP cube and builds local cube files |
| **Connected to a Local Cube File or Data Mining Model** | Describes how PivotTable Service retrieves data from a local cube file or a local mining model |
Connected to Analysis Services

There are a variety of connection contexts to be considered when connected to Microsoft® SQL Server 2000™ Analysis Services. The context of the connection determines how the connection is established and which features and properties are available to the client application. For example, when connecting over the Internet, using HTTP, you should consider whether the **User ID** and **Password** properties will be needed to establish the connection. When connecting to an Analysis server, you may want to optimize cache properties for better performance.

Retrieving Data from Analysis Services

When using PivotTable® Service to retrieve data from Analysis Services, PivotTable Service communicates with a remote Analysis server through a network connection or through shared memory on the local computer. The network protocol is either TCP/IP or HTTP. If a connection request is made that references a remote SQL provider or other tabular data provider, the request is automatically routed to the provider in question. The caching of result sets and other optimizations is not visible to the client application.

Deriving Local Cube Files from Server Cubes (Slicing)

It is possible to build local cubes, which are based upon cubes that reside on a server. This process is similar to creating a local cube based on a fact table, except that the server cube itself is used as the data source. When a WHERE clause is specified as part of the CREATE CUBE statement's WHERE clause, or when only a subset of the available dimensions or measures are specified, the operation is referred to as a slice. For more information, see [Building Local Cubes](#).

**Building, Training, and Retrieving Data from a Data Mining Model**
When connected to Analysis Services, it is possible to create, train (that is, process) and interact with mining models on the local computer.

You can create a mining model on the server using Decision Support Objects (DSO). To create a mining model locally, use data definition language (DDL).

For more information about creating data mining models on the Analysis server, see Data Mining Examples.

**Connecting Using HTTP**

This feature enables the user to connect to the Analysis server through Microsoft Internet Information Services (IIS). By setting the Data Source connection string property to an HTTP or HTTPS URL, PivotTable Service is able to tunnel a connection to the Analysis server through firewalls or proxy servers. This is accomplished by use of a special Active Server Pages (ASP) page, Msolap.asp, which is installed by default to C:\Program Files\Microsoft Analysis Services\Bin.

The rest of the connection string is specified normally.

For more information about the **ConnectionString** property, see the ADO documentation.

**See Also**

Connecting Using HTTP
Analysis Services Programming

**Connected to an OLE DB Provider**

It is possible for PivotTable® Service to connect to a relational OLE DB provider directly:

- When retrieving data from a relational OLAP (ROLAP) local cube file
- When building a local cube file

**Retrieving Data from a Local ROLAP Cube**

When communicating with a local relational OLAP (ROLAP) cube file, you must have a connection to a relational data provider. The local cube file stores the structural definition of the cube but not actual or precalculated data. To retrieve the data itself, the connection to the tabular data provider is used. This process is transparent to the client application. This transparency results in smaller cubes than are possible using a multidimensional OLAP (MOLAP) storage mode. However, the performance of such a cube is less than that of a local MOLAP cube due to the processing requirements of calculating the aggregate function at run time.

**Building Local Cube Files**

You can also build a local ROLAP cube file that is based on a relational OLE DB provider by using the DEFER_DATA option in the INSERT INTO statement. Queries that are used to define the local cube file are passed to the data source where they are resolved. The resulting OLE DB recordsets are interpreted by PivotTable Service and used to build the local cube.

**See Also**

- [Building Local Cubes](#)
- [Using DRILLTHROUGH to Retrieve Source Data](#)
INSERT INTO Statement
Connected to a Local Cube File or Data Mining Model

The processes for connecting to a local cube file and a local mining model are almost identical.

Retrieving Data from a Local MOLAP Cube

PivotTable® Service connects to the local cube file in the same way that it connects to any other data source. PivotTable Service processes queries against the local cube file and returns data to the application. The client application can access the dimensions, levels, properties, and so on of a particular cube. No connection to a remote server is required, except to create the cube. This diagram illustrates communication between PivotTable Service and a local MOLAP cube file.

Retrieving Data from a Local Data Mining Model

PivotTable Service connects to a data mining model in the same way it connects to any other data source. Prediction queries can be passed to a local data mining model in the same fashion as they are made and passed to a server model. A connection to a remote server is not required, except to create the mining model file.

See Also

Connecting to a Data Source
Analysis Services Programming
Organization of Multidimensional Data

PivotTable® Service supports three different hierarchy types. The following table contains links to the topics that discuss each type, and how to address compatibility with existing client applications developed for Microsoft® SQL Server™ 7.0 OLAP Services.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Balanced Hierarchies</strong></td>
<td>Describes how SQL Server 2000 Analysis Services implements dimension hierarchies in which all leaf nodes of a level are the same distance from the root node</td>
</tr>
<tr>
<td><strong>Ragged Hierarchies</strong></td>
<td>Describes how Analysis Services implements dimension hierarchies in which one or more levels do not contain members in one or more branches of the hierarchy</td>
</tr>
<tr>
<td><strong>Unbalanced Hierarchies</strong></td>
<td>Describes how Analysis Services implements dimension hierarchies in which leaf nodes differ in their distances from the root node</td>
</tr>
<tr>
<td><strong>Using the MDX Compatibility Property</strong></td>
<td>Describes the use of this property to address compatibility with existing client applications</td>
</tr>
</tbody>
</table>
Balanced Hierarchies

A balanced hierarchy is one in which the presence of children for any given member does not depend on its value. Instead, it depends on the level of that member in the hierarchy. For example, a dimension based on time might have the following structure.

This structure applies in most situations. Some natural variations may occur; for example, an application may use a Julian calendar instead of a traditional one. In this case, you could use the Julian hierarchy exclusively or define multiple hierarchies for the dimension containing time information.

See Also

Balanced and Unbalanced Hierarchies
Ragged Hierarchies

A ragged hierarchy is one in which one or more levels are skipped in the members of the hierarchical structure. For example, a Geography dimension might have the following structure.

For countries that have states or provinces, such as Canada or Mexico, this dimension works well. Consider the case of Washington, D.C.: The parent of this member of the City level is USA, which is not a member of the State level. However, other siblings of Washington D.C., such as Los Angeles and New York, have parents that are members of the State level. This is an example of a ragged hierarchy.

See Also

Ragged Hierarchies
Unbalanced Hierarchies

An unbalanced hierarchy is one in which the children of a member may or may not have children themselves, depending on the value of that child. PivotTable® Service supports parent-child structures to contain unbalanced hierarchies. Consider the case of an organizational chart in a company. Executive assistants may report directly to the CEO, a director, or a manager. Technicians may report to a lead technician, a manager, or a technical sales person. In these hierarchies, the level of the individual is less important than the individual’s relationship to the superior. Relationships of this type are often referred to as parent-child relationships and are often defined in relational databases using self-referential joins.

See Also

Balanced and Unbalanced Hierarchies
Using the MDX Compatibility Property

Empty positions in a hierarchy can affect some functions in Microsoft® SQL Server™ version 7.0 OLAP Services. For example, `DrillDownLevel([Romania])` returns an empty set because Romania has no states or provinces. For this reason, a connection string property, **MDX COMPATIBILITY**, is provided for backward compatibility with client applications developed using SQL Server 7.0 OLAP Services. The following table describes this property.

<table>
<thead>
<tr>
<th>MDX COMPATIBILITY property value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Default)</td>
<td>The same as Value 1</td>
</tr>
<tr>
<td>1</td>
<td>Compatible with SQL Server 7.0 OLAP Services</td>
</tr>
<tr>
<td>2</td>
<td>Compatible with SQL Server 2000 Analysis Services</td>
</tr>
</tbody>
</table>

If the **MDX COMPATIBILITY** property value is set to 1, a client application using `DrillDownLevel([Romania])` receives a single dummy member for the States/provinces level of Romania, which can then be drilled down again, incrementally, to provide access to the city members of the Romanian geography hierarchy. When this property is set to 2, PivotTable® Service returns an empty set for this function. The following table shows which functions are affected by empty positions in a hierarchy.

<table>
<thead>
<tr>
<th>Function called on empty positions</th>
<th>Results (MDX COMPATIBILITY=1)</th>
<th>Results (MDX COMPATIBILITY=2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllMembers</td>
<td>Returns the name of the empty level in the hierarchy</td>
<td>Ignores empty positions on the resulting axis</td>
</tr>
<tr>
<td>Members Descendants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range (:)</td>
<td>Returns the name of the empty level in the hierarchy</td>
<td>Returns an error</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Children</code></td>
<td>Returns the name of the empty level in the hierarchy</td>
<td></td>
</tr>
<tr>
<td><code>DrillDownLevel</code></td>
<td>Returns the name of the empty level in the hierarchy</td>
<td></td>
</tr>
<tr>
<td><code>DrillDownLevelTop</code></td>
<td>Returns an empty set</td>
<td></td>
</tr>
<tr>
<td><code>DrillDownLevelBottom</code></td>
<td>Returns the name of the empty level in the hierarchy</td>
<td></td>
</tr>
<tr>
<td><code>DrillUpLevel</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DrillDownMember</code></td>
<td>Returns the name of the empty level in the hierarchy</td>
<td></td>
</tr>
<tr>
<td><code>DrillDownMemberTop</code></td>
<td>Skips empty positions in the hierarchy and returns the first position that is not empty</td>
<td></td>
</tr>
<tr>
<td><code>DrillDownMemberBottom</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DrillUpMember</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>LastPeriods</code></td>
<td>Returns the normal value of the requested cell</td>
<td></td>
</tr>
<tr>
<td><code>YTD</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>QTD</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>MTD</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>WTD</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>AddCalculatedMembers</code></td>
<td>Returns the normal value of the requested cell</td>
<td></td>
</tr>
<tr>
<td><code>VisualTotals</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Parent</code></td>
<td>Returns the normal value of the requested cell</td>
<td></td>
</tr>
<tr>
<td><code>Ancestor</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>FirstChild</code></td>
<td>Skips empty positions in the hierarchy and returns the first position that is not empty</td>
<td></td>
</tr>
<tr>
<td><code>LastChild</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ClosingPeriod</code></td>
<td>Returns the name of the empty level in the hierarchy</td>
<td></td>
</tr>
<tr>
<td><code>OpeningPeriod</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Cousin</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ParallelPeriod</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PrevMember</code></td>
<td>Returns the normal value of the requested cell</td>
<td></td>
</tr>
<tr>
<td><code>NextMember</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Lead</code></td>
<td>Returns the resultant real members at that level</td>
<td></td>
</tr>
<tr>
<td><code>Lag</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See Also

[MDX](#)

[MDX Compatibility Property](#)
Analysis Services Programming
Advanced Data Mining and Analysis

In this release, Microsoft® SQL Server™ 2000 Analysis Services introduces a new feature, data mining, that integrates significant data analysis and prediction capabilities into Analysis Services. PivotTable® Service enables clients to interact with these new data mining features. For more information about data mining in Analysis Services, see Data Mining Models and Data Mining Columns.

PivotTable Service supports data mining by providing support services that are very similar to the services it provides for online analytical processing (OLAP). For example, PivotTable Service can create and maintain local data mining models just as it can create and maintain local cubes. To create a data mining model on an Analysis server, you must use Decision Support Objects (DSO). For more information about building mining models using DSO, see Data Mining Examples.

Two data mining algorithms are included with Analysis Services: Microsoft Decision Trees and Microsoft Clustering. The decision trees algorithm is based on the notion of classification. The clustering algorithm uses an expectation-maximization method to group records into clusters (or segments) that exhibit some similar, predictable characteristic. For more information, see Microsoft Clustering.

The following table describes topics that contain information about data mining in PivotTable Service. For detailed information about creating and using data mining models, including special functions for mining models, mining model XML format, and examples, see the OLE DB for Data Mining specification, available on the Microsoft OLE DB Web page at the Microsoft Web site.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building a Local Data Mining Model</td>
<td>Describes the process of building local data mining models</td>
</tr>
<tr>
<td>Training a Local Data Mining Model</td>
<td>Describes how to process a local data mining model with training data</td>
</tr>
<tr>
<td>Predictions and Results of Data Mining</td>
<td>Describes how to run prediction queries against a data mining model</td>
</tr>
</tbody>
</table>
and how to browse its contents

See Also

Data Mining Models
Data Mining Algorithms
Building a Local Data Mining Model

Conceptually, the structure of a local data mining model is similar to that of a table in a relational database. Like tables, data mining models are defined by the column definitions they contain. However, unlike a table in Microsoft® SQL Server™ 2000, the columns in a data mining model can contain nested tables. SQL Server 2000 Analysis Services supports two kinds of data mining models: models that are based on OLAP cubes, and models that are based on relational tables (or, more accurately, a rowset from an OLE DB provider).

The syntax for defining a mining model is also similar to that for defining a table. There are two different forms of the CREATE MINING MODEL statement, one for OLAP mining models and one for relational mining models.

Building a Model Based on an OLAP Cube

To create a mining model that is based on an OLAP cube, use the CREATE OLAP MINING MODEL statement. The general form of the statement is as follows:

CREATE OLAP MINING MODEL <Model Name> FROM <Case Cube Name> (<Cube Members>) USING <Algorithm Name>

The <Model Name> token specifies the name of the model that will be created. The physical location for this model will be the directory specified by the Mining Location property. If the Mining Location property is not specified in the connection string, the mining model created by this statement will have connection scope, and it will only exist for the duration of the session. The <Case Cube Name> token is the name of the cube that contains the training cases for the model <Cube Members>. Finally, the <Algorithm Name> token contains the name of the mining model algorithm that will be used to create the model. This token can have one of two values: Microsoft_Decision_Trees or Microsoft_Clustering.

The following example creates an OLAP mining model that predicts the Member Card Type property for members of the Customers dimension:
CREATE OLAP MINING MODEL [MyOlapModel] FROM [Sales]
(
    CASE
    DIMENSION [Customers]
    LEVEL [Name]
    PROPERTY [Marital Status],
    PROPERTY [Education],
    PROPERTY [Member Card Type] PREDICT
    )
USING Microsoft_Decision_Trees

The mining model that this example defines is based on the Sales cube in the current database (that is, the default database for this session). The three columns that will be included in this mining model are defined next. Each column is based on a member property that applies to each member contained in the Name level of the Customers dimension. The presence of the PREDICT specifier in the definition for the last column, *Member Card Type*, indicates that the column is predictable.

**Building a Model Based on a Relational Database Table**

You define relational mining models (that is, models that are based on tables in a relational database) by specifying the columns to be included in the model. Because the format and structure of the source data is not known in advance, each column is defined by a name, the data type of its content, its statistical nature, and whether the column will be predictable in a query. The general form of the statement that creates a relational mining model is as follows:

CREATE MINING MODEL <Model Name> (<Column Members>) USING <Algorithm Name>

For example, consider the following relational mining model definition:

CREATE MINING MODEL [MemberCards]
(
    [customer Id] LONG KEY ,
    [Yearly Income] TEXT DISCRETE ,
)
[Member Card Type] TEXT DISCRETE PREDICT,
[Marital Status] TEXT DISCRETE
)
USING Microsoft_Decision_Trees

In this example, a mining model named MemberCards is defined using the CREATE MINING MODEL statement. The syntax of this statement is similar to that of the CREATE TABLE statement in SQL. The columns that make up this mining model are named and their types are defined with additional information concerning the content they contain. The Member Card Type column is specified as being predictable by using the PREDICT specifier in its column definition.

**Columns That Contain Nested Tables**

You may want to create a mining model that contains a column with a nested table. In this case, use the TABLE type specifier in the CREATE MINING MODEL statement:

```
CREATE MINING MODEL [Age Prediction]
(
    [Customer ID] LONG KEY,
    [Gender] TEXT DISCRETE,
    [Age] DOUBLE DISCRETIZED() PREDICT,
    [Product Purchases] TABLE
    (
        [Product Name] TEXT KEY,
        [Product Type] TEXT DISCRETE RELATED TO [Product Name],
        [Quantity] DOUBLE NORMAL CONTINUOUS
    )
)
USING [Decision Trees]
```

In this example, the **Product Purchases** column contains a nested table that contains three columns: **Product Name**, **Quantity**, and **Product Type**. The first column in the nested table is a key column. The next column in the nested table,
**Product Type**, is related to the **Product Name** column in a hierarchical relationship. The last column, **Quantity**, contains a floating-point number that is statistically normal and continuous across its domain (as opposed to having discrete values within the domain).

The last clause in the CREATE MINING MODEL states that the model should be built using the Microsoft Decision Trees data mining algorithm.

For more information about the CREATE MINING MODEL statement, see [CREATE MINING MODEL Statement](#).

For more information, see the OLE DB for Data Mining specification.

**See Also**

[Data Mining Models](#)

[Data Mining Columns](#)
Training a Local Data Mining Model

In data mining, training is the process that inserts the data into the model that will be used as the basis for making predictions. The INSERT INTO statement is used to accomplish this task. The syntax of the statement depends on the kind of object on which the model is to be based. Microsoft® SQL Server™ 2000 Analysis Services supports two different kinds of base objects for data mining models: OLAP cubes and relational tables.

The process of training a mining model can be broken down into two parts. First, the columns that define the model's structure are populated with content from the data source. Second, the content is analyzed using the algorithm specified in the CREATE MINING MODEL statement. The results of this analysis are stored in the mining model as a collection of nodes. These nodes can be browsed using the MINING_MODEL_CONTENT schema rowset or by executing a content query against the mining model.

Training an OLAP Data Mining Model

For OLAP mining models, the general form for the INSERT INTO statement is:

```
INSERT INTO <model Name>
```

No column names or other source data is needed to train the model. This is because the structure of the mining model is based on a cube and is therefore known in advance. No other steps are necessary to complete the processing of the model.

Training a Relational Mining Model

Training a data mining model based on a table in a relational database is slightly more complicated than processing an OLAP mining model. When training a relational data mining model, the columns to be populated must be specified explicitly along with their data source. This is because the INSERT INTO command in the relational data mining model does not have the same information available as an OLAP mining model. The general form of the command for training relational mining models is as follows:
To understand the process of training a relational mining model, consider the example of a model with the columns Name, Age, and Hair Color. The following statement can be used to populate this model:

**The OPENROWSET Statement**

Analysis Services does not support the use of direct SQL SELECT queries to retrieve data rowsets for training data mining models. Instead, it supports the OPENROWSET statement, which enables applications to specify an external query in place of actual data or an SQL SELECT statement. The syntax of this command is as follows:

```
OPENROWSET ( '<Provider Name>', '<Connection String>', '<Query Syntax>' )
```

The `<Provider Name>` token must correspond to an OLE DB compliant data source, such as 'SQLOLEDB' or 'MSOLAP'. The `<Connection String>` token must correspond to a valid connection string for the data source, minus the provider property. Finally, the `<Query Syntax>` token should correspond to a valid query in the supported language of the provider that will return the desired rowset. In this example, the provider used is the SQL Server 2000 OLE DB provider. The **Connection String** property specifies that the **FoodMart 2000** database is to be used as the default database for the query. Finally, the query itself is defined as a standard Transact-SQL query that returns three columns from a table called **Customers**.
Training Models That Include Nested Columns

The SHAPE command must be used to populate the columns in a nested table. The general format for this command is as follows:

SHAPE { <Rowset Query> }
APPEND
(
  { <Rowset Query> }
  RELATE <Parent Key Column> TO <Child Key Column>
)
AS <Nested Column Name>

The following example demonstrates populating a nested table by using the shape provider:

INSERT INTO [Age Prediction]
(  /* Define the columns of the case table */
   [Customer Id], [Gender], [Age],
   /* Define the columns of the nested table */
   [Product Purchases](SKIP, [Product Name], [Quantity], [Product Type])
)
SHAPE
{
  OPENROWSET ('SQLOLEDB','INITIAL CATALOG=FoodMart 2000;',
  'SELECT [Customer Id], [Gender], [Age] FROM Customers ORDER BY [Customer ID]')
}
APPEND
(
  {
    OPENROWSET ('SQLOLEDB','INITIAL CATALOG=FoodMart 2000;',
    SELECT [CustID], [Product Name], [Quantity], [Product Type] FROM Sales ORDER BY [CustID])
    RELATE [Customer Id] To [CustID]
  }
) AS [Product Purchases]
In this example, a column in the case table called [Product Purchases] is populated by a nested table. The names of the columns in this nested table are defined inside the parentheses of the fourth line of the INSERT INTO statement. The SHAPE command is then used to define the columns that will be used to populate the case table. The SHAPE clause defines columns that will be used to populate the nested table as columns that are contained within the APPEND clause. The relationship between the case table and the nested table is then defined by using the RELATE clause. The result of the SHAPE command is then aliased to be the same as that of the original column that contained the nested tale.

When using the shape command it is important to use the ORDER BY clause to enforce the order of columns in the query. Failure to use this clause may cause some or all data to be ignored in your nested tables.

See Also

Data Mining Models
Data Mining Columns
Predictions and Results of Data Mining

For retrieving information from a processed data mining model, Microsoft® SQL Server™ 2000 Analysis Services supports two different kinds of queries.

<table>
<thead>
<tr>
<th>Query type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction query</td>
<td>Returns the predicted values of a set of columns, whose contents are unknown, after applying the results contained within a processed data mining model to them</td>
</tr>
<tr>
<td>Content query</td>
<td>Returns information about the values and rules discovered by training the mining model</td>
</tr>
</tbody>
</table>

Prediction Queries

Prediction queries allow the user to make predictions for unknown case sets using contents from a previously trained data mining model. Prediction queries are run by means of the SELECT statement:

```
SELECT [FLATTENED] <SELECT-expressions> FROM <mining model> PREDICTION JOIN <source data query> ON <join condition> [WHERE <WHERE-expression>]
```

The <Source Data Query> token identifies the set of new cases that will be predicted. <Mining Model Name> identifies the mining model that will be used to generate the predictions.

After the source data has been identified, a relationship between it and the data in the mining model must be defined. This is done using the ON clause of the PREDICTION JOIN statement.

Example

The following example attempts to predict the age of customers using the Age Prediction data mining model and the Customers and Sales cubes:

```
SELECT t.[Customer ID], [Age Prediction].[Age]
```

For more information, see the OLE DB for Data Mining specification.

**Content Queries**

Browsing the content of a data mining model can provide important insight into the data. For example, it may expose patterns or trends that can be used to predict new data points or train expert systems. The content of the data mining model depends on the algorithm that generated it, and it can vary widely from algorithm to algorithm.

Querying the model directly will return contents of the model. For example, consider the following query:

```
SELECT * FROM MyOlapModel.CONTENT
```

This example provides a result table whose structure is the same as that of the MINING_MODEL_CONTENT schema rowset.

More sophisticated queries against the content in a data mining model are also
possible. For example, consider the following content query:

```
SELECT Age
    FROM HairColorPredictDMM.Content
    WHERE Gender = 'Male' and HairColor = 'Black'
```

This query returns all of the nodes that are concerned with black-haired men.

<table>
<thead>
<tr>
<th>Gender</th>
<th>Age</th>
<th>HairColor</th>
<th>P(HairColor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>2</td>
<td>Black</td>
<td>.667</td>
</tr>
<tr>
<td>Male</td>
<td>91</td>
<td>Black</td>
<td>.300</td>
</tr>
<tr>
<td>Male</td>
<td>45</td>
<td>Black</td>
<td>.667</td>
</tr>
<tr>
<td>Male</td>
<td>NULL</td>
<td>Black</td>
<td>.600</td>
</tr>
</tbody>
</table>

**See Also**

[Predict](#)
Analysis Services Programming
Working with OLAP Data

When working with OLAP data, it is possible to retrieve that data in two different forms: datasets and flattened rowsets. Datasets store the results of a query in an axis structure that is determined by the shape of the results. Flattened rowsets have only two axes. In this case, the data in the additional axes are stored by mapping them onto the two existing axes.

Multidimensional DataSets

A dataset is a multidimensional representation of the results of a Multidimensional Expressions (MDX) query. Each dimension that is returned by the query is represented in the dataset by an axis. The members of each dimension make up the coordinates on the axis. The measures are returned in cells. Each cell is located at the intersection of the coordinates along each axis.

In OLE DB for OLAP, the Dataset object provides methods for interacting with the axes and cells the dataset contains. The primary interface for working with the Dataset object is IMDDataset. Using this interface it is possible to retrieve the value of a cell (IMDDataset::GetCellData), retrieve a pointer to the Command object interface that created the cellset(IMDDataset::GetSpecification), or retrieve information about the axes(IMDDataset::FreeAxisInfo, IMDDataset::GetAxisInfo, IMDDataset::GetAxisRowset). For more information, see the OLE DB documentation.

In Microsoft® ActiveX® Data Objects (Multimensional) (ADO MD), the Cellset object contains a collection of cells, axes and properties. To access an individual cell in the cellset, use the Item() method. Cells can be specified by providing one of the following:

- The position numbers of the cell.
- The member names (that is, the tuple) for the cell.
- The ordinal position of the cell.
For more information, see the ADO MD documentation.

**Flattened Rowsets**

Whenever the results of an MDX query that returns data on more than two axes must be represented in two dimensions, such as in an OLE DB Rowset object or an ADO Recordset object, the results must be mapped onto the two dimensions using a process called *flattening*. For more information about flattening rowsets, see the OLE DB for OLAP documentation.

In OLE DB, the **Rowset** object exposes the results of a query (either MDX or SQL) in a tabular form. It is represented by a set of rows. Each row contains a set of columns that contain the data returned from the query. The primary interface for interacting with Rowset objects is the OLE DB **IRowset** interface. In addition to this interface, the following helper interfaces are used to navigate through the rowsets: **IAccessor**, **IColumnsInfo**, **IConvertType**, and **IRowsetInfo**. For more information, see the OLE DB documentation.

In ADO, the **Recordset** object represents the results of a query in tabular form. Each Recordset object consists of a collection of **Fields** and **Properties**. The **IMDDataset::** collection represents the columns in the query results. The **Properties** collection contains the properties that describe the rowset. ADO provides numerous methods and objects for navigating through returned recordsets.

**Note**  Because recordsets that contain flattened rowsets are read-forward only, **Recordset** object methods such as **MoveFirst**, **MovePrevious**, and **RecordCount** return the error **0x80004001 – Not Implemented**.

For more information, see the ADO documentation.
Analysis Services Programming
Calculated Members

Calculated members are members whose values depend on an expression rather than the value of a cell.

You can define a calculated member using one of the following scopes:

Query scope

The calculated member can be used only within the query in which it is defined. Use the WITH clause in the SELECT statement.

Session scope

The calculated member can be used only within the session in which it is defined, but can be used by multiple queries. Use the CREATE MEMBER statement.

A calculated member can be stored in a local cube if a CREATE MEMBER statement is specified in the COMMAND clause of the CREATE CUBE statement.

Use the following code to create a calculated member:

```
CREATE CUBE MYWAREHOUSE (  
  DIMENSION . . .  
  . . . ,  
  COMMAND (CREATE MEMBER [MYWAREHOUSE].[MEASURES].[WAREHOUSEPROFIT] AS '([MEASURES].[WAREHOUSESALES] - [MEASURES].[WAREHOUSECOST])')  
)
```

**Note**  You must use single quotes (') to enclose the expression for the calculated member even though the OLE DB specification does not require these quotes.

For more information, see [CREATE CUBE Statement](#).

Custom Rollups
In addition to the standard rollup (that is, aggregate) functions **Sum**, **Min**, **Max**, and **Count**, more sophisticated custom rollup functions can be defined for any given member in the CREATE CUBE statement. One common usage scenario is the inventory problem where inventory levels are not summed along the Time dimension, as they might be for other dimensions. For example, if you have one item in inventory on seven consecutive days, you do not have a total of seven items for the week. By using the **FirstChild** function in Multidimensional Expressions (MDX) to define a custom rollup formula, you can automatically roll up closing balances along time.

**See Also**

[Calculated Members](#)
Analysis Services Programming
Managing the Client Cache

PivotTable® Service maintains a local cache on the client computer. When PivotTable Service executes a query, the cache is used to store the data locally. If the data is used more than once, PivotTable Service does not need to request the data multiple times. The contents of this cache and when they are updated changes with the value of the Default Isolation Mode property. The following table describes this property's values.

<table>
<thead>
<tr>
<th>Property value</th>
<th>Cache mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>ISOLATION</td>
<td>The cache is invalidated when a query or a REFRESH CUBE statement is executed. The cache is never refreshed.</td>
</tr>
<tr>
<td>FALSE</td>
<td>NON-ISOLATION</td>
<td>The cache is invalidated when a REFRESH CUBE statement is executed or when PivotTable Service receives a refresh notification from Microsoft® SQL Server™ 2000 Analysis Services. The cache is refreshed when a query is executed.</td>
</tr>
</tbody>
</table>

Isolating the Client Application from External Updates

In isolation mode, the cache is populated with axis and cell data each time a query is executed. If the original source data is changed, for example, as a result of a writeback by another user, the client cache is not updated. In OLE DB, this value is referred to as Repeatable Read mode (ISOLATIONLEVEL_REPEATABLEREAD).

In nonisolation mode, the cache is populated with axis data each time a query is executed. However, the cell data for the query is not populated. The cell data is populated only when the client application itself requests it (that is, refers to it). If the cell data is never referred to, the cache never receives it. The cache may
also be refreshed if Analysis Services sends a refresh notification to PivotTable Service; in this case, PivotTable Service invalidates the existing cache. If the client application requests either axis or cell data, PivotTable Service refreshes its cache from Analysis Services. In OLE DB, this value is referred to as Read Committed mode (ISOLATIONLEVEL_READCOMMITTED).

In either mode, executing a REFRESH CUBE statement refreshes the cache.

**See Also**

*Isolation Levels*

*Default Isolation Mode Property*
Analysis Services Programming
Transactions in Analysis Services

PivotTable® Service supports transaction management for allocations and writebacks to cubes on the Analysis server. An allocation or a writeback to a cube changes a cached copy of that cube in order to analyze the effects of the change. This transaction process enables users to:

- Perform what-if analysis on cubes that are not write-enabled (that is, that do not support writeback).

- Perform what-if analysis on local cubes, which cannot be write-enabled.

- Perform what-if analysis on cubes to which they have only read permission.

- Perform what-if analysis without committing the updates.

- Make multiple what-if changes and reverse or alter some before committing all changes at once.

PivotTable Service supports these transactions by supporting the use of the Microsoft® ActiveX® Data Objects (ADO) Connection objects transaction methods.

Changes made during what-if analysis are visible only to the user who makes them; they are not committed to a shared cube until a Commit transaction is performed. Therefore a user may make a change to a cube's displayed data transparently without affecting other users. The changes are recorded in a writeback partition (that is, a table), separate from the cube's underlying source tables. After a successful writeback, all users who are synchronized with the server see the effect of the writeback change reflected in the cube.

In PivotTable Service, a new transaction is implicitly started whenever a session begins. Each transaction must either be explicitly completed by executing the ADO Commit transaction method, or be rolled back using the rollback
transaction method. If a transaction is not completed properly, then the transaction and all the changes it contains are automatically rolled back when the session ends. A new transaction begins implicitly when the preceding transaction is completed.

Automatic commits do not occur. Changes are not propagated to the cube's writeback table and will not be visible to other users unless the Commit transaction method is used.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating Cubes</td>
<td>Describes updating values in a cube</td>
</tr>
<tr>
<td>Transaction Scope</td>
<td>Describes the effect of scope on a cube transaction</td>
</tr>
<tr>
<td>Synchronization of Client and Server</td>
<td>Contains information about synchronizing client and server data</td>
</tr>
<tr>
<td>Cumulative Effect of Transactions on Data</td>
<td>Describes how PivotTable Service handles multiple users working with the same data</td>
</tr>
<tr>
<td>Isolation Levels</td>
<td>Contains information about isolation levels, which control when changes are made visible to users</td>
</tr>
<tr>
<td>Committing a Transaction</td>
<td>Describes the use of the ADO Commit property in PivotTable Service</td>
</tr>
<tr>
<td>Commit Time-out</td>
<td>Describes how PivotTable Service handles commits that fail due to errors</td>
</tr>
</tbody>
</table>
Updating Cubes

There are three ways to update information in a cube:

- Update the fact table and reprocess the cube.
- Write back to leaf members of the cube.
- Use cell allocation on nonleaf members.

Reprocessing a Cube

This method of updating a cube's contents depends on the context of the cube itself. If the cube resides on the Analysis server, then the Decision Support Objects (DSO) Process method (of the MDStore interface) should be used to process the cube using the existing dimensions, measures, aggregations and so on. For more information about DSO, see Decision Support Objects. For more information about the Process method of the MDStore interface, see Process (MDStore Interface).

Local cubes can use the UPDATE CUBE statement for what-if analyses, but the allocations cannot be saved. That is, commit will fail if it is executed for the UPDATE CUBE statement. Because local cubes cannot have a writeback partition, writeback always fails against a local cube. Therefore, permanent changes to a local cube must be made by changing the local cube's fact table and rebuilding.

For more information on creating local cubes, see Connected to an OLE DB Provider or Building Local Cubes.

Writeback

Writebacks (that is, updates) can be accomplished on atomic cell members of write-enabled cubes. These updates result in a new aggregate value being propagated up through the cell's parent members.
Because data at higher levels is represented as a precalculation of data at lower levels, writebacks are permitted only on cells at the lowest level (that is, atomic or leaf cells) of a cube's data. The atomic cells coming from the fact table are represented in the cube by a single member in the lowest level of each dimension or measure in the cube. Updates at these levels are saved to a writeback table that stores the deltas for each value. The updated value is then propagated up through the affected aggregate members by PivotTable® Service.

If you want to update a higher-level member, use the UPDATE CUBE statement instead. This will assist you in preventing inconsistent results from being entered into the cube.

**Note** You cannot write back to local cubes. You can reprocess them by executing their original CREATE CUBE and INSERT INTO statements in a connection string.

Changes to a cube may not be immediately visible to other client applications that are connected to the cube, depending on their cache settings. For more information, see [Managing the Client Cache](#) and [Isolation Levels](#).

### Allocations in a Cube

When the value of a nonatomic cell is changed, the cells that contribute to that cell need to be updated to avoid inconsistent data within the cube. The UPDATE CUBE statement provides this facility. The new value of a nonatomic cell must be allocated among all of its constituent cells. The developer determines the method by which this allocation is made at design time.

The following table describes the allocation methods that are available.

<table>
<thead>
<tr>
<th>Allocation method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal allocation</td>
<td>Each constituent cell is assigned an equal value</td>
</tr>
<tr>
<td>Equal increment</td>
<td>Every constituent cell will be changed according to an incremental value</td>
</tr>
<tr>
<td>Weighted allocation</td>
<td>Each constituent cell will be assigned an equal value that is weighted against a formula</td>
</tr>
<tr>
<td>Weighted increment</td>
<td>Every constituent cell is changed incrementally according to a weighting formula</td>
</tr>
</tbody>
</table>
**IMPORTANT** When supplying expressions for the allocation of a value, the client application must ensure that the expression assigns values whose aggregate value equals the originally allocated value. Additionally, the application must take into account the allocation on all dimensions concurrently.

For the purpose of transactions management, all of the operations required to make a cell update are considered to be monatomic. That is, if one atomic cell update fails, then all of them will fail, and the update itself will fail.

**Note** Allocations on local cubes cannot be saved by use of the transaction COMMIT method. Any allocations made on a local cube are only present while the connection to the local cube is still active (that is, for the duration of the session).

**See Also**

UPDATE CUBE Statement  
Writing Back to Cells and Cube Transactions  
Write-Enabled Cubes  
Write-Enabled Dimensions  
Using Writebacks
Analysis Services Programming

**Transaction Scope**

The scope of a transaction on the Analysis server is limited to a single cube. Transactions that contain updates to a single cube either commit or fail on the cube as a complete atomic operation.

**IMPORTANT** Transactions that contain updates to more than one cube are not guaranteed to be atomic. In the case when a transaction contains updates that affect more than one write-enabled cube, it is possible for the updates to commit for some cubes but fail for others. This includes the case when updates are being applied to a virtual cube that contains more than one underlying write-enabled cube; it is possible for a transaction applied to the virtual cube to commit on one or more of the underlying cubes but fail on others.

If a transaction that includes updates to multiple cubes fails due to time-out, it is safe to attempt to commit the same transaction again. The transaction will be applied only to cubes that were not updated in the previous attempt. However, it is recommended that client applications use a separate transaction for each cube when updating multiple cubes.
**Synchronization of Client and Server**

In writeback scenarios, the frequency of client/server synchronization determines when a user sees the most recent updates to a cube. Some queries are resolved entirely from client cache; if the server cube has been updated since the last synchronization, the results of such a query will not reflect the updates until the cache is refreshed from the server.

The frequency of client/server synchronization can be controlled with the **Auto Synch Period** property. For more information, see [Auto Synch Period Property](#).

For mining model queries, you can use the **Mining Execution Location** property to control synchronization of the client mining model query cache. For more information, see [Mining Execution Location Property](#).
Cumulative Effect of Transactions on Data

When multiple users are connected to a cube and their changes are in conflict, the last changes made are the ones that take effect. Updates recorded in the writeback table of a cube are cumulative, so the cube is displayed with the net effect of all changes in the writeback table. The last user to commit an update to a cell determines the displayed values of the cell and all precalculated cells that are derived from it.

The act of updating cells is performed atomically for each cube. That is, each committed update is recorded separately in the writeback table.
Isolation Levels

PivotTable® Service supports the read-committed and repeatable-read isolation levels (that is, isolated and nonisolated cache modes). By default, the visibility of changes made concurrently by others is read-committed, meaning that only committed updates are visible to a command or query. That is, commits executed by other users are immediately available to PivotTable Service. The repeatable-read isolation level provides a higher degree of isolation. In this mode, the client cache is frozen when a command or query is opened, and remains frozen until the command or query is closed.

You can set the isolation level to isolated using the Default Isolation Mode property.

See Also

Default Isolation Mode Property


**Committing a Transaction**

Use the Microsoft® ActiveX® Data Objects (ADO) **Commit** transaction method to commit updates to the writeback table of a write-enabled cube. An error results if the transaction attempts to commit updates to a table other than the writeback table or if the cube is not write-enabled.

If a transaction includes updates for multiple cubes, the updates may be successful for some cubes and unsuccessful for others. Therefore, a transaction should only include updates for a single cube. For more information, see [Transaction Scope](#).

Some possible causes of commit failure are commit time-out during a writeback attempt, attempting to commit to a cube that is not write-enabled, or network errors. For more information about errors in transaction processing, see the ADO documentation.

**See Also**

[Commit Time-out](#)
Commit Time-out

If a commit is not successful because a client application's attempt to update a cube's writeback table times out (that is, the time spent attempting the commit reached the value of the DBPROP_MSMD_WRITEBACK_TIMEOUT property), the following error message is raised in the connection objects Errors collection:

Server unable to accept transaction at this time. Transaction pending or in progress.

In this case, the transaction's state is the same as it was immediately before the commit attempt. The client application can again attempt to commit, attempt to roll back, or allow more what-if changes.

The preceding message is produced as a result of the following return code from the ITransaction::Commit method:
MSMD_E_TRANSACTION_COMMIT_TIMEOUT.

Increasing the value of the DBPROP_MSMD_WRITEBACK_TIMEOUT property can reduce the number of time-outs.

For more information, see Writeback Timeout Property.
Analysis Services Programming
Security in PivotTable Service

PivotTable® Service supports security in two ways: by providing security at various levels of the server object model, and by supporting authentication of users.

Server Object Model Security

Different levels of the server object model handle security in different ways:

Database, cube, and mining model security

Database administrators (DBAs) can use roles to grant read and write permissions for the members of a database or an individual cube. Roles that grant read permission can also be created for mining models. These roles are available in the database schema rowset. For more information, see Roles.

Member security

Individual members of a cube or mining model can be secured independently from a level, dimension, and so on. Members that are secured in this manner are invisible to client applications that do not have permission to access them. No errors are raised and placeholders are not returned.

Cell Security

Queries that involve these secured members will return an error. Updates to a secured member will also return an error. The value of this error depends on the value of the Secured Cell Value property. For more information, see Cell Security.

Drillthrough security

Read permission for the Multidimensional Expressions (MDX) Drillthrough command can be granted for the entire cube using the cube's role. Drillthrough requests against secured data return an error.

User Authentication

Authentication is the process by which a user is positively identified to
determine the permissions the user has been granted. Microsoft® SQL Server™
2000 Analysis Services supports three authentication providers:

- NTLM protocol (Windows authentication)
- Kerberos
- Negotiate
- Anonymous user

After authentication for a user has been obtained, a user can connect to a
database using any role of which he or she is a member by using the Roles
property in the connection string for the session, as long as that role has been
granted access to the database.

See Also

Secured Cell Value Property
SSPI Property
Roles Property
Analysis Services Programming
## Client Operations in PivotTable Service

The following topics describe common client operations that can be performed using PivotTable® Service.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error and Exception Handling</td>
<td>Describes recovering from multiple errors and determining the original source of an error using the <strong>Err</strong> object.</td>
</tr>
<tr>
<td>Connecting to a Data Source</td>
<td>Illustrates the various methods for connecting a client application to a data source.</td>
</tr>
<tr>
<td>Retrieving Schema Information</td>
<td>Describes how OLE DB schema rowsets can be retrieved.</td>
</tr>
<tr>
<td>Retrieving Data</td>
<td>Shows how to execute queries and commands. It also explains different methods for retrieving the returned data.</td>
</tr>
<tr>
<td>Updating Information in a Cube</td>
<td>Relates the different ways a cube or dimension can be modified.</td>
</tr>
<tr>
<td>Building Local Cubes</td>
<td>Illustrates how to build a local cube.</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Error and Exception Handling

The first error trapped by Microsoft® Visual Basic® can be ambiguous if you are working with Microsoft ActiveX® Data Objects (ADO), and if the error is displayed alone. You should retrieve any additional error information. ADO provides an additional layer for handling exceptions that result from data operations. You can use the Errors collection of the Connection object to retrieve information about more than one kind of error.

Examples

A. Using the ADO Connection Object

The following code example introduces an error into the connection string of the Connection object. This error induces two new errors in addition to the standard error passed through by the Visual Basic Err object. These errors are assembled into a single string.

```
Dim sErrDesc As String
Dim erCur As Error
Dim cn As New ADODB.Connection

' Define a connection to an object that does not exist.
cn.ConnectionString = "Provider=msolap;Datasource=NoSuchServer;"
' This provides two errors in the ADO errors collection.
On Error GoTo found_error

cn.Open

' Because a computer named NoSuchServer does not exist,
' you should never reach this point.
Exit Function
```
found_error:

' Keep Visual Basic error description -- On Error Resume Next clear
sErrDesc = Err.Description & vbCrLf

On Error Resume Next

' Get the ADO errors.
If cn.Errors.Count > 0 Then
  For Each erCur In cn.Errors
    sErrDesc = sErrDesc & erCur.Source & ": " & erCur.Description & vbCrLf
  Next erCur
End If
MsgBox sErrDesc
Analysis Services Programming
Connecting to a Data Source

The primary way to interact with PivotTable® Service is to connect to a data source using a **Connection** object or the **ActiveConnection** property of a **Catalog** object. Parameters for this connection can be set using a connection string. For example, the properties in the connection string of a **Connection** object determine whether a connection connects to an Analysis server, creates a new cube, or connects to an existing local cube file. For more information about these data source properties, see [Connection String Properties](#).

After a connection to an OLE DB for OLAP provider or a local cube has been established, queries can be issued against the data source and the results displayed. Information about the **schema** of the data source can also be retrieved. For more information about data sources in OLE DB, see the OLE DB documentation.
Using the Connection String

The primary way to interact with PivotTable® Service is to connect to a data source using a Connection object or the ActiveConnection property of a Catalog object. Parameters for this connection can be set using a connection string. For example, the properties in the connection string of a Connection object determine whether a connection connects to an Analysis server, creates a new cube, or connects to an existing local cube file. For more information about these data source properties, see Connection String Properties.

After a connection to an OLE DB for OLAP provider or a local cube has been established, queries can be issued against the data source and the results displayed. Information about the schema (that is, structure) of the data source can also be retrieved. For more information about data sources in OLE DB, see the OLE DB documentation.

The ADO Connection Object

The Open method of the Connection object provides for the inclusion of connection parameters in its ConnectionString property. A semicolon delineates each parameter. When this method is executed, a connection to the data source defined in the connection string is created.

The syntax of the open method is:

connection.Open ConnectionString, UserID, Password, OpenOptions

Connecting to Analysis Services

To connect to Microsoft® SQL Server™ 2000 Analysis Services, the Datasource property must be set to the name or IP address of the Analysis server to which you want to connect. The Provider property must also be set to "MSOLAP". Optionally, the Initial Catalog property may be set to specify a connection to a specific database on the Analysis server.

Connecting to a Local Cube
Connecting to a local cube is identical to connecting to an Analysis server with one exception: The **Datasource** property is set to the file location for the local cube instead of being set to the name of an Analysis server.

**Setting the Connect Timeout Property**

In OLE DB, the connection property that defines when a connection times out is DBPROP_INIT_TIMEOUT. In the connection string, this property is referred to as **Connect Timeout**. If a connection to a data source cannot be established in the number of seconds specified by this property, an error occurs.

The following example connects to an Analysis server on the local computer and sets the connect timeout property to 5 seconds:

```
Dim MyCon as ADODB.Connection
Set MyCon = new ADODB.Connection
MyCon.Open("provider=msolap; Datasource=LocalHost; Initial Catalog=")
```

Using the OLE DB Connection Dialog Box

OLE DB specifies that each provider must provide a dialog box for defining connections to its data sources. Microsoft® SQL Server™ 2000 Analysis Services complies with this requirement by providing a dialog box that enables the client application to connect to an Analysis server or a local cube.

To use this prompt, the **Prompt** property of the connection string must be set to 1:

```
Dim Conn As New ADODB.Connection
Conn.Open "Provider=msolap; Prompt=1;"
```
Connecting Using HTTP

This feature enables a client application to connect to an Analysis server through Microsoft® Internet Information Services (IIS) by specifying a URL in the **Data Source** property in the client application's connection string. This connection method allows PivotTable® Service to tunnel through firewalls or proxy servers to the Analysis server. A special Active Server Pages (ASP) page, Msolap.asp, enables the connection through IIS. The directory in which this file resides must be included as part of the URL when connecting to the server (for example, http://www.myserver.com/myolap/).

The rest of the connection string is specified normally.

The port used during connection is defined by the default port of the Web site. In most cases, the default port of the web site is set to 80 (or 443 if Secure Sockets Layer (SSL) is used).

For more information about the **ConnectionString** property, see the Microsoft ActiveX® Data Objects (ADO) documentation.

Examples

**A. Using a URL as the Data Source**

In this example, the **Datasource** property is set to the URL of a computer running IIS. From this point forward, the connection is seamless from the point of view of the client application. The following code shows how to connect to the default Analysis server:

```vba
Dim cat as new ADOMD.Catalog
cat.ActiveConnection = "Provider = msolap;" & _
  "Datasource =" _ &
  "http://<URL>/;" & _
  "Initial Catalog = FoodMart 2000"
```
**B. Using SSL**

In this example, the parameter for the server name is passed as a part of the URL, and a Secure Sockets Layer (SSL) connection is specified. The following code shows how to connect to a specified Analysis server:

```vba
Dim cat as new ADOMD.Catalog
cat.ActiveConnection = "Provider = msolap; Datasource =" & _
    " https://<URL>/;" & _
    " Initial Catalog = FoodMart 2000"
```
Analysis Services Programming
Retrieving Schema Information

You can use Microsoft® ActiveX® Data Objects (Multidimensional) (ADO MD), ADO, or OLE DB to retrieve schema rowsets using PivotTable® Service.

To retrieve schema information for a cube, use the CubeDef object in ADO MD or the OpenSchema method in ADO. The CubeDef object contains a hierarchy of collections describing a cube's structure. A CubeDef object for a particular cube can be obtained from the Catalog object's CubeDef property. Its collections can then be iterated through to retrieve the desired schema information. Some information about the cube is not contained in the CubeDef object, such as defined actions and cell formulas. You must use the OpenSchema method to retrieve this information.

To retrieve schema rowsets, use ADO or OLE DB. In ADO, use the OpenSchema method of the Connection object to retrieve schema information into an ADO Rowset object. These results can then be browsed using usual methods. In OLE DB, use the IDBSchemaRowset COM interface to retrieve schema information.

See Also

Schema Rowsets
Using the CubeDef Object
Using the OpenSchema Method
Using the CubeDef Object

To retrieve cube schema information, use the Microsoft® ActiveX® Data Objects (Multidimensional) (ADO MD) CubeDef object, which exposes the dimensions of the local cube using its Dimensions collection. The Dimensions collection exposes the individual Dimensions, which in turn expose the Hierarchies collection, and so on.

For more information about using the ADO MD CubeDef object to retrieve schema rowsets, see the ADO documentation.

The CubeDef Object Model

The following diagram illustrates the object model used by ADO MD.

Examples

**Using ADO MD to Print Member Properties**

The following code uses ADO MD to print member properties. This code uses the local cube created by the sample code in Building Local Cubes. This code prints the name and properties of every member of the [Product].[Product Name] level in the cube to the immediate window.

Private Sub Form_Load()
Dim cn As ADODB.Connection
Dim ct As ADOMD.Catalog
Dim cb As ADOMD.CubeDef
Dim dm As ADOMD.Dimension
Dim hr As ADOMD.Hierarchy
Dim lv As ADOMD.Level
Dim mb As ADOMD.Member
Dim pr As ADODB.Property
Set cn = New ADODB.Connection
   cn.Open "provider=msolap;data source=c:\warecube.cub"

Set ct = New ADOMD.Catalog
   Set ct.ActiveConnection = cn

Set cb = ct.CubeDefs(0)
Set dm = cb.Dimensions("Product")
Set hr = dm.Hierarchies(0)
Set lv = hr.Levels("Product Name")

For Each mb In lv.Members
   Debug.Print mb.Name
   Debug.Print "--------------"
   For Each pr In mb.Properties
      Debug.Print pr.Name & ": " & pr.Value
      Next pr
   Debug.Print
Next mb
End Sub
Using the OpenSchema Method

In addition to the **CubeDef** object, Microsoft® ActiveX® Data Objects (ADO) provides the **OpenSchema** method for the connection object. To use this method to get schema information about multidimensional and data mining meta data, use the following query types:

- AdSchemaCatalogs
- AdSchemaCubes
- adSchemaDimensions
- adSchemaHierarchies
- adSchemaLevels
- adSchemaMeasures
- adSchemaMembers
- adProviderSpecific

Using Restriction Columns

Restriction columns enable the returned recordset of an **OpenSchema** function call to be filtered by certain constraints. For any given schema rowset, a number of restrictions may be supported. For example, the MINING_MODELS schema rowset supports the following restriction columns:

- **MODEL_CATALOG**
To use a particular column (or set of columns), build an array of strings that corresponds to the list of restriction column in their order. For instance, to retrieve a list of all of the mining models in the **FoodMart 2000** database that use the Microsoft Decision Trees algorithm, construct the following array in Microsoft Visual Basic®:

Array("FoodMart 2000", Empty, Empty, Empty, Empty, "0")

Each element in the array corresponds to an element in the restriction columns list. The first, "FoodMart 2000," specifies that all of the records returned should be members of the **FoodMart 2000** database (that is, catalog). This is because the MODEL_CATALOG is the first element in the restriction columns. The next four elements are built as empty and specify that no restrictions should be placed on the returned records based upon their respective restriction columns. The last element of the array, "0", is in the position reserved for the SERVICE_TYPE_ID restriction column. The value "0" is determined by looking up the allowed list of values for this restrictions column in the OLE DB for Data Mining specification.

Use this array in the ADO **OpenSchema** method as the **Criteria** parameter.

**Retrieving Rowsets Unsupported by ADO**

To use a schema rowset that is not supported by the ADO **SchemaEnum** enumeration in the ADO **OpenSchema** method, use the enumeration value **adSchemaProviderSpecific** with any restriction columns that are appropriate to
the schema rowset. The **SchemaID** parameter of the **OpenSchema** method will contain the schema's GUID in a string format. For more information, see [Schema Rowsets](#).

**Examples**

A. **Retrieving a List of Cubes**

The following code shows how to use ADO to retrieve a list of cubes in the current database. For more information about the **OpenSchema** method, see the ADO documentation.

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim szCubeName As String
Const CubeNamePosition = 2

Set cn = New ADODB.Connection
cn.Open "provider=msolap; Data Source=LocalHost; Initial Catalog=FoodMart 2000;"

Set rs = cn.OpenSchema(adSchemaCubes, Array("FoodMart 2000", Empty, Empty))

Do Until rs.EOF
    szCubeName = rs.Fields(CubeNamePosition).Value
    Debug.Print szCubeName
    rs.MoveNext
Loop
```

B. **Retrieving a List of Mining Models**

The following example retrieves all of the mining models that exist in the **FoodMart 2000** database:

```
Const DMSCHEMA_MINING_MODELS = "{3add8a77-d8b9-11d2-8d2a-003029144fde}"

' Open the MINING_SERVICES schema rowset. Assume the existence 'connection (cn) and an ADO recordset.
```
Set rst = cn.OpenSchema(adSchemaProviderSpecific, Array("FoodMar..."))
Analysis Services Programming
Retrieving Data

There are two methods for retrieving data with Microsoft® Visual Basic using PivotTable® Service: you can use the Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD) Cellset object or the ADO DB Command and Recordset objects. The ADO MD Cellset and Axes objects are used to retrieve the results of a Multidimensional Expressions (MDX) query. Using the ADO DB Command and Recordset objects with an SQL or MDX statement retrieves the data into a flattened rowset.

You can also use OLE DB to retrieve data from a cube or data mining model. The primary interfaces for this are IMDDataSet and IRowset interfaces. For more information, see the OLE DB documentation.

See Also

Using the Cellset Object
Using the Recordset Object
Using the Cellset Object

You can use Microsoft® ActiveX® Data Objects (Multidimensional) (ADO MD) to retrieve Multidimensional Expressions (MDX) query results from a local cube using the Cellset object. To retrieve a tabular result set, use the ADO Command and Recordset objects. For more information about the Cellset object, see the ADO MD documentation.

Examples

A. Using the Cellset Object

The following example uses a Connection object to define a connection to the Analysis server. The Source property of the Cellset object is then set to an MDX query that returns all of the measures for product families and promotion media. The ActiveConnection property of the Cellset object is then set to the ActiveConnection property of the Connection object, and the Open method is called to retrieve the actual results.

The Cellset object contains a collection called Axes, which describes each axis returned by the MDX query. There is one Axis object in this collection for each dimension you request. Each Axis object contains a Positions collection, which contains information about the individual rows, columns, pages, and so on of the returned result set. In this example, a Microsoft FlexGrid control is formatted to display the results of the query:

```vba
Dim conn As New ADODB.Connection
Dim cst As New ADOMD.Cellset
Dim axs As ADOMD.Axis
Dim pos As ADOMD.Position
Dim iCol As Integer, cCol As Integer
Dim iRow As Integer, cRow As Integer
Dim nFixedCols As Integer, nFixedRows As Integer
```
'Set up the connection to the server.
conn.ConnectionString = "Datasource=LocalHost; Provider=msolap"
conn.Open
Set cst.ActiveConnection = conn ' You must use Set.
cst.Source = "Select CrossJoin([Product],[Product Family].Members, 
    "[Promotion Media].Members) on rows," & _
    "[Measures].Members on Columns " & _
    "From Sales"
cst.Open

'Set up the FlexGrid control.
MSFlexGrid1.Clear
nFixedCols = 2
nFixedRows = 1
cCol = cst.Axes(0).Positions.Count
MSFlexGrid1.Cols = cCol + nFixedCols
cRow = cst.Axes(1).Positions.Count
MSFlexGrid1.Rows = cRow + nFixedRows
MSFlexGrid1.FixedCols = nFixedCols
MSFlexGrid1.FixedRows = nFixedRows
    MSFlexGrid1.MergeCol(0) = True
MSFlexGrid1.MergeCol(1) = True

'Add column headers.
iCol = 2
For Each pos In cst.Axes(0).Positions
    'The caption for each member is used as the header.
    MSFlexGrid1.TextMatrix(0, iCol) = pos.Members(0).Caption
    iCol = iCol + 1
Next

'Add row headers.
iRow = 1
For Each pos In cst.Axes(1).Positions
  ' The CrossJoin function in MDX indicates that this axis will have
  MSFlexGrid1.TextMatrix(iRow, 0) = pos.Members(0).Caption
  MSFlexGrid1.TextMatrix(iRow, 1) = pos.Members(1).Caption
  iRow = iRow + 1
Next

' Iterate through the cellset array values.
For iCol = 0 To cCol - 1
  For iRow = 0 To cRow - 1
    ' Retrieve each value with the default method of the cst object.
    MSFlexGrid1.TextMatrix(iRow + nFixedRows, iCol + nFixedCols)
  Next
Next

For more information, see the ADO MD documentation.
Using the Recordset Object

You can use the **Command** and **Recordset** objects to retrieve data from a multidimensional source as well as from a tabular source. However, the data is returned in a flattened rowset. For more information about flattened rowsets, see [Working with OLAP Data](#).

Examples

**A. Using the Recordset Object**

The following example retrieves a list of products and their sales from the Sales cube in the **FoodMart 2000** database and places them in a Microsoft® FlexGrid control. The control is filled with the values retrieved by an SQL query using the Microsoft ActiveX® Data Objects (ADO) **Connection** and **Recordset** objects.

A connection to the **FoodMart 2000** database is declared, using MSOLAP as a provider. The connection is then opened. Next, an SQL query is created inside a string variable. A **Recordset** object is then opened with this SQL query as its source.

A loop is then entered, which fills the FlexGrid control with the retrieved values until the end of the recordset is reached.

For more information about using the ADO **Command** and **Recordset** objects, see the ADO documentation. For more information about the FlexGrid control, see the Microsoft Visual Basic® documentation.

```vbnet
Dim cn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim sql As String
Dim fld As ADODB.Field

cn.ConnectionString = "Provider=MSOLAP; Datasource=LocalHost; Initial Catalog=FoodMart 2000"
cn.Open
```
sql = "Select [Product:Product Name],[Measures:Store Sales] from \\
set rs.ActiveConnection = cn
rs.Open sql, cn, adOpenForwardOnly, adLockReadonly
MSFlexGrid1.Clear
MSFlexGrid1.AddItem "Product" & Chr(9) & "Value"
Do While (Not rs.EOF)
    MSFlexGrid1.AddItem rs.Fields(0).Value & Chr(9) & rs.Fields(1).
    rs.MoveNext
Loop
Analysis Services Programming
Updating Information in a Cube

PivotTable® Service supports a number of methods of updating the contents of both server cubes and local cubes. Transaction support for writeback operations is provided by the transaction methods of the Microsoft® ActiveX Data Objects® (ADO) Connection object.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing a Value Back to a Cell</td>
<td>Describes the different methods of updating the value of a cell in the cube itself, rather than through the cube's fact table</td>
</tr>
<tr>
<td>Transaction Processing</td>
<td>Shows how to use the transaction methods provided by the ADO Connection object</td>
</tr>
</tbody>
</table>

See Also

Transactions in Analysis Services
Analysis Services Programming

Writing a Value Back to a Cell

There are different methods for updating the value of a cell directly in the cube, depending on the level of that value. The first method, called writeback, sets the value of a leaf level member directly, using the transaction methods of the Connection object. The second method, cell allocation, sets the value of a nonleaf member and specifies how the change should be distributed among the children of the member. For this method, it is not necessary to use the transaction methods of the Connection object. You can indirectly update a value in a cube by modifying its fact table and reprocessing it.

Writing to a Local Cube

Because local cubes do not use individual partitions to store data, changes to their data cannot be stored permanently. All changes made to the data of a local cube are stored only for the duration of the session scope. If you need to change the contents of a local cube permanently, you must make changes to the source data, not the local cube, and then rebuild the local cube.

Writing Back to Leaf Cells

For cells that are at the lowest level of the hierarchy, you can use the writeback method. The choice of method for updating the values of aggregate members within a server cube depends on the level depth of the member. For members that are at the lowest level of a hierarchy (and are therefore nonaggregated and atomic), update the cell's value property in a matching set of connection transaction methods.

Writing Back to Nonleaf Cells

For cells that are not at the lowest level of a hierarchy, use the UPDATE CUBE to execute a cell allocation. Using this method, an application can make a change to a nonleaf member and describe how that change should be allocated to the children of that member. It may be helpful to think of the UPDATE CUBE statement as a subroutine that automatically generates a series of individual
writeback operations to atomic cells that roll up into a specified sum.

Examples

A. Updating a Leaf Cell Using Transactions

The following transaction updates a cell in a **Cellset** object, using cell writeback and transactions:

'Assume the existence of an open ADO Connection object (cn) and a Cellset object. Also assume that ix and iy are integers pointing to an updatable cell. txtNewValue is assumed to be a string containing a new value for the cell.

- cn.BeginTrans ' Start a new transaction.
- cs(ix, iy).Value = Val(txtnewValue.Text) ' Write the new value to the cell.
- cn.CommitTrans

B. Allocating a Budget Based on Previous Sales

The following example demonstrates cell allocation by updating the various departments' 1999 budgets based on their 1998 sales:

```
UPDATE CUBE [Budget Cube]
SET
  ([1999], [Marketing], [Budget], [All Departments]) = 1000

USE_WEIGHTED_ALLOCATION BY
  ([1998], [Sales], [Actual])/
  ([1999], [Sales], [Actual], [All Departments])
```

C. Allocating a Budget Based on Percentage Increase

The following example demonstrates cell allocation by updating the various departments' 1999 budgets by specifying that each department will receive a 10% increase for each month over the budget of the previous month:

```
UPDATE CUBE [Budget Cube]
SET
```
([1999], [Sales], [Budget]) = 1000

USE_WEIGHTED_ALLOCATION BY

([Sales], [Budget])/
1 + (Rank([1999].Children, Time.CurrentMember) * 0.1
/
Sum(Rank([1999].Children,
1 + (Rank([1999].Children, Time.CurrentMember) * 0.1)

See Also

Transaction Processing
Transactions in Analysis Services
UPDATE CUBE Statement
Transaction Processing

Transaction support for PivotTable® Service client applications is provided by the Microsoft® ActiveX® Data Objects (ADO) **Connection** object. This object provides three methods for conducting transactions against cubes and data mining models: **BeginTrans**, **CommitTrans**, and **Rollback**. For more information, see [Performing Transactions in ADO](#).

Setting the Writeback Timeout Property

The amount of time that will elapse before a writeback operation times out can be specified in seconds using the **Writeback Timeout** Property. This property can be set by using the connection string of the ADO **Connection** property when a session is established. After setting this value it cannot be changed for the duration of the session. The following code sets the value of this property to 60 seconds:

```vba
dim cn as ADO.Connection
cn.Open "provider=msolap; Initial Catalog=FoodMart 2000; Datasource=
```

See Also

[Writeback Timeout Property](#)
Analysis Services Programming
Building Local Cubes

The process of creating a local cube can be summarized in a few steps.

1. Define dimensions.
2. Define measures.
3. Define calculated members.
4. Define other objects such as levels, member properties, and so on.
5. Populate the dimensions.
6. Populate the measures and calculated members.
7. Map the dimensions and measures into the cube structure.
8. Process the cube by connecting to the source provider.

To accomplish steps 1 through 6, use strings that consist of data definition language (DDL) statements such as CREATE CUBE. Assign each statement to its corresponding connection string and then assemble all the connection string properties into a single connection string. The local cube is then created during the process of connecting to the data source.

Before creating a local cube, you must decide which storage mode to use. You can create local cubes in multidimensional OLAP (MOLAP) or relational OLAP (ROLAP) storage mode. For more information about OLAP storage modes, see Flexible Data Model.

To create a MOLAP cube, use the CREATE CUBE statement in the connection string to define the cube's dimensions, levels, members, and measures. The
INSERT INTO statement is then used to populate the cube with data. The result is a local cube saved on the client computer, which can then be connected to and analyzed offline.

To create a ROLAP cube, use the CREATE CUBE statement in the connection string to define the cube (as described in the previous paragraph). Then use the INSERT INTO statement with the OPTIONS DEFER_DATA clause to populate its dimensions and members. This saves the structural definition of the local cube (that is, the cube and dimension definitions) on the local computer, but does not save the member data. The client application can then connect to the local cube and analyze its data (while connected to its data source) without a connection to an Analysis server.

Local MOLAP cubes generally take longer to create than ROLAP cubes because the cube data must be added to the meta data. These cubes are usually much larger than ROLAP cubes. However, local MOLAP cubes provide better performance during query execution than local ROLAP cubes do.

PivotTable® Service can only be used to define local cubes. You cannot create cubes on an Analysis server using PivotTable Service.

The source data used to create a local cube must be to a tabular data provider, such as a relational database, or from an Analysis server, which can act as a tabular data provider. The name of the local cube file to be created is defined using the Datasource property in the connection string. The file extension of the cube file is .cub. You cannot specify other extensions for files of this type.

**Caution** If the specified cube file already exists on the local computer, PivotTable Service overwrites this existing file with the new local cube unless you set the UseExistingFile property of the Microsoft® ActiveX® Data Objects (ADO) Connection object to a value that begins with Y (for YES), T (for TRUE), or a nonzero numerical value.

If the name of a cube is different from the name of a cube already in a cube file, this new cube is appended to the old one.

**See Also**

[Building and Processing Cubes](#)
CREATE CUBE Statement

Data Source Property

UseExistingFile Property

Flexible Data Model

INSERT INTO Statement
Using the CREATE CUBE Statement

The following topic describes the first two steps in creating a local cube: defining the local cube's dimensions, and defining the local cube's measures.

Defining Dimensions

The first step in creating a local cube is to define its dimensions and levels.

The example code contained at the end of this topic creates a local cube called C:\Warecube.cub from the sample FoodMart 2000 database (FoodMart 2000.mdb), which is provided with Microsoft® SQL Server™ 2000 Analysis Services. The cube has the following structure:

Store Dimension

<table>
<thead>
<tr>
<th>Level</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Stores</td>
<td>ALL</td>
</tr>
<tr>
<td>Store Country</td>
<td>Default</td>
</tr>
<tr>
<td>Store State</td>
<td>Default</td>
</tr>
<tr>
<td>Store City</td>
<td>Default</td>
</tr>
<tr>
<td>Store Name</td>
<td>Default</td>
</tr>
</tbody>
</table>

Store Type Dimension

<table>
<thead>
<tr>
<th>Level</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Store Types</td>
<td>ALL</td>
</tr>
<tr>
<td>Store Type</td>
<td>Default</td>
</tr>
</tbody>
</table>

Time Dimension

Column Hierarchy

<table>
<thead>
<tr>
<th>Level</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>YEAR</td>
</tr>
</tbody>
</table>
Defining Measures

The next step in building a local cube is to define the measures that will be used by that cube. The following table describes the measures used in the example at the end of this topic.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Function</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store Invoice</td>
<td>Sum</td>
<td>#.#</td>
</tr>
<tr>
<td>Supply Time</td>
<td>Sum</td>
<td>#.#</td>
</tr>
<tr>
<td>Warehouse Cost</td>
<td>Sum</td>
<td>#.#</td>
</tr>
<tr>
<td>Warehouse Sales</td>
<td>Sum</td>
<td>#.#</td>
</tr>
<tr>
<td>Units Shipped</td>
<td>Sum</td>
<td>#.##</td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>Units Ordered</td>
<td>Sum</td>
<td>#.##</td>
</tr>
</tbody>
</table>

After the dimensions and measures are defined, they must be populated. For more information about populating a cube's dimensions and measures, see Using the INSERT INTO Statement.

**Examples**

**A. Defining a Local Cube's Dimensions**

Use the following code to define the dimensions of a local cube:

```vbscript
dim cnCube as adodb.connection
Dim s As String
Dim strProvider As String
Dim strDataSource As String
Dim strSourceDSN As String
Dim strSourceDSNSuffix As String
Dim strcreateCube As String
Dim strInsertInto As String

on error goto Error_cmdCreateCubeFromDatabase

'*-----------------------------------------------------
'* Add the provider that will process the connection string.
'*-----------------------------------------------------

strProvider = "PROVIDER=MSOLAP"
```

'*-----------------------------------------------------
'* Add the data source and the name of the cube file (.cub)
'* that will be created.
'*-----------------------------------------------------
strDataSource = "DATA SOURCE=c:\warecube.cub"

\*-----------------------------------------------------
\* Add the source DSN, the connection string for where the data comes
\* Quote the value so it is parsed as one value.
\* This can be either an ODBC connection string or
\* an OLE DB connection string
\* (as returned by the Data Source Locator component).
\*
\* strSourceDSN = "SOURCE_DSN=""""DRIVER=Microsoft Access	
\* 
\* strSourceDSN = "SOURCE_DSN=FoodMart 2000"
\*
\*-----------------------------------------------------
\*-----------------------------------------------------
\* There may be some other parameters that you want applied
\* at run time but not stored in the cube file
\* or returned in the output string.
\* Example:
\* strSourceDSNSuffix = "UID=;PWD="
\*-----------------------------------------------------
\*-----------------------------------------------------
\* Add CREATE CUBE. This defines the structure of the cube,
\* but not the data in it.
\* The BNF for this statement is in the
\* Analysis Services documentation.
\* Note: The names are quoted with square brackets.
\*-----------------------------------------------------

strCreateCube = "CREATECUBE=CREATE CUBE Mycube( "
strCreateCube = strCreateCube & "DIMENSION [Product],"
strCreateCube = strCreateCube & "LEVEL [All Products] TYPE A,
strCreateCube = strCreateCube & "LEVEL [Product Family] ,"
strCreateCube = strCreateCube & "LEVEL [Product Department]"
strCreateCube = strCreateCube & "LEVEL [Product Category] ,"
strCreateCube = strCreateCube & "LEVEL [Product Subcategory]",
strCreateCube = strCreateCube & "LEVEL [Brand Name] ,"
strCreateCube = strCreateCube & "LEVEL [Product Name] ,"
strCreateCube = strCreateCube & "DIMENSION [Store],"
strCreateCube = strCreateCube & "LEVEL [All Stores] TYPE A,
strCreateCube = strCreateCube & "LEVEL [Store Country] ,"
strCreateCube = strCreateCube & "LEVEL [Store State] ,"
strCreateCube = strCreateCube & "LEVEL [Store City] ,"
strCreateCube = strCreateCube & "LEVEL [Store Name] ,"
strCreateCube = strCreateCube & "DIMENSION [Store Type],"
strCreateCube = strCreateCube & "LEVEL [All Store Type] TYPE A,
strCreateCube = strCreateCube & "LEVEL [Store Type]",
strCreateCube = strCreateCube & "DIMENSION [Time] TYPE TIME,
strCreateCube = strCreateCube & "HIERARCHY [Column],"
strCreateCube = strCreateCube & "LEVEL [All Time] TYPE ALL,
strCreateCube = strCreateCube & "LEVEL [Year] TYPE YEAR,
strCreateCube = strCreateCube & "LEVEL [Quarter] TYPE QUARTER,
strCreateCube = strCreateCube & "LEVEL [Month] TYPE MONTH,
strCreateCube = strCreateCube & "LEVEL [Week] TYPE WEEK,
strCreateCube = strCreateCube & "LEVEL [Day] TYPE DAY,
strCreateCube = strCreateCube & "HIERARCHY [Formula],"
strCreateCube = strCreateCube & "LEVEL [All Formula Time]",
strCreateCube = strCreateCube & "LEVEL [Year] TYPE YEAR,
strCreateCube = strCreateCube & "LEVEL [Quarter] TYPE QUARTER,
strCreateCube = strCreateCube & "LEVEL [Month] TYPE MONTH,
strCreateCube = strCreateCube & "DIMENSION [Warehouse],"
strCreateCube = strCreateCube & "LEVEL [All Warehouses] TYPE ALL,
strCreateCube = strCreateCube & "LEVEL [Country] ,"
B. Defining a Local Cube's Measures

In this example, each measure is named and assigned an aggregate function (an expression for a calculated measure) and a format for display.

strCreateCube = strCreateCube & "MEASURE [Store Invoice] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Supply Time] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Warehouse Cost] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Warehouse Sales] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Units Shipped] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#',"
strCreateCube = strCreateCube & "MEASURE [Units Ordered] "
strCreateCube = strCreateCube & "Function Sum "
strCreateCube = strCreateCube & "Format '#.#')"
Using the INSERT INTO Statement

This topic describes the next three steps necessary to build a local cube: populate the local cube's dimensions, populate the local cube's measures, and map the source data for the dimensions and measures onto the local cube's structure.

Populate the Dimensions

Populating a local cube with dimension members and measure data is accomplished using the INSERT INTO statement in the connection string, which follows the CREATE CUBE statement.

**CAUTION** If the name of the cube file to be created does not exist (as defined by the [DBPROP_INIT_DATASOURCE](https://docs.microsoft.com/en-us/dotnet/api/system.data.odbc.dbprop.initdatasource) property), it is created during the processing of this statement. If a cube file with that name already exists, it is overwritten with the new cube structure and data. If the name is not specified, a temporary name is assigned.

Use the portion of the INSERT INTO statement before the SELECT clause to identify the elements of the cube that will be populated from the data source. For more information, see [INSERT INTO Statement](https://docs.microsoft.com/en-us/dotnet/api/system.data.odbc.odbcconnection.insertinto).

Before using the INSERT INTO statement, the cube structure must be defined with a CREATE CUBE statement. For more information, see [Building Local Cubes](https://docs.microsoft.com/en-us/dotnet/api/system.data.odbc.odbcconnection.buildlocalcubes).

Populate the Measures

The measures of a cube are populated in the same way the dimensions are populated.

Map the Dimensions and Measures into the Cube Structure

Use a SELECT clause within the INSERT INTO statement to populate the dimension and level structures. This clause identifies the source tables and columns from the fact table.
If you are creating a local ROLAP cube:

- Precede the SELECT clause with an OPTIONS DEFER_DATA clause.
  (If the OPTIONS DEFER_DATA clause is omitted, a local MOLAP cube is created.)

- Remove the AS Coln clauses.

  **Note** The order of columns in the SELECT clause must match the order of cube elements in the preceding INSERT INTO clause. So, the first column in the SELECT clause populates the first cube element in the INSERT INTO clause, the second populates the second, and so on.

**Examples**

**A. Populating the Dimensions**

This example populates the elements of the cube defined in *Building Local Cubes*:

```
'--------------
'*  Note: In some circumstances the SELECT clause may be passed through
to the relational database: For example, a stored procedure
could be passed in.
'*  Note: Columns in the SELECT can be in any order. Just
  adjust the order of the list of level/measure names to
  match the order of columns in the SELECT clause.
'--------------
strInsertInto = strInsertInto & "INSERT INTO=INSERT INTO Mycube (";
strInsertInto = strInsertInto & "Product.[Product Category], Product.[Product Department],
strInsertInto = strInsertInto & "Product.[Product Category], Product.[Product Subcategory],
strInsertInto = strInsertInto & "Product.[Brand Name], Product.[Product Name],
strInsertInto = strInsertInto & "Store.[Store Country], Store.[Store State],
strInsertInto = strInsertInto & "Store.[Store Name], Store.[Store Type], [Store Type].[Store Type],
strInsertInto = strInsertInto & ")[Time].Formula.Year, [Time].Formula.Quarter, [Time].Formula.Month.[Key],
```
B. Populating the Measures

The following code shows how to populate the measures with data from the fact table:

```sql
strInsertInto = strInsertInto & "inventory_fact_1997.store_invoice,";
strInsertInto = strInsertInto & "inventory_fact_1997.supply_time,";
strInsertInto = strInsertInto & "inventory_fact_1997.warehouse_cost,";
strInsertInto = strInsertInto & "inventory_fact_1997.warehouse_sales,";
strInsertInto = strInsertInto & "inventory_fact_1997.units_shipped,";
strInsertInto = strInsertInto & "inventory_fact_1997.units_ordered ";
strInsertInto = strInsertInto & "From [inventory_fact_1997], [product],
strInsertInto = strInsertInto & "Where [inventory_fact_1997].[product_id] = [product].[product_id]
strInsertInto = strInsertInto & "[product].[product_class_id] = [product_class].[product_class_id]
strInsertInto = strInsertInto & "[inventory_fact_1997].[time_id] = [time_by_day].[time_id]
strInsertInto = strInsertInto & "[inventory_fact_1997].[store_id] = [store].[store_id]
strInsertInto = strInsertInto & "[inventory_fact_1997].[warehouse_id] = [warehouse].[warehouse_id]
```

C. Mapping the Dimensions and Measures onto the Local Cube

The following code shows how to populate dimensions and levels. It includes the SELECT clause.

```sql
'*-----------------------------------------------------
'* Add some options to the INSERT INTO if you need to.
'* These can control if the SELECT clause is analyzed
'* or just passed through,
'* and if the storage mode is MOLAP or ROLAP (DEFER_DATA).
'* Examples:
'* strInsertInto = strInsertInto & " OPTIONS DEFER_DATA"
'* strInsertInto = strInsertInto & " OPTIONS ATTEMPT_ANALYSIS"
'*-----------------------------------------------------
```
'/*-----------------------------------------------------
'/* Add the SELECT clause of the INSERT INTO statement.
'/* Note: SELECT is concatenated onto the end of
'/* the INSERT INTO statement.
'/* Analysis Services passes this
'/* through to the source database if unable to parse it.
'/* Note: For Analysis Services to analyze the SELECT clause,
'/* each column must be qualified with the table name.
'/*-----------------------------------------------------

strInsertInto = strInsertInto & "SELECT product_class.product_family"
strInsertInto = strInsertInto & "product_class.product_department AS Col1,"
strInsertInto = strInsertInto & "product_class.product_category AS Col2,"
strInsertInto = strInsertInto & "product_class.product_subcategory AS Col3,"
strInsertInto = strInsertInto & "product.brand_name AS Col4,"
strInsertInto = strInsertInto & "product.product_name AS Col5,"
strInsertInto = strInsertInto & "store.store_country AS Col6,"
strInsertInto = strInsertInto & "store.store_state AS Col7,"
strInsertInto = strInsertInto & "store.store_city AS Col8,"
strInsertInto = strInsertInto & "store.store_name AS Col9,"
strInsertInto = strInsertInto & "store.store_type AS Col10,"
strInsertInto = strInsertInto & "time_by_day.the_date AS Col11,"
strInsertInto = strInsertInto & "time_by_day.the_year AS Col12,"
strInsertInto = strInsertInto & "time_by_day.month_of_year AS Col13,"
strInsertInto = strInsertInto & "time_by_day.the_month AS Col14,"
strInsertInto = strInsertInto & "time_by_day.month_of_year AS Col15"
strInsertInto = strInsertInto & "time_by_day.the_month AS Col16,"
strInsertInto = strInsertInto & "warehouse.warehouse_country AS Col17,"
strInsertInto = strInsertInto & "warehouse.warehouse_state_province AS Col18,"
strInsertInto = strInsertInto & "warehouse.warehouse_city AS Col19,"
strInsertInto = strInsertInto & "warehouse.warehouse_name AS Col20,"

"*/
Processing a Local Cube

The final step in creating a local cube includes creating a **Connection** object from Microsoft® ActiveX® Data Objects (ADO) and opening the data source connection with the ADO **Open** method. This causes PivotTable® Service to create the local cube and populate it with data.

The code in this topic includes the **Connection** object and **Open** method. This code also completes the creation of the local cube (.cub) file example provided in previous topics.

**Examples**

**A. Connecting to the Data Source**

The following code shows how to process a local cube:

```vbscript
'*-----------------------------------------------------
'* Set a new ADO DB Connection object.
'* Create the cube by passing concatenated connection
'* string to Open method of the connection object.
'*-----------------------------------------------------

Set cnCube = New ADODB.Connection
s = strProvider & ";" & strDataSource & ";" & strSourceDSN & ";" &

Screen.MousePointer = vbHourglass
cnCube.Open s
Screen.MousePointer = vbDefault
Exit Sub

Error_cmdCreateCubeFromDatabase:
    Screen.MousePointer = vbDefault
```

On Error Resume Next

' Get the ADO errors.
Dim erCur as Error
Dim sErrDesc as String
If cnCube.Errors.Count > 0 Then
    For Each erCur In cnCube.Errors
        sErrDesc = sErrDesc & erCur.Source & ": " & erCur.Description
    Next erCur
End If
MsgBox Err.Description & sErrDesc
Defining Calculated Members

Calculated members are members whose value is dependent on an expression rather than on the value of a cell.

You can define a calculated member using one of the following scopes:

Query scope

The calculated member can be used only within the query in which it is defined. Use the WITH clause in the SELECT statement.

Session scope

The calculated member can be used only within the session in which it is defined, but can be used by multiple queries. Use the CREATE MEMBER statement.

Custom Rollups

In addition to the standard aggregate functions Sum, Min, Max, and Count and Distinct Count, more sophisticated custom rollup functions can be defined for any given member in the CREATE CUBE statement. One common usage scenario is the inventory problem where inventory levels are not summed along the Time dimension, as they would be for every other dimension. That is, if you have one item for seven days of time, you do not have seven items for the week. By using the LastChild function in Multidimensional Expressions (MDX) to define a custom rollup formula, you can automatically roll up closing balances along time.

Examples

A. Creating a Calculated Member

Use the following code to create a calculated member. You must use single quotes to enclose the expression for the calculated member. The OLE DB specification, however, does not require these quotes.
CREATE CUBE MYWAREHOUSE (DIMENSION . . . ,
COMMAND (CREATE MEMBER [MYWAREHOUSE].[MEASURES].[WAREHOUSEPROFIT]
AS '[MEASURES].[WAREHOUSE SALES] - [MEASURES].[WARE
"
See Also

Creating Calculated Members
CREATE CUBE Statement
Refresh Local Cubes

Use the REFRESH CUBE statement to refresh the data in a local cube. The REFRESH CUBE statement rebuilds the local cube file with the CREATE CUBE and INSERT INTO statements that were originally used to create the local cube. These statements and a reference to the original source database are stored in the local cube file. The original data source must be available for this statement to succeed.

For more information, see Managing the Client Cache.

See Also

CREATE CUBE Statement
INSERT INTO Statement
Analysis Services Programming
PivotTable Service Programmer's Reference

The Programmer's Reference contains reference pages necessary for working with PivotTable® Service. The following table describes the information contained in each set of reference topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PivotTable Service Properties</td>
<td>Description of properties that can be set in the connection</td>
</tr>
<tr>
<td>Data Definition Language</td>
<td>Statements for defining a local cube, creating a local cube, and altering the structure of a server cube</td>
</tr>
<tr>
<td>Data Manipulation Language</td>
<td>Information about data manipulation commands, such as the SELECT and INSERT INTO statements</td>
</tr>
<tr>
<td>Function Reference</td>
<td>Details about available OLAP and data mining functions</td>
</tr>
<tr>
<td>Schema Rowsets</td>
<td>Additions to the schema rowsets defined by the OLAP portion of the OLE DB specification and new schema rowset restriction columns</td>
</tr>
</tbody>
</table>
Analysis Services Programming
**PivotTable Service Properties**

Properties in PivotTable® Service can be referred to by either property name or property ID. When setting or reading the value of a property using Microsoft® ActiveX® Data Objects (ADO), use the property name. When setting or reading the value of a property using OLE DB, use the property ID. The property ID is listed in the individual reference topic for each property.

**IMPORTANT** Some property names contain embedded spaces; others do not. Use the property name exactly as it is listed under the **Property Name** heading in the property detail topic.

The following table lists the property names and property IDs that PivotTable Service supports. For more information, see the OLAP portion of the OLE DB specification.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArtificialData Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>Authenticated User Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>Auto Synch Period Property</strong></td>
<td>Controls the frequency (in milliseconds) of client/server synchronization</td>
</tr>
<tr>
<td><strong>Cache Policy Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>Cache Ratio Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>Client Cache Size Property</strong></td>
<td>Controls the amount of memory used by the client cache</td>
</tr>
<tr>
<td><strong>CompareCaseNotSensitiveStringFlags Property</strong></td>
<td>Adjusts case-insensitive string comparisons for a specified locale</td>
</tr>
<tr>
<td><strong>CompareCaseSensitiveStringFlags Property</strong></td>
<td>Adjusts case-sensitive string comparisons for a specified locale</td>
</tr>
<tr>
<td><strong>Connect Timeout Property</strong></td>
<td>Determines the maximum amount of time the client application will attempt to connect to the server before timing out</td>
</tr>
<tr>
<td><strong>CreateCube Property</strong></td>
<td>The CREATE CUBE statement to</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Data Source Property</strong></td>
<td>The name of the server computer or local cube file</td>
</tr>
<tr>
<td><strong>Datasource Connection Type Property</strong></td>
<td>Describes the type of connection that is currently active</td>
</tr>
<tr>
<td><strong>Default GUID Dialect Property</strong></td>
<td>Controls the precedence in which language dialects are applied when resolving queries</td>
</tr>
<tr>
<td><strong>Default Isolation Mode Property</strong></td>
<td>Controls whether the isolation level is isolated or determined by the cursor type requested by the rowset properties</td>
</tr>
<tr>
<td><strong>Default MDX Visual Mode Property</strong></td>
<td>Determines the default behavior of visual totals</td>
</tr>
<tr>
<td><strong>Distinct Measures By Key Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>Do Not Apply Commands Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>Execution Location Property</strong></td>
<td>Determines the location of query resolution: the client application, server, or a combination</td>
</tr>
<tr>
<td><strong>Initial Catalog Property</strong></td>
<td>The name of the initial database (catalog)</td>
</tr>
<tr>
<td><strong>InsertInto Property</strong></td>
<td>The INSERT INTO statement used to populate a local cube file created with the CREATE CUBE statement</td>
</tr>
<tr>
<td><strong>Large Level Threshold Property</strong></td>
<td>Determines the definition of large level for client/server handling of level members</td>
</tr>
<tr>
<td><strong>Locale Identifier Property</strong></td>
<td>The locale ID of preference for the client application</td>
</tr>
<tr>
<td><strong>Log File Property</strong></td>
<td>Specifies a file name for logging queries</td>
</tr>
<tr>
<td><strong>MDX Calculated Members Mode Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>MDX Compatibility Property</strong></td>
<td>Determines how empty members are treated for ragged and unbalanced</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MDX Object Qualification Property</td>
<td>Describes how object names are qualified in Microsoft SQL Server™ 2000 Analysis Services</td>
</tr>
<tr>
<td>MDX Unique Name Style Property</td>
<td>Determines the technique for generating unique names</td>
</tr>
<tr>
<td>Mining Execution Location Property</td>
<td>Determines the location of query resolution for data mining queries</td>
</tr>
<tr>
<td>Mining Location Property</td>
<td>Determines the directory in which a local data mining model will be created</td>
</tr>
<tr>
<td>Mining Persistence Format Property</td>
<td>Determines how data mining models are saved</td>
</tr>
<tr>
<td>OLE DB for OLAP Version Property</td>
<td>Indicates the version of the OLE DB provider</td>
</tr>
<tr>
<td>Password Property</td>
<td>Specifies the password to use when connecting using HTTP</td>
</tr>
<tr>
<td>Provider Property</td>
<td>A predefined string containing other initialization properties</td>
</tr>
<tr>
<td>Read Only Session Property</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>Roles Property</td>
<td>Specifies a comma-delimited string of the role names by which a client application connects to the server</td>
</tr>
<tr>
<td>Safety Options Property</td>
<td>Determines how security for user-defined functions is handled</td>
</tr>
<tr>
<td>Secured Cell Value Property</td>
<td>Determines the type of return value that results from a reference to a secured cell</td>
</tr>
<tr>
<td>Show Hidden Cubes Property</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>Source_DSN Property</td>
<td>The OLE DB connection string, ODBC connection string, or ODBC data source name (DSN) for the source relational database; used only when creating a local cube file</td>
</tr>
<tr>
<td>Source_DSN_Suffix Property</td>
<td>Used to specify DSN properties for</td>
</tr>
</tbody>
</table>
Creating local cubes that should not be stored as part of the local cubes structure, such as the user ID and password for the local cube's data source

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL Compatibility Property</strong></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><strong>SSPI Property</strong></td>
<td>Determines the security package to use during the session</td>
</tr>
<tr>
<td><strong>UseExistingFile Property</strong></td>
<td>Determines whether a local cube file is overwritten if the connection string contains CREATE CUBE and INSERT INTO statements</td>
</tr>
<tr>
<td><strong>User ID Property</strong></td>
<td>Specifies a valid user name, such as a valid domain logon or local logon</td>
</tr>
<tr>
<td><strong>Writeback Timeout Property</strong></td>
<td>Determines the maximum amount of time the client application will attempt to communicate updates to a writeback table on the server before timing out</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**ArtificialData Property**
Reserved for future use.

**Property Name**
ArtificialData

**Property ID**
DBPROP_MSMD_ARTIFICIALDATA
Authenticated User Property

Reserved for future use.

Property Name
Authenticated User

Property ID
DBPROP_MSMD_AUTHENTICATED_USER
Auto Synch Period Property

This property controls the frequency (in milliseconds) of client/server synchronization.

Property Name
Auto Synch Period

Property ID
DBPROP_MSMD_AUTOSYNCHPERIOD

Remarks
The default is 10,000 milliseconds (10 seconds).

When this property is set to a NULL value or 0 (zero), automatic synchronization is turned off. Synchronization occurs only when you send a query to the server.

Because some client queries are resolved solely from the client cache, too high a value in this property can result in query results that do not reflect recent updates in the data source. However, too low a value can impede performance. The lowest valid nonzero value is 250 milliseconds. If a value between 1 and 249 (inclusive) is specified, a value of 250 milliseconds is used.

You will usually set the value of this property when you establish a session; however, you can change its value during the session if necessary.
Cache Policy Property
Reserved for future use.

Property Name
Cache Policy

Property ID
DBPROP_MSMD_CACHEPOLICY
Cache Ratio Property

Reserved for future use.

Property Name
Cache Ratio

Property ID
DBPROP_MSMD_CACHERATIO
Client Cache Size Property

This property controls the amount of memory used by the client cache.

Property Name
Client Cache Size

Property ID
DBPROP_MSMD_DATA_CACHE_SIZE

Remarks
If this property is set to 0 (zero), the client cache can use unlimited memory. If this property is set to a value between 1 and 99 (inclusive), the client cache can use the specified percentage of total available virtual memory (physical and page file). If this property is set to 100 or more, the client cache can use up to the specified amount of memory, in kilobytes (KB).

This property's value is used when a session is established, and the value can be changed during the session.
CompareCaseNotSensitiveStringFlags Property

This property adjusts case-insensitive string comparisons for a specified locale.

Property Name
CompareCaseNotSensitiveStringFlags

Property ID
DBPROP_MSMD_COMPARECASENOTSENSITIVESTRINGFLAGS

Remarks

Flags specified for this property are used in case-insensitive string comparisons. These flags control string comparisons and sort order. This property controls how comparisons are made in character sets that do not support uppercase and lowercase characters, such as Katakana (for Japanese) and Hindi. The default is the value of the CompareCaseNotSensitiveStringFlags registry entry on the client computer.

The client application can override the registry entry for case-insensitive string comparisons by setting the CompareCaseNotSensitiveStringFlags property in the connection string. PivotTable® Service can have only one value for this property for each process.

The value of this property, as set in the first connection of the process thread, affects all subsequent connections in that process thread.

It is an error for a subsequent connection to set the property to a value different from that established by the first connection. This includes any scenario in which a first client application sets a nondefault value, and a second client application does not set any value, expecting to use the default. It is the responsibility of the client application to manage these settings when there are multiple sessions per process.

This property's value cannot be changed during the session.
Use the following table to determine which flags to use.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORM_IGNORECASE</td>
<td>0x00000001</td>
<td>Case is ignored.</td>
</tr>
<tr>
<td>Not applicable</td>
<td>0x00000002</td>
<td>Binary comparison. Characters are compared based on their underlying value in the character set, not on their order in their particular alphabet.</td>
</tr>
<tr>
<td>NORM_IGNORENONSENSE</td>
<td>0x00000010</td>
<td>Nonspacing characters are ignored.</td>
</tr>
<tr>
<td>NORM_IGNORESYMBOLS</td>
<td>0x00000100</td>
<td>Symbols are ignored.</td>
</tr>
<tr>
<td>NORM_IGNOREKANJIATYPE</td>
<td>0x00001000</td>
<td>No differentiation is made between Hiragana and Katakana characters. Corresponding Hiragana and Katakana characters, when compared, are considered to be equal.</td>
</tr>
<tr>
<td>NORM_IGNOREWIDTH</td>
<td>0x00010000</td>
<td>No differentiation is made between single-byte and double-byte versions of the same character.</td>
</tr>
<tr>
<td>SORT_STRINGSORT</td>
<td>0x00100000</td>
<td>Punctuation is treated the same as symbols.</td>
</tr>
</tbody>
</table>

For more information about comparing strings in OLE DB, search on "CompareString" in the Platform SDK section of the MSDN® Library at the [Microsoft Web site](https://msdn.microsoft.com).
CompareCaseSensitiveStringFlags Property

This property adjusts case-sensitive string comparisons for a specified locale.

Property Name

CompareCaseSensitiveStringFlags

Property ID

DBPROP_MSMD_COMPARECASESENSITIVESTRINGFLAGS

Remarks

Flags specified for this property are used in case-sensitive string comparisons. These flags control string comparisons and sort order. This property controls how comparisons are made in character sets that do not support uppercase and lowercase characters, such as Katakana (for Japanese) and Hindi. The default is the value of the CompareCaseSensitiveStringFlags registry entry on the client computer if this registry entry exists.

The client application can override the registry entry for case-insensitive string comparisons by setting the CompareCaseSensitiveStringFlags property in the connection string. PivotTable® Service can have only one value for this property for each process.

The value of this property, as set in the first connection of the process thread, affects all subsequent connections in that process thread.

It is an error for a subsequent connection to set the property to a value different from that established by the first connection. This includes any scenario in which a first client application sets a nondefault value, and a second client application does not set any value, expecting to use the default. It is the responsibility of the client application to manage these settings when there are multiple sessions per process.

This property's value cannot be changed during the session.
Use the following table to determine which flags to use.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORM_IGNORECASE</td>
<td>0x00000001</td>
<td>Case is ignored.</td>
</tr>
<tr>
<td>Not applicable</td>
<td>0x00000002</td>
<td>Binary comparison. Characters are compared based on their underlying value in the character set, not on their order in their particular alphabet.</td>
</tr>
<tr>
<td>NORM_IGNORENONSPACE</td>
<td>0x00000010</td>
<td>Nonspacing characters are ignored.</td>
</tr>
<tr>
<td>NORM_IGNORESYMBOLS</td>
<td>0x00000100</td>
<td>Symbols are ignored.</td>
</tr>
<tr>
<td>NORM_IGNOREKANATYPE</td>
<td>0x00001000</td>
<td>No differentiation is made between Hiragana and Katakana characters. Corresponding Hiragana and Katakana characters, when compared, are considered to be equal.</td>
</tr>
<tr>
<td>NORM_IGNOREWIDTH</td>
<td>0x00010000</td>
<td>No differentiation is made between single-byte and double-byte versions of the same character.</td>
</tr>
<tr>
<td>SORT_STRINGSORT</td>
<td>0x00100000</td>
<td>Punctuation is treated the same as symbols.</td>
</tr>
</tbody>
</table>

For more information about comparing strings in OLE DB, search on "CompareString" in the Platform SDK section of the MSDN® Library at the Microsoft Web site.
**Connect Timeout Property**

This property determines the amount of time that will elapse before an unsuccessful connection times out and returns an error.

**Property Name**

**Connect Timeout**

**Property ID**

**DBPROP_INIT_TIMEOUT**

**Remarks**

Indicates the amount of time (in seconds) to wait for initialization to complete.

For more information about **DBPROP_INIT_TIMEOUT**, see the OLE DB documentation.
CreateCube Property

This property contains the CREATE CUBE statement that is used in the connection string during creation of a local cube.

Property Name
CreateCube

Property ID
DBPROP_MSMD_CREATECUBE

Remarks
You must use this property with the InsertInto and Source_DSN properties. When these three properties are used, they are always used together.

This property's value is used when a session is established, and it cannot be changed during the session.

Note The name of this property is formatted differently than the name for the CREATE CUBE statement with which it is associated. When used in the connection string, it must be formatted as a single word: CreateCube.

See Also
CREATE_CUBE Statement
InsertInto Property
Source_DSN Property
Data Source Property

This property is used to set the name of the server computer or local cube file.

Property Name
Data Source

Property ID
DBPROP_INIT_DATASOURCE

Remarks

The property's value depends on the intended operation mode:

- To work with the Analysis server, specify the name of the server computer, or LOCALHOST, or LOCAL, an IP address, or an HTTP or HTTPS URL.

- To create or work with a local cube file, specify the name of the cube file. Local cube files must have an extension of .cub.

- To create a temporary local cube file that will be deleted when the session ends, specify a null value.

This property's value is used when a session is established, but the value cannot be changed during the session.
**Datasource Connection Type Property**

This property is a read-only property that describes the type of connection that is currently active.

**Property Name**

*Datasource Connection Type*

**Property ID**

*DBPROP_DATASOURCE_CONNECTION_TYPE*

**Remarks**

This property returns a bitmask that describes the type of connection that is currently active. The following table lists the values this bitmask can contain.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The connection is to an Analysis server.</td>
</tr>
<tr>
<td>2</td>
<td>The connection is to a <a href="#">local cube</a> file.</td>
</tr>
<tr>
<td>4</td>
<td>The connection is to the Internet or an intranet through HTTP.</td>
</tr>
</tbody>
</table>
**Default GUID Dialect Property**

This property controls the precedence of language dialect parsers when the provider attempts to resolve query syntax.

**Property Name**

Default GUID Dialect

**Property ID**

**DBPROP_MSMD_DEFAULT_GUID_DIALECT**

**Remarks**

Because query syntax can be similar for language dialects such as data mining and SQL, the provider may not be able to infer the dialect from the query syntax. The provider may attempt to execute the query a second time in a different dialect if the query fails to execute in the first dialect. For example, if a data mining query fails, the provider may resubmit the query as SQL. If this second attempt also fails, the provider returns an SQL error message instead of a data mining error message.

The **Default GUID Dialect** property establishes the dialect the provider will use first to attempt to execute the query and the dialect of execution errors returned for query failures, even if the provider attempts to execute the query a second time in another dialect. For example, if the **Default GUID Dialect** property is set to MDGUID_DM, the provider first attempts to execute the query as a data mining query. If this attempt fails, the provider resubmits the query as an SQL query. However, because the value of this property is MDGUID_DM, if the SQL query also fails, the data mining error message is returned, not the SQL error message.

This property can be used in situations in which queries are expected to be more prevalent in one dialect than another.

**Note** If the OLE DB parameter **rguidDialect** is set in
**ICommandText::SetCommandText**, that setting takes precedence over the **Default GUID Dialect** property setting and unequivocally specifies the dialect of the query. That is, the precedence algorithm established by the **Default GUID Dialect** property does not apply.

The following table describes possible values for this property.

<table>
<thead>
<tr>
<th>Property value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBGUID_SQL</td>
<td>The SQL parser has precedence (default)</td>
</tr>
<tr>
<td>MDGUID_DM</td>
<td>The data mining dialect parser has precedence</td>
</tr>
<tr>
<td>MDGUID_MDX</td>
<td>The MDX parser has precedence</td>
</tr>
</tbody>
</table>
Default Isolation Mode Property

This property controls the default transaction level isolation mode, which can override the transaction level determined by the rowset cursor type.

Property Name
Default Isolation Mode

Property ID
DBPROP_MSMD_DEFAULT_ISOLATION_MODE

Remarks
If the first character of this string is Y, T, or a numeric digit other than 0 (zero), the isolation level is Serializable (also known as Isolated). Otherwise, the cursor type requested by the rowset determines the isolation level. For more information about isolation levels, see the OLE DB documentation.

Use this property to provide read repeatability.

This property's value is used when a session is established, and the value can be changed during the session.
Default MDX Visual Mode Property

This property determines the default behavior for visual totals.

Property Name
Default MDX Visual Mode

Property ID
DBPROP_MSMD_DEFAULT_MDX_VISUAL_MODE

Remarks
Use the following table to determine the function of this property.

<table>
<thead>
<tr>
<th>Property value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPROPVAL_VISUAL_MODE_DEFAULT</td>
<td>Provider-dependent. In Microsoft SQL Server™ 2000 Analysis Services, equivalent to DBPROPVAL_VISUAL_MODE_VISUAL</td>
</tr>
<tr>
<td>DBPROPVAL_VISUAL_MODE_VISUAL</td>
<td>Visual totals are enabled.</td>
</tr>
<tr>
<td>DBPROPVAL_VISUAL_MODE_ORIGINAL</td>
<td>Visual totals are not enabled.</td>
</tr>
</tbody>
</table>
Distinct Measures By Key Property

Reserved for future use.

Property Name
Distinct Measures By Key

Property ID
DBPROP_MSMD_DISTINCTMEASURESBYKEY
Do Not Apply Commands Property

Reserved for future use.

Property Name
Do Not Apply Commands

Property ID
DBPROP_MSMD_DONOTAPPLYCOMMANDS
Execution Location Property

This property determines the location of query resolution: the client application, server, or a combination of both.

Property Name
Execution Location

Property ID
DBPROP_MSMD_EXECLLOCATION

Remarks
The following values are available.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default. For compatibility with earlier versions, this means the same as Value 1. The meaning of this default value is subject to change in future versions.</td>
</tr>
<tr>
<td>1</td>
<td>PivotTable® Service selects the query execution location (client application or server) that will provide the best performance.</td>
</tr>
<tr>
<td>2</td>
<td>Queries are executed on the client application.</td>
</tr>
<tr>
<td>3</td>
<td>Queries are executed on the server. (Queries that contain session-scoped calculated members, user-defined sets, or user-defined functions are exceptions.)</td>
</tr>
</tbody>
</table>

This property's value is used when a session is established, and the value can be changed during the session.
Initial Catalog Property

This property is used to set the name of the initial database (also known as the catalog) of a data source during connection.

Property Name
Initial Catalog

Property ID
DBPROP_INIT_CATALOG

Remarks
The value of this property is used when a session is established, but the value cannot be changed during the session.
Analysis Services Programming

InsertInto Property

This property contains the INSERT INTO statement that is used in the connection string during creation of a local cube.

Property Name

InsertInto

Property ID

DBPROP_MSMD_INSERTINTO

Remarks

You must use this property with the CreateCube property and the Source_DSN property. When these three properties are used, they are always used together.

This property's value is used when a session is established, but the value cannot be changed during the session.

Note  The name of this property is formatted differently than the name for the INSERT INTO statement with which it is associated. When used in the connection string, it must be formatted as a single word: InsertInto.

See Also

INSERT INTO Statement
CreateCube Property
Source_DSN Property
Large Level Threshold Property

This property determines the point at which a level is too large to be sent to the client application in a single piece.

Property Name
Large Level Threshold

Property ID
DBPROP_MSMD_LARGE_LEVEL_THRESHOLD

Remarks
Use this property to help manage client application memory usage. If the number of members in a level is below the threshold you set in this property, it is sent to the client application in one piece. If the number of members in a level is equal to or more than this threshold, the level is broken into smaller groups, which are sent as needed. Levels that contain a number of members that is less than this property's value are sent to the client application. If a level contains a number of members that is greater than or equal to the value of this property, the level is incrementally sent from the server to the client application.

The default value is set on the server in the Large level defined as box in the Properties dialog box for the server. The minimum value for this property is 10. Setting this property to a value less than the minimum causes the value to be set to the minimum, without error.

This property's value is used when a session is established, and it cannot be changed during the session.

The default value for this property is 1000.

See Also
Environment Tab (Properties Dialog Box)
Locale Identifier Property

This property is used to set the locale ID (LCID) of preference for the client application.

Property Name
Locale Identifier

Property ID
DBPROP_INIT_LCID

Remarks

The client application can modify the LCID by setting the Locale Identifier property. PivotTable® Service can have only one LCID for each Microsoft® Windows® process. The LCID must be installed in Control Panel in Windows. If it is not, the attempt to set the LCID fails. By default, the Locale Identifier property is reported as null.

When multiple connections are opened in the same process, connections subsequent to the first connection must specify the LCID to be the same as that established by the first connection.

It is an error for a subsequent connection to set the LCID to a value that would be different from that established by the first connection. This includes the scenario in which a first client application sets a nondefault value, and a second client application does not set any value, expecting to use the default. It is the client application's responsibility to manage these settings in scenarios that involve multiple sessions per process.

This property's value is used when a session is established, but the value cannot be changed during the session.
Log File Property

This property sets or returns the name of the file used to log Multidimensional Expressions (MDX) queries.

Property Name
Log File

Property ID
DBPROP_MSMD_LOG_FILE

Remarks
Use this property to specify a file name for logging MDX queries. The specified file is opened for exclusive use by the connection; subsequent attempts by other connections to open the same file will fail. The format of the log file is as follows:

Process name : Process ID
Date
Time
Command type, one of the following:
  MDX - MDX query
  SQL - SQL query
  DM - Data mining query
Query text

Each field in the log file is separated by a space character and terminated by a carriage return/line feed character.
MDX Calculated Members Mode Property

Reserved for future use.

Property Name
MDX Calculated Members Mode

Property ID
DBPROP_MSMD_MDX_CALC_MEMBERS_MODE
MDX Compatibility Property

This property determines how missing members in the hierarchy are treated in ragged and unbalanced hierarchies.

Property Name
MDX Compatibility

Property ID
DBPROP_MSMD_MDXCOMPATIBILITY

Remarks
This property determines how placeholder members in a ragged or unbalanced hierarchy are treated. The following table describes possible values for this property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default. For compatibility with earlier versions, this is the same as Value 1. The meaning of this default value is subject to change in future versions.</td>
</tr>
<tr>
<td>1</td>
<td>Default value. Placeholder members are exposed.</td>
</tr>
<tr>
<td>2</td>
<td>Placeholder members are not exposed.</td>
</tr>
</tbody>
</table>

This property overrides the following registry setting:

HKEY_CLASSES_ROOT\CLSID\{a07ccd0c-8148-11d0-87bb-00c04:

See Also
Ragged Hierarchies
MDX Object Qualification Property

This property provides a bitmask that specifies how multidimensional schema object names are qualified in Multidimensional Expressions (MDX) expressions.

Property Name
MDX Object Qualification

Property ID
MDPROP_OBJQUALIFICATION

Remarks
This read-only property determines how a provider qualifies object names.

The following table describes the function of each bit in this property.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDPROPVAL_MOQ_DATASOURCE</td>
<td>Cubes are qualified by data source name.</td>
</tr>
<tr>
<td>MDPROPVAL_MCQ_CATALOG</td>
<td>Cubes are qualified by catalog name.</td>
</tr>
<tr>
<td>MDPROPVAL_MCQ_SCHEMA</td>
<td>Cubes are qualified by schema name.</td>
</tr>
<tr>
<td>MDPROPVAL_MOQ_CUBE_DIM</td>
<td>Dimensions are qualified by cube name.</td>
</tr>
<tr>
<td>MDPROPVAL_MOQ_DIM_HIER</td>
<td>Hierarchies are qualified by dimension name.</td>
</tr>
<tr>
<td>MDPROPVAL_MOQ_DIMHIER_LEVEL</td>
<td>Levels are qualified by dimension name.</td>
</tr>
<tr>
<td>MDPROP_NAMED_LEVELS</td>
<td>This bit is set if named levels are not supported.</td>
</tr>
<tr>
<td>MDPROPVAL_MOQ_DIMHIER_MEMBER</td>
<td>Members are qualified by dimension name.</td>
</tr>
<tr>
<td>MDPROPVAL_MOQ_LEVEL_MEMBER</td>
<td>Members are qualified by level name.</td>
</tr>
<tr>
<td>MDPROPVAL_MOQ_MEMBER_MEMBER</td>
<td>Members are qualified by ancestor name(s).</td>
</tr>
</tbody>
</table>
PivotTable® Service and Microsoft® SQL Server™ 2000 Analysis Services set the following bitmask for this property:

```
MDPROPVAL_MOQ_DIM_HIER
| MDPROPVAL_MOQ_DIMHIER_LEVEL
| MDPROPVAL_MOQ_DIMHIER_MEMBER
| MDPROPVAL_MOQ_LEVEL_MEMBER
| MDPROPVAL_MOQ_MEMBER_MEMBER
```

The client application cannot change the value of this property.
MDX Unique Name Style Property

This property determines which algorithm is used to generate unique names.

Property Name
MDX Unique Name Style

Property ID
DBPROP_MSMD_MDXUNIQUENAMES

Remarks
Because members in Microsoft® SQL Server™ 2000 Analysis Services can change position in a dimension, the method for generating unique names in SQL Server version 7.0 OLAP Services may result in an unstable unique name over time. To accommodate this change, you can use this property to specify the algorithm for generating unique names. Use the following table to specify which algorithm to use.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default. For compatibility with earlier versions, this is the same as Value 2. The meaning of this default value is subject to change in future versions.</td>
</tr>
<tr>
<td>1</td>
<td>Key path algorithm: [dim].&amp;[k1].&amp;[k2]</td>
</tr>
<tr>
<td>2</td>
<td>Compatible with version 7.0, name path algorithm: [dim].[n1].[n2]</td>
</tr>
<tr>
<td>3</td>
<td>Compatible with SQL Server 2000 Analysis Services. The algorithm uses guaranteed unique names, which are stable over time.</td>
</tr>
</tbody>
</table>

This property overrides the following registry setting:

HKEY_CLASSES_ROOT\CLSID\{a07ccd00-8148-11d0-87bb-00c402458754\}
Mining Execution Location Property

This property determines the location of data mining query resolution: the client application, server, or a combination of both.

Property Name
Mining Execution Location

Property ID
DBPROP_MSMD_MINING_EXECUTION_LOCATION

Remarks
The following table describes possible values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default. For compatibility with earlier versions, this means the same as Value 1. The meaning of this default value is subject to change in future versions.</td>
</tr>
<tr>
<td>1</td>
<td>PivotTable® Service selects the query execution location (client application or server) that will provide the best performance.</td>
</tr>
<tr>
<td>2</td>
<td>The server updates the mining model on the client, and then the query is executed and resolved on the client.</td>
</tr>
<tr>
<td>3</td>
<td>Queries are executed and resolved on the server.</td>
</tr>
</tbody>
</table>

This property's value is used when a session is established; its value cannot be changed during the session.
Mining Location Property

This property specifies the directory location for local data mining models that will be used or created during the session.

Property Name
Mining Location

Property ID
DBPROP_MSMD_MINING_LOCATION

Remarks
Queries can reference more than one local data mining model during a session. This property determines the directory location for all local data mining models that will be used during the session. This property also determines where local data mining model files will be stored.

Caution If this property's value is not set during connection, local mining models will be created with connection scope. This means that they will only last for the duration of the session, and they will be lost when the session ends.
Mining Persistence Format Property

This property determines how mining models are stored.

Property Name

Mining Persistence Format

Property ID

DBPROP_MSMD_MINING_PERSISTENCE_FORMAT

Remarks

Data mining models can be stored in two different formats: binary and XML. The value of this property determines which format is used. The following table describes the possible values for this property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default. This value is the same as Value 2.</td>
</tr>
<tr>
<td>1</td>
<td>Models are stored in XML.</td>
</tr>
<tr>
<td>2</td>
<td>Models are stored in binary format.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**OLE DB for OLAP Version Property**

This property indicates the version of OLE DB supported by the provider.

**Property Name**

**OLE DB for OLAP Version**

**Property ID**

**DBPROP_PROVIDEROLEDBOLAPVER**

**Remarks**

The version is of the form ##.##, where the first two digits are the major version and the next two digits are the minor version. For example, OLE DB for OLAP providers conforming to the 2.0 specification would return "02.00".

The OLE DB for OLAP specification was included as part of the version 2.6 OLE DB specification. This property has been retained to ensure backward compatibility with client applications using earlier versions of PivotTable® Service, but it is not recommended for current use. This property contains the same value as **DBPROP_PROVIDEROLEDBVER** for providers that comply with version 2.6 of the OLE DB specification.
Password Property

This property specifies a valid domain logon password.

Property Name
Password

Property ID
DBPROP_AUTH_PASSWORD

Remarks
This property is used only when the application is connecting to Microsoft® SQL Server™ 2000 Analysis Services using HTTP. Its value must be set to a valid domain logon password. You can also use this property with the Source_DSN_Suffix property when connecting to a relational data source to build a local cube.

See Also

User ID Property
Source_DSN_Suffix Property
Analysis Services Programming

Provider Property

This property is used to set a predefined string containing initialization properties specific to the provider.

Property Name

Provider

Property ID

DBPROP_INIT_PROVIDERSTRING

Remarks

This is a standard OLE DB property; however, it does not specify usage. Usage is specific to the providers that use it. For connections to Microsoft® SQL Server™ 2000 Analysis Services, this property should be set to "MSOLAP;".

In PivotTable® Service, this property is used by Microsoft ActiveX® Data Objects (ADO) and ADO (Multidimensional) (ADO MD). The property string is semicolon-delimited. Double quotation marks can be used to delimit individual values.

During initialization (IDBInit::Initialize), every recognized property from the Provider property is copied into its respective property value, overwriting existing property values.

The Provider property is cleared after the initialization process is complete (that is, after a connection to the Analysis server is established or a local cube is created).

It is the responsibility of the client application to store the connection string.

See Also

Connecting to a Data Source
Read Only Session Property
Reserved for future use.

Property Name
Read Only Session

Property ID
DBPROP_MSMD_READ_ONLY_SESSION
Roles Property

This property specifies a comma-delimited string of the role names under which a client application connects to the server.

Property Name
Roles

Property ID
DBPROP_MSMD_ROLES

Remarks
This property allows the user to connect using a role other than the one he or she is currently using. For example, a member of the OLAP Administrators role may want to connect to a cube as a member of the Users role to test permissions granted to the Users role. This user must be a member of the role specified in order to connect using this property.

IMPORTANT  Role names are case-sensitive, and spaces should not be used between the comma-delimited role names. Otherwise errors and unexpected results may be returned as a result of queries to secured cell sets.

See Also
User ID Property
Password Property
Safety Options Property

This property determines how security for user-defined functions and actions is handled.

Property Name
Safety Options

Property ID
DBPROP_MSMD_SAFETY_OPTIONS

Remarks
The value of the property determines whether unsafe libraries can be registered and loaded by PivotTable® Service or the server. If an attempt is made to load an unsafe library or reference an unsafe user-defined function while safety checking is enabled, then PivotTable Service will return the following error:

User defined function <function name> cannot be used, because the class that implements it is not safe for initialization or scripting, which is required by client settings.

The following table lists the possible values for this property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPROPVAL_MSMD_SAFETY_OPTIONS_DEFAULT</td>
<td>For connections to a local cube, or via an IIS server, this value is the same as DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_SAFE for all other connections.</td>
</tr>
<tr>
<td>DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_ALL</td>
<td>This value enables loading all user-defined function libraries without verifying that they are safe for initialization and scripting.</td>
</tr>
<tr>
<td>DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_SAFE</td>
<td>This value ensures that all classes for a particular user-defined function library are safe for initialization and scripting.</td>
</tr>
</tbody>
</table>
Function libraries are checked to ensure that they are safe for initialization and scripting.

| DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_NONE | This value prevents user-defined functions from being used during the session. |
Secured Cell Value Property

This property determines the value returned for secured cells.

Property Name
Secured Cell Value

Property ID
DBPROP_MSMD_SECURED_CELL_VALUE

Remarks

Depending on the value you set for the Secured Cell Value property, queries that attempt to access a secured cell can return a specified error code and content for the Value and Formatted Value properties of the cell. The following table lists possible values you can set for the Secured Cell Value property and the error code and cell property values returned as a result of each setting.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default. For compatibility with earlier versions, this means the same as Value 1. The meaning of this default value is subject to change in future versions.</td>
</tr>
<tr>
<td>1</td>
<td>Returns: HRESULT = NO_ERROR&lt;br&gt;The Value property of the cell contains the result as a variant data type. The string &quot;#N/A&quot; is returned in the Formatted Value property.</td>
</tr>
<tr>
<td>2</td>
<td>An error is returned as the value of HRESULT.</td>
</tr>
<tr>
<td>3</td>
<td>NULL is returned in both the Value and Formatted Value properties.</td>
</tr>
<tr>
<td>4</td>
<td>A numerical zero (0) is returned in the Value property, and a formatted zero is returned in the Formatted Value property. For example, 0.00 is returned in the Formatted Value property for a</td>
</tr>
</tbody>
</table>
The string "#SEC" is returned in both the **Value** and **Formatted Value** properties.

For more information, see [Security in PivotTable Service](#).

**IMPORTANT** This property interacts with the execution location of the query. In some cases, the value of the **Execution Location** property may override the **Secured Cell Value** connection string property. For example, if the **Execution Location** property is set to **Isolated Mode**, a query that involves cells for which the user does not have read permissions will return #N/A even if the **Secured Cell Value** property has been set to 2, 3, 4, or 5. For more information about isolation levels, see [Isolation Levels](#) and the OLE DB documentation.

**See Also**

[Execution Location Property](#)
Show Hidden Cubes Property

Reserved for future use.

Property Name
Show Hidden Cubes

Property ID
DBPROP_MSMD_SHOW_HIDDEN_CUBES
Source_DSN Property

This property contains the OLE DB connection string, ODBC connection string, or ODBC data source name (DSN) for the relational database or cube that will be used as the source for creating a local cube.

Property Name
Source_DSN

Property ID
DBPROP_MSMD_SOURCE_DSN

Remarks
This property is used only when creating a local cube file.

You must use this property with the CreateCube property and the InsertInto property. When these three properties are used, they are always used together.

This property's value is used when a session is established, but the value cannot be changed during the session.
Source_DSN_Suffix Property

This property contains a string that is appended to the Source_DSN property value when creating or connecting to a local cube. This property's value is set when a session is established, and it can be reused but not changed during the session.

Property Name
Source_DSN_Suffix

Property ID
DBPROP_MSMD_SOURCE_DSN_SUFFIX

Remarks
This property is useful for separating data permanently stored in the local cube file from data used only for the session, such as the user account and password, because this property's value is not stored in the local cube file. Its primary purpose is to specify a user ID and password that the relational data source uses to create or access a local cube. If this property is used when you create a local ROLAP cube, it may also be required when you later connect to the cube. For example, if you specify a user account and password in this property when creating a local ROLAP cube, you must specify them in this property for each subsequent connection to the local ROLAP cube file.

See Also
Source_DSN Property
Analysis Services Programming

**SQL Compatibility Property**

Reserved for future use.

**Property Name**

SQL Compatibility

**Property ID**

DBPROP_MSMD_SQLCOMPATIBILITY
SSPI Property

This property determines the security package that will be used during the session.

Property Name
SSPI

Property ID
DBPROP_MSMD_SSPI

Remarks
The value of this property corresponds to the name of a security package. You can use the Security Support Provider Interface (SSPI) \texttt{EnumerateSecurityPackages} function to enumerate the providers that are supported on a given computer.

Microsoft™ SQL Server® 2000 Analysis Services supports the following packages:

- Negotiate
- Kerberos
- NTLM
- Anonymous User

Example

Suppose that the Kerberos security provider is the security package used for authentication in the domain on which a particular client application must run.
The following code opens a connection using this provider and then displays a dialog box to complete the connection:

Dim cn as New ADODB.Connection
cn="Provider=MSOLAP; SSPI=Negotiate; Prompt=1"
cn.open
UseExistingFile Property

This property determines whether an existing local cube file is overwritten when creating a local cube file of the same name.

Property Name
UseExistingFile

Property ID
DBPROP_MSMD_USEEXISTINGFILE

Remarks

This function of this property depends on the condition of the cube file being created. This table shows the interaction between the condition of the cube file and the value to which this property is set.

<table>
<thead>
<tr>
<th>Condition</th>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>File and cube do not exist</td>
<td>Returns E_FAIL when you open a new data source.</td>
<td>A new file is created when the data source is opened.</td>
</tr>
<tr>
<td>File exists, cube does not</td>
<td>The cube is created in the file.</td>
<td>The cube is created in the file.</td>
</tr>
<tr>
<td>File and cube exist</td>
<td>The existing cube is used.</td>
<td>The existing cube is overwritten.</td>
</tr>
</tbody>
</table>

This property's value is used when a session is established, but the value cannot be changed during the session.
Analysis Services Programming

**User ID Property**

This property specifies a valid user name that will be recognized by the user's computer, such as a valid domain logon or local logon.

**Property Name**

User ID

**Property ID**

DBPROP_AUTH_USERID

**Remarks**

This property is used only when the client application is connecting to Microsoft® SQL Server™ 2000 Analysis Services using HTTP. Its value must be set to a valid domain logon. You can also use this property with the Source_DSN_Suffix property when connecting to a relational data source to build a local cube.

**See Also**

Password Property

Source_DSN_Suffix Property
**Writeback Timeout Property**

This property determines the maximum amount of time (in seconds) the client application will attempt to communicate updates to a writeback table on the server.

**Property Name**
Writeback Timeout

**Property ID**
DBPROP_MSMD_WRITEBACK_TIMEOUT

**Remarks**

When a client application attempts to commit writeback changes to the server, PivotTable® Service begins a count in seconds. The count continues until the commit is successful or the number of seconds specified in this property is reached. If the count reaches the value of this property, the commit fails and the update does not occur. After a timeout failure, the client can roll back the transaction or attempt to commit the transaction again.

This property's value is specified when a session is established, and it cannot be changed during the session.
Analysis Services Programming
# Data Definition Language

The following topics describe the data definition language (DDL) used by PivotTable® Service.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER CUBE Statement</td>
<td>Allows client applications to control the structure of a cube after it has been created.</td>
</tr>
<tr>
<td>CREATE ACTION Statement</td>
<td>Allows the user to create action definitions that can be associated with a member and executed when that member is referenced.</td>
</tr>
<tr>
<td>CREATE CACHE Statement</td>
<td>Populates the cache with a slice of cube data defined by sets of members.</td>
</tr>
<tr>
<td>CREATE CELL CALCULATION Statement</td>
<td>Creates a calculated cell formula for specified tuples within a cube.</td>
</tr>
<tr>
<td>CREATE CUBE Statement</td>
<td>Creates a local cube or virtual cube on the client computer. The virtual cube can include dimensions based on mining models.</td>
</tr>
<tr>
<td>CREATE MEMBER Statement</td>
<td>Creates a calculated member.</td>
</tr>
<tr>
<td>CREATE MINING MODEL Statement</td>
<td>Creates a local data mining model on the client computer.</td>
</tr>
<tr>
<td>CREATE SET Statement</td>
<td>Creates a user-defined set.</td>
</tr>
<tr>
<td>DROP_ACTION Statement</td>
<td>Deletes an action from the database.</td>
</tr>
<tr>
<td>DROP CUBE Statement</td>
<td>Deletes a cube from the database.</td>
</tr>
<tr>
<td>CREATE CELL CALCULATION Statement</td>
<td>Removes a calculated cell.</td>
</tr>
<tr>
<td>DROP LIBRARY Statement</td>
<td>Removes a user-defined function library from use during a session.</td>
</tr>
<tr>
<td>DROP MEMBER Statement</td>
<td>Deletes a calculated member.</td>
</tr>
<tr>
<td>DROP MINING MODEL Statement</td>
<td>Deletes a mining model.</td>
</tr>
<tr>
<td>DROP SET Statement</td>
<td>Deletes a user-defined set.</td>
</tr>
<tr>
<td><strong>REFRESH CUBE Statement</strong></td>
<td>Causes the memory cached on the client application to be synchronized with the server.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>USE LIBRARY Statement</strong></td>
<td>Loads a user-defined function library for use during a session.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**ALTER CUBE Statement**

This statement allows client applications to control the structure of a cube after it has been created.

**BNF**

_for updating the hierarchy of a calculated member_

ALTER CUBE <cube>

   UPDATE DIMENSION MEMBER <member> AS '<MDX rule>'

_for updating user defined default members_

ALTER CUBE <cube>

   UPDATE DIMENSION <dimension_name>, DEFAULT_MEMBER = '<MDX rule>'

_for updating dimensions_

<alter_statement> ::= <create_statement>|<remove_statement>|<move_statement>|<update_statement>

<create_statement> ::= CREATE DIMENSION MEMBER <parent_unique_name>.<member_name> [AS '<MDX expr.>', KEY='<key_value>' [, <property_name> = '<value>' [, <property_name> = '<value>' ... ]]

<remove_statement> ::= DROP DIMENSION MEMBER <member_unique_name> [WITH DESCENDANTS]

<move_statement> ::= MOVE DIMENSION MEMBER <member_unique_name> [, SKIPPED_LEVELS = '<value>'] [WITH DESCENDANTS] UNDER <member_unique_name>

<update_statement> ::= UPDATE DIMENSION MEMBER
<member_unique_name> {AS '<MDX expr.' | ,<property_name> = '<value>'}[
, <property_name> = '<value>'...
]

<member_unique_name> ::= <dimension_name>&[[<key>]]

**Remarks**

You can change the value of a custom rollup member by using the UPDATE DIMENSION MEMBER syntax.

Use the DEFAULT_MEMBER syntax to define a new default member in each dimension.

Alternatively, cubes that have parent-child relationships between members (that is, unbalanced and ragged hierarchies) can have their structures changed by the following basic operations:

- Creating a new dimension member

- Dropping a member of an existing dimension

- Moving an existing dimension member within the structure of the existing cube

- Updating the Multidimensional Expressions (MDX) definition of a dimension member

**Examples**

**A. Creating a New Dimension Member**

This example shows how to create a new dimension member. To add a member, specify its parent member in the CREATE DIMENSION MEMBER statement, and specify a key that uniquely identifies the member.

The following code adds Idaho and Boise to the Geography dimension:

```sql
ALTER_CUBE = ALTER CUBE Sales CREATE DIMENSION MEME
```
B. Dropping a Dimension Member

This example shows how to drop a dimension member and all its children. It drops the dimension member Idaho and all its children, including the city Boise, from the Geography dimension. If you omit the WITH DESCENDANTS phrase, the statement promotes Boise to the same level as Washington and Oregon.

ALTER_CUBE= ALTER CUBE Sales DROP DIMENSION MEMBER STATE_IDAHO WITH DESCENDANTS

C. Moving a Dimension Member

This example shows how to move a dimension member within the structure of a cube. It specifies both the node to move and its new position. The following code moves a member of the Employees dimension from under the [Southern California] hierarchy to the [Northwest] hierarchy. If you use the WITH DESCENDANTS phrase before the UNDER statement, child nodes move along with their parent.

ALTER_CUBE = ALTER CUBE Sales MOVE DIMENSION MEMBER Salesperson.[Francisco Ramirez] UNDER Region.Northwest

D. Defining a Default Member of a Dimension Hierarchy

This example shows how to define the default member of a dimension or hierarchy using the ALTER CUBE command. You can use any valid MDX expression that evaluates to a member in the definition of a default member:

ALTER_CUBE = ALTER CUBE Sales UPDATE DIMENSION Customers, DEFAULT_MEMBER = 'Customers.[All Customers].[USA]'
CREATE ACTION Statement

This statement allows the user to create action definitions that can be associated with a member and executed when that member is referenced.

BNF

ALTER CUBE <cube name>
    CREATE ACTION <action name> <action body>

CREATE ACTION <cube name>.<action name> <action body>

<action body> ::= FOR <target object> AS '<MDX expression>' [, TYPE = '<action type>' ] [, INVOCATION = '<action invocation>' ] [, APPLICATION = '<app name>' ] [, DESCRIPTION = '<action description>' ] [, CAPTION = '<MDX expression>' ]

<target object>::= CUBE
  |  <dimension name> [MEMBERS]
  |  <level_name> [MEMBERS]
  |  CELLS
  |  SET

<action type>::= URL | HTML | STATEMENT |
  DATASET | ROWSET | COMMANDLINE | PROPRIETARY

<action invocation>::= INTERACTIVE | ON_OPEN | BATCH

ALTER CUBE <cube name>
    DROP ACTION <action name>

DROP ACTION <cube name>.<action name>

Remarks
The following table describes the different types of actions available in Microsoft SQL Server 2000 Analysis Services.

<table>
<thead>
<tr>
<th>Action type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>The returned action string is a URL that should be launched using an Internet browser.</td>
</tr>
<tr>
<td>HTML</td>
<td>The returned action string is an HTML script. The string should be saved to a file and the file should be rendered using an Internet browser. In this case, a whole script may be executed as part of the generated HTML.</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>The returned action string is a statement that needs to be executed by setting the <code>ICommand::SetText</code> method of a command object to the string and calling the <code>ICommand::Execute</code> method. If the command does not succeed, an error is returned.</td>
</tr>
<tr>
<td>DATASET</td>
<td>The returned action string is a Multidimensional Expressions (MDX) statement that needs to be executed by setting the <code>ICommand::SetText</code> method of a command object to the string and calling the <code>ICommand::Execute</code> method. The requested interface ID (IID) should be <code>IDataset</code>. The command succeeds if a data set has been created. The client application should allow the user to browse the returned data set.</td>
</tr>
<tr>
<td>ROWSET</td>
<td>Similar to DATASET, but instead of requesting an IID of <code>IDataset</code>, the client application should ask for an IID of <code>IRowset</code>. The command succeeds if a rowset has been created. The client application should allow the user to browse the returned rowset.</td>
</tr>
<tr>
<td>COMMANDLINE</td>
<td>The client application should execute the action string. The string is a command line.</td>
</tr>
<tr>
<td>PROPRIETARY</td>
<td>A client application should not display nor execute the action unless it has a custom, nongeneric knowledge of the specific action. Proprietary actions are not returned to the client application unless the client application explicitly asks for these by setting the appropriate restriction on the APPLICATION_NAME.</td>
</tr>
</tbody>
</table>
It is possible for client applications to create and run actions that are unsafe; it is also possible for client applications to use unsafe functions. To avoid these situations, use the **UDF Safety Options** property. For more information, see [UDF Safety Options Property](#).

**Scope**

Each action is defined for a specific cube and has a unique name in that cube. An action can have one of the following scopes:

**Cube scope**

For actions independent on specific dimensions, members, or cells, for example: "Launch terminal emulation for AS/400 production system".

**Dimension scope**

The action applies to a specific dimension. Those actions are not dependent on specific selection of levels or members.

**Level scope**

The action applies to a specific dimension level. Those actions are not dependent on specific selection of a member in that dimension.

**Member scope**

The action applies to specific level members.

**Cell scope**

The action applies to specific cells only.

**Set scope**

The action applies to a set only. The name `ActionParameterSet` is reserved for use by the application inside the expression of the action.

**Examples**

**A. Creating an Action**
The following example creates an action that enables the client application to open a Web page with Active Server Pages (ASP) script. This script displays the details of a customer, given the customer ID number.

CREATE ACTION [Sales Cube].[Show Customer Details]
FOR [Customer] MEMBERS As
TYPE = URL
APPLICATION = 'IE'
DESCRIPTION = 'Launch the customer details page for this specific c

See Also

Actions
CREATE CACHE Statement

This statement populates the cache with a slice of cube data defined by sets of members.

You can define cache for use by a single query with the WITH clause in the SELECT statement or for use by multiple queries in a session with the CREATE CACHE statement. For more information, see Using WITH to Create Caches.

BNF

<create-cache-statement> ::= CREATE <optional-scope> <create-cache-subset> [<create-cache-subset>...]
<create-cache-subset> ::= CACHE FOR <cube-name> AS '(<set-expression> [,<set-expression>...])'
<optional-scope> ::= <empty> | SESSION
<cube name> ::= CURRENTCUBE | <Cube Identifier>

Remarks

Each <set-expression> token must contain members from only one dimension. Each member must be distinct. Each <set-expression> token must be from a different dimension.

The <set-expression> token can contain functions that support Multidimensional Expressions (MDX) syntax. The <set-expression> token cannot contain measures.

A cache created with CREATE CACHE without an <optional-scope> value token has session scope.

It is an error to specify a cube other than that to which it is currently connected. Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

Scope
Cache can occur within one of the following scopes:

**Query scope**

The lifetime of the cache is limited to the query. The cache is defined in an individual query. Query scope overrides session scope. For more information, see Using WITH to Create Caches.

**Session scope**

The lifetime of the cache is limited to the session in which it is created. The CREATE CACHE statement is used to create cache with session scope.

**Examples**

**Creating a Cache with Session Scope**

The following example creates a cache with session scope:

CREATE SESSION CACHE FOR Warehouse AS '{USA,Canada}'}
CREATE CELL CALCULATION Statement

This statement creates a calculated cell formula for a specified set of tuples within a cube.

BNF

<create cell formula> ::= CREATE CELL CALCULATION
  <cube name>.<formula name> <formula body> [ <conditions> ]
  | ALTER CUBE <cube name> CREATE CELL CALCULATION <formula name> <formula body>

<with cell formula> ::= WITH CELL CALCULATION <formula name> <formula body>

<formula body> ::= FOR '('<set description clause>')' AS '<formula clause>'
  [, <cell property list>]

<cell property list> ::= <condition property> <disabled property> <description property>
  <pass number property> <pass depth property>

<condition property> ::= [CONDITION = '<Conditions Expression>']

<condition expression> = <boolean member expression> [ & <condition expression>]

<disabled property> = [ , DISABLED = { TRUE | FALSE }]

<description property> = [ , DESCRIPTION = '<user-friendly description>']

<pass number property> ::= [ , CALCULATION_PASS_NUMBER = <long integer>]

<pass depth property> ::= [ , CALCULATION_PASS_DEPTH = <long integer>]

Remarks

By using calculated cells, the client application can specify a rollup value for a particular set of cells, instead of for an entire set of cells as in the case of a
custom rollup formula or a calculated member. For example, it is possible to specify that any cell in the slice defined by \{[Canada],[Time].[2000]\} can contain a value that is defined by a formula. Any other cells that are not contained within this slice would be computed normally.

**Examples**

**A. Creating a Calculated Cell with a Condition Clause**

The following example creates a calculated cell that adjusts values for cities in Mexico. It contains a condition that causes the formula to be applied only to members that reside in the year 2000 time period.

CREATE CELL CALCULATION [Sales].[Mexico Adjustments] FOR '(Descendants([Mexico], [City], SELF))' AS '<expression>',
CONDITION = '[Time].CURRENTMEMBER.NAME=[2000]'

**B. Creating a Calculated Cell Without a Condition Clause**

The same calculated cell can be defined by moving the CONDITION clause into the set description clause. This is the recommended method and is demonstrated in the following example:

CREATE CELL CALCULATION [Sales].[Mexico Adjustments] FOR '(Descendants([Mexico], [City], SELF), {[2000]})' AS '<expression>'

**See Also**

- DROP CELL CALCULATION Statement
- Calculated Cells
- Using WITH to Create Calculated Cells
CREATE CUBE Statement

This statement defines the structure of a new local cube. This statement shares much of the syntax and semantics of SQL-92 syntax and shares the semantics of the CREATE TABLE statement. However, the CREATE CUBE statement contains syntax specific to cubes.

The cube is not populated when the CREATE CUBE statement is executed. The cube is populated using the INSERT INTO statement in a manner similar to the SQL-92 approach for creating and populating tables.

CREATE CUBE Statement (Local Cube)

BNF

<create-cube-statement> ::= CREATE CUBE <cube name> <open paren> <dimensions def> <measures def> [ <command expression> ] <close paren> <dimensions def> ::= DIMENSION <dimension name> [ <time def> ] [DIMENSION_STRUCTURE <sub_type>] [<hidden def>] <options def> <comma> <hierarchy def list> <time def> ::= TIME | ...
<dimension name> ::= <legal name> <sub_type>::= PARENT_CHILD <hidden_def>::= HIDDEN <options def>::= OPTIONS <open paren> <dim options list> <close paren> <dim options list>::= <dim option> [ <comma> <dim options list>] <dim option>::= UNIQUE_NAME | UNIQUE_KEY | NOTRELATEDTOFACTTABLE | ALLOWSIBLINGSWITHSAMENAME <hierarchy def list>::= <hierarchy def> [ <comma> <hierarchy def list> <hierarchy def>::= [HIERARCHY <hierarchy name> [<hidden_def>] <comma>] <level def> <hierarchy name>::= <legal name>
<level def > ::= <parent-child level def> | <normal level def list >

<parent-child level def> ::= [ <all level def> <comma> ] LEVEL <Template>
//only if dimension is parent-child

<normal level def list> ::= <normal level def> [ <comma> <normal level def list> ]

<all level> ::= LEVEL <level name> TYPE ALL
<level name> ::= <legal name>
<normal level def> ::= [ <all level> <comma> ] LEVEL <level name> [ TYPE <level type> ] [ <level format def> ] [ <level options def> ] [ <hidden def> ] [ <hole def> ] [ <root member def> ] [ <custom_rollup_expr def> ] [ <comma> <level prop def list> ]

[level type] ::= YEAR
| QUARTER
| MONTH
| WEEK
| DAY
| DAYOFWEEK
| DATE
| HOUR
| MINUTE
| SECOND

[level format def] ::= FORMAT_NAME <expression> [FORMAT_KEY <expression>]

[level options def] ::= OPTIONS ( [ <sort option> <comma> ] <level option list> [ <level option list> <comma> ] <sort option> )
[level option list] ::= = <option> [ <comma> <level option list> ]

[sort option] ::= SORTBYNAME
| SORTBYKEY
| SORTBYPROPERTY <property name>

<option> ::= UNIQUE
| UNIQUE_NAME
| UNIQUE_KEY
| NOTRELATEDTOFACTTABLE

[hole def] ::= HIDE_MEMBER_IF <hide values>
[hide values] ::= ONLY_CHILD_AND_BLANK_NAME
| ONLY_CHILD_AND_PARENT_NAME
<root member def> ::= ROOT_MEMBER_IF <root values>

<root values> ::= ROOT_IF_PARENT_IS_BLANK
                 | ROOT_IF_PARENT_IS_MISSING
                 | ROOT_IF_PARENT_IS_SELF
                 | ROOT_IF_PARENT_IS_BLANK_OR_SELF_OR_MISSING

<custom_rollup_exp> ::= CUSTOM_ROLLUP_EXPRESSION <MDX expression>

[level prop def list > ::= <level prop def > [<comma> <level prop def list>]

[level prop def] ::= PROPERTY <legal name> [<prop type def>] [<hidden def>] [<prop caption def>]

[prop type def] ::= TYPE <prop type value>

[property_type value> ::= REGULAR
                      | ID
                      | RELATION_TO_PARENT
                      | ORG_TITLE
                      | CAPTION
                      | CAPTION_SHORT
                      | CAPTION_DESCRIPTION
                      | CAPTION_ABREVIATION
                      | WEB_URL
                      | WEB_HTML
                      | WEB_XML_OR_XSL
                      | WEB_MAIL_ALIAS
                      | ADDRESS
                      | ADDRESS_STREET
                      | ADDRESS_HOUSE
                      | ADDRESS_CITY
                      | ADDRESS_STATE_OR_PROVINCE
                      | ADDRESS_ZIP
                      | ADDRESS_QUARTER
                      | ADDRESS_COUNTRY
                      | ADDRESS_BUILDING
                      | ADDRESS_ROOM
Remarks

In the DIMENSION clause of the CREATE CUBE statement, the name given to
a level of TYPE ALL applies the specified name to the All member rather than the (All) level; the (All) level always has the name (All), including the parentheses. For example, the clause LEVEL [All Customers] TYPE ALL creates a level named (All) containing a single member named [All Customers]. There is no [All Customers] level.

If the <expression> value of the COMMAND clause has spaces, the entire expression should be surrounded by brackets. It is not recommended that quotation marks be used for this purpose because the body of the command might include quotation marks. (Microsoft® SQL Server™ 2000 Analysis Services supports nested brackets but not nested quotation marks.)

Examples

Creating a Local Cube

The following code shows how to define a local cube's dimensions and measures:

CREATE CUBE Sales
(
  DIMENSION Time TYPE TIME,
  HIERARCHY [Fiscal],
    LEVEL [Fiscal Year] TYPE YEAR,
    LEVEL [Fiscal Qtr] TYPE QUARTER,
    LEVEL [Fiscal Month] TYPE MONTH OPTIONS (SORTBYKEY
  HIERARCHY [Calendar],
    LEVEL [Calendar Year] TYPE YEAR,
    LEVEL [Calendar Month] TYPE MONTH,
  DIMENSION Products,
    LEVEL [All Products] TYPE ALL,
    LEVEL Category,
    LEVEL [Sub Category],
    LEVEL [Product Name],
  DIMENSION Geography,
    LEVEL [Whole World] TYPE ALL,
LEVEL Region,
LEVEL Country,
LEVEL City,
MEASURE [Sales]
  FUNCTION SUM
  FORMAT 'Currency',
MEASURE [Units Sold]
  FUNCTION SUM
  TYPE DBTYPE_UI4
)

CREATE CUBE Statement (Virtual Cube)

This statement facilitates the construction of complex data mining queries by client applications. Virtual cubes can be created that incorporate dimensions based on data mining models. Such dimensions are not related to fact tables.

BNF

<create vcube> ::= CREATE {SESSION} CUBE <cube name>
FROM <cube list>

<cube list> ::= <cube> [,<cube list>]
<param list> ::= <param>,<param list> | <param>
<param> ::= <measures list> | <dims list>

<measures list> ::= <measure> [,<measures list>]
<measure> ::= MEASURE <cube name>.<measure name> [ [<visibility qualifier> ] ] AS <measure name>
<visibility qualifier> ::= HIDDEN

<dim list> ::= <dim def> [, <dim list>]
<dim def> ::= <derived dim def> | <regular dim def>
<regular dim def> ::= DIMENSION <cube>,<dimension name>
<derived dim def> ::= DIMENSION <dim name>
<flags> [ [<visibility qual>] ]
FROM <from clause>
Remarks

This feature allows client applications to create virtual cubes on the fly in order to conduct more sophisticated analysis. The virtual cubes that are created are of session scope, and they cannot be saved on the client computer.

To delete a virtual cube, use the DROP CUBE statement.

Examples

Creating a Session Virtual Cube

The following example creates a virtual cube that contains portions of an existing cube named Children and the results of a data mining model named My DM Model:

CREATE SESSION CUBE [Student DMM]
    FROM [Children]
    (
        MEASURE [Children].[Count],
        MEASURE [Children].[Avg Age]
        DIMENSION [Children].[Population],
        DIMENSION [Children].[Area]
        DIMENSION [Decision Tree] NOT_RELATED_TO_FACTS
            FROM [My DM Model] COLUMN [Measures.Sales]
    )
CREATE MEMBER Statement

This statement creates a calculated member.

You can define a calculated member for use by a single query with the WITH clause in the SELECT statement, or for use in multiple queries in a session with the CREATE MEMBER statement. For more information, see Using WITH to Create Calculated Members.

BNF

```<create-member-statement> ::= CREATE <optional-scope> <create-member-subset> [<create-member-subset>...]<create-member-subset> ::= MEMBER <cube-name>.<fully-qualified-member-name> AS '<expression>' [,<property-definition-list>]<cube name> ::= CURRENTCUBE | <Cube Identifier><property-definition-list> ::= <property-definition> | <property-definition>,<property-definition-list><property-definition> ::= <property-identifier> = <property-value><property-identifier> ::= VISIBLE | SOLVEORDER | FORMAT_STRING| <ole db member properties><property-value> ::= <string> | <number><optional-scope> ::= <empty> | SESSION```

Remarks

The <expression> clause of the calculated member syntax can contain any function that supports Multidimensional Expressions (MDX) syntax. Valid <property-identifier> values are listed later in this topic. Calculated members created with CREATE MEMBER without an <optional-scope> value have session scope. Additionally, strings inside calculated member definitions are delimited with double quotation marks. This is opposite of the method defined by OLE DB, which specifies that strings should be delimited by single quotation marks.

It is an error to specify a cube other than that to which it is currently connected.
Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

For more information about member properties that are defined by OLE DB, see the OLE DB documentation.

A calculated member can be stored in a local cube if a CREATE MEMBER statement is specified in the COMMAND clause of the CREATE CUBE statement:

```
CREATE CUBE MYWAREHOUSE
  
  DIMENSION . . .
  
  . . .,
  
  COMMAND (CREATE MEMBER [MYWAREHOUSE].[MEASURES].[WAREHOUSEPROFIT] AS '[MEASURES].[WAREHOUSE SALES] - [MEASURES].[WAREHOUSE COST]')
```

**Standard Properties**

Each calculated member has a set of default properties. When a client application is connected to Microsoft® SQL Server™ 2000 Analysis Services, the default properties are either supported or available to be supported, as the administrator chooses.

Additional member properties may be available, depending upon the cube definition. The following properties represent information relevant to the dimension level in the cube.

<table>
<thead>
<tr>
<th>Property identifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SolveOrder</td>
<td>The order in which the calculated member will be solved in cases where a calculated member references one more other calculated member (that is, where calculated members intersect each other).</td>
</tr>
<tr>
<td>Format_String</td>
<td>A Microsoft Office style format string that the client application can use when displaying cell values.</td>
</tr>
<tr>
<td>Visible</td>
<td>Determines whether the calculated member is visible in a schema rowset. Visible calculated members can</td>
</tr>
</tbody>
</table>
be added to a set with the `AddCalculatedMembers` function. A nonzero value indicates that the calculated member is visible. The default value for this property is Visible.

Calculated members that are not visible (where this value is set to zero) are generally used as intermediate steps in more complex calculated members. These calculated members can also be referred to by other types of members, such as measures.

**Scope**

A calculated member can occur within one of the following scopes:

**Query scope**

The visibility and lifetime of the calculated member is limited to the query. The calculated member is defined in an individual query. Query scope overrides session scope. For more information, see [Using WITH to Create Calculated Members](#).

**Session scope**

The visibility and lifetime of the calculated member is limited to the session in which it is created. (The lifetime is less than the session duration if a DROP MEMBER statement is issued on the calculated member.) The CREATE MEMBER statement is used to create a calculated member with session scope.

**Examples**

**Creating Calculated Members**

The following code creates two calculated members:

```sql
```

See Also

Calculated Members
CREATE CUBE Statement
CREATE MINING MODEL Statement

This statement creates a local data mining model on the client computer. You can create mining models from relational databases, PMML, or OLAP cubes.

BNF (CREATE MINING MODEL)

<dm_create>::= CREATE MINING MODEL <identifier> ( <col_def_list> ) USING <algorithm> [(<algo_param_list>)]

<pmml_create>::= CREATE MINING MODEL <identifier> FROM PMML <string>

<select_into>::= SELECT * INTO <identifier> USING <algorithm> FROM <identifier>

<col_def_list>::= <col_def> | <col_def_list> , <col_def>
<col_def>::= <col_def_reg> | <col_def_tbl>
<col_def_reg>::= <identifier> <col_type> [<col_distribution>] [<col_binary>] [<col_content>] [<col_content_qual>] [<col_qualif>] [<col_prediction>] [<relation_clause>]
<col_def_tbl>::= <identifier> TABLE <col_prediction> ( <col_def_list> )
<algorithm>::= MICROSOFT_DECISION_TREES | MICROSOFT_CLUSTERING
<algo_param>::= <identifier> = <value>
<algo_param_list>::= <algo_param>
 | <algo_param>, <algo_param_list>
<col_type>::= LONG
 | BOOLEAN
 | TEXT
 | DOUBLE
 | DATE
<col_distribution>-> NORMAL
| UNIFORM
<col_binary>::= MODEL_EXISTENCE_ONLY
  | NOT NULL

<col_content>::= DISCRETE
  | CONTINUOUS
  | DISCRETIZED( [<disc_method> [,<numeric_const>]] )
  | SEQUENCE_TIME

<disc_method>::= AUTOMATIC
  | EQUAL_AREAS
  | THRESHOLDS
  | CLUSTERS

<col_content_qual>::= ORDERED
  | CYCLICAL

<col_qualif>::= KEY
  | PROBABILITY
  | VARIANCE
  | STDEV
  | STDDEV
  | PROBABILITY_VARIANCE
  | PROBABILITY_STDEV
  | PROBABILITY_STDDEV
  | SUPPORT

<col_prediction> -> PREDICT
  | PREDICT_ONLY

<relation_clause> -> <related_to_clause>
  | <of_clause>

<related_to_clause>::= RELATED TO <identifier>
  | RELATED TO KEY

<of_clause>::= OF <identifier>
  | OF KEY

BNF (CREATE OLAP MINING MODEL)
Use this syntax to create mining models that are based on OLAP cubes instead of relational database tables. Each OLAP mining model contains one or more case dimensions and zero or more case measures. Columns within each case can be based on any object in the Dimension object model, such as a hierarchy, level, or property, or can be based upon the value of a measure. The flags that are used with each OLAP mining model column are the same as those used for relational mining models. OLAP mining models are trained in the same manner as relational mining models, using the same syntax.

<olap create statement> ::= CREATE OLAP MINING MODEL <dmm name>
FROM <cube name> <olap definition>
USING <dmm algorithm> [(dmm flag list)]

<olap definition> ::= CASE <olap dimension> [, <olap dimension list>] [, <olap measure list>]

<olap dimension list> ::= <olap dimension> [, <olap dimension list>]

<olap dimension> ::= DIMENSION <dimension name> <predict qualifier>
{ <olap level list> | <olap hierarchy list> }

<olap hierarchy list> ::= <olap hierarchy>
[, <olap hierarchy list>]

<olap hierarchy> ::= HIERARCHY <hierarchy name> <predict qualifier> <olap level list>

<olap level list> ::= <olap level> [, <olap level list>]

<olap level> ::= LEVEL <level name> <predict qualifier> <olap property list>

<olap property list> ::= <olap property> [, <olap property list>]

<olap property> ::= PROPERTY <property name> <predict qualifier>

<olap measure list> ::= <olap measure> [, <olap measure list>]

<olap measure> ::= MEASURE <measure name> <predict qualifier>
<predict qualifier> = <nothing> | PREDICT | PREDICT_ONLY

<dmm flag list> ::= <dmm flag> [, <dmm flag list>]
<dmm flag> ::= <flag name> = <value>
<flag Name> ::= <col_type> [<col_distribution>] [<col_binary>] [<col_content>] [<col_content_qual>] [<col_qualif>]

**Remarks**

The CREATE MINING MODEL statement creates a new mining model based on the column definition list. Each column is described by content flags in the column definition. These flags provide additional information to the mining algorithm concerning the content of the training data or model. No more than one flag from a particular group can be used (that is, flags within a flag type group are exclusive of each other) and they must be placed in their correct order. The flag type groups and correct orders for the content flags are listed in the following table.

<table>
<thead>
<tr>
<th>Flag type</th>
<th>Flag name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>NORMAL</td>
<td>The values of the column appear in a normal distribution.</td>
</tr>
<tr>
<td></td>
<td>LOG NORMAL</td>
<td>The values of the column appear in a log normal distribution.</td>
</tr>
<tr>
<td></td>
<td>UNIFORM</td>
<td>The values of the column appear in a uniform distribution.</td>
</tr>
<tr>
<td>Content Type</td>
<td>KEY</td>
<td>The column is discrete and is a key. Key columns will not have any other flags except in the case of a nested table with no attribute columns.</td>
</tr>
<tr>
<td></td>
<td>CONTINUOUS</td>
<td>The column contains values in a continuous range, such as Age or Salary.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>DISCRETE</td>
<td>The column contains a discrete set of values, such as Gender.</td>
<td></td>
</tr>
<tr>
<td>DISCRETIZED()</td>
<td>The column contains a continuous set of values that should be converted to buckets.</td>
<td></td>
</tr>
<tr>
<td>ORDERED</td>
<td>The column contains a discrete set of values that are ordered, such as Salary Level.</td>
<td></td>
</tr>
<tr>
<td>CYCLICAL</td>
<td>The column contains an ordered discrete set of values that are cyclical, such as Day of Week or Month.</td>
<td></td>
</tr>
<tr>
<td>SEQUENCE TIME</td>
<td>The column contains time measurement units.</td>
<td></td>
</tr>
<tr>
<td>Modeling</td>
<td>MODEL_EXISTENCE_ONLY</td>
<td></td>
</tr>
<tr>
<td>NOT NULL</td>
<td>The column cannot accept NULL values.</td>
<td></td>
</tr>
<tr>
<td>Special Property</td>
<td>PROBABILITY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The value in this</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>COLUMN</td>
<td>column is the probability (0-1) of the associated value.</td>
<td></td>
</tr>
<tr>
<td>VARIANCE</td>
<td>The value in this column is value variance of the associated value.</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>The value in this column is the standard deviation of the associated value.</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY VARIANCE</td>
<td>The value in this column is the variance of the probability associated with the associated value.</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY STD</td>
<td>The value in this column is the standard deviation of the probability associated with the associated value.</td>
<td></td>
</tr>
<tr>
<td>SUPPORT</td>
<td>The value in this column is the weight (case replication factor) of the associated value.</td>
<td></td>
</tr>
</tbody>
</table>

Column relations are described in one of the following ways.

<table>
<thead>
<tr>
<th>&lt;Column relation&gt; clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF</td>
<td>This form is restricted to use for columns with Special Property content flags, for example,</td>
</tr>
</tbody>
</table>
ProbGender Double PROBABILITY OF Gender.

<table>
<thead>
<tr>
<th>RELATED TO</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>This form indicates a value hierarchy. The target of a related to column</td>
<td>This form indicates a value hierarchy. The target of a related to</td>
</tr>
<tr>
<td>can be a key column in a nested table, a discretely valued column on the</td>
<td>column can be a key column in a nested table, a discretely valued</td>
</tr>
<tr>
<td>case row, or another column with a RELATED TO clause (indicating a deeper</td>
<td>case row, or another column with a RELATED TO clause (indicating a</td>
</tr>
<tr>
<td>hierarchy).</td>
<td>deeper hierarchy).</td>
</tr>
</tbody>
</table>

The following flags are used to describe how a prediction column functions.

<table>
<thead>
<tr>
<th>Prediction flag clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREDICT</td>
<td>This column can be predicted by the model and it can be supplied in input cases to predict the value of other predictable columns.</td>
</tr>
<tr>
<td>PREDICT ONLY</td>
<td>This column can be predicted by the model, but its values cannot be used in input cases to predict the value of other predictable columns.</td>
</tr>
</tbody>
</table>

**See Also**

[Building a Data Mining Model](#)
CREATE SET Statement

This statement creates user-defined sets.

You can define a set for use by a single query with the WITH clause in the SELECT statement or for use in multiple queries in a session with the CREATE SET statement. For more information about WITH, see Using WITH to Create Named Sets.

BNF

<create-set-statement> ::= CREATE <optional-scope> <create-set-subset> [<create-set-subset>...]
<create-set-subset> ::= SET <cube-name>.<set-name> AS '<set-expression>'
<cube name> ::= CURRENTCUBE | <Cube Identifier>
<optional-scope> ::= <empty> | SESSION

Remarks

A named set is a set of dimension members (or an expression that defines a set) that is created to be used again. For example, by using a named set it is possible to define a set of dimension members that consists of the set of top 10 stores by sales. This set can be defined statically, or by means of a function like TOPCOUNT. This named set can then be used wherever the set of top 10 stores is needed.

The <expression> clause of the calculated member syntax can contain any function that supports Multidimensional Expressions (MDX) syntax. Sets created with the CREATE SET statement that do not specify an <optional-scope> clause have session scope.

It is an error to specify a cube other than that to which it is currently connected. Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

Scope
A user-defined set can occur within one of the following scopes:

**Query scope**

The visibility and lifetime of the set is limited to the query. The set is defined in an individual query. Query scope overrides session scope. For more information, see [Using WITH to Create Named Sets](#).

**Session scope**

The visibility and lifetime of the set is limited to the session in which it is created. (The lifetime is less than the session duration if a DROP SET statement is issued on the set.) The CREATE SET statement is used to create a set with session scope.

**Examples**

**A. Creating a Named Set Using a Function Expression**

The following example creates a named set consisting of the top ten stores, as ranked by their sales, in the Sales cube:

```sql
CREATE SET [Sales].[TopStores] as 'TopCount([Store].Members,10,[Measures].[Store Sales])'
```

**B. Creating a Named Set Using a Set Expression**

In this example, a named set is statically defined to consist of states in the Northwest region of the United States:

```sql
CREATE SET [Sales].[NorthwesternStores] as '
  { [Store].[All Stores].[USA].[WA],
    [Store].[All Stores].[USA].[OR],
    [Store].[All Stores].[USA].[ID] }
'
DROP ACTION Statement

This statement deletes an action from the database.

BNF

<drop-action-statement> ::= DROP ACTION <action-name>

Example

The following example drops the action defined by the example in CREATE ACTION Statement:

DROP ACTION [Sales Cube].[Show Customer Details]
DROP CUBE Statement

This statement deletes a local cube or a virtual cube.

BNF

<drop-cube-statement> ::= DROP [SESSION] CUBE <cube-name>

Remarks

This statement deletes the cube or virtual cube specified in <cube-name>.

Example

The following example drops the cube created by the example in CREATE CUBE Statement:

DROP CUBE [Sales]
DROP CELL CALCULATION Statement

This statement removes the specified calculated cell.

BNF

<drop cell formula> ::= DROP CELL CALCULATION <cube name>.<formula name>
   | ALTER CUBE <cube name> DROP CELL CALCULATION <formula name>

Example

The following example deletes a cell formula from the Sales cube:

DROP CELL CALCULATION [Sales].[Budget Adjustment]
Analysis Services Programming

**DROP LIBRARY Statement**

This statement unloads the specified libraries.

**BNF**

\[
<\text{drop\_library}> ::= \text{DROP LIBRARY} \ <\text{lib\_list}> | \text{ALL}
\]

\[
<\text{lib\_list}> ::= <\text{lib\_def}> [\ , <\text{lib\_list}>]
\]

\[
<\text{lib\_def}> ::= <\text{prog\_id}> | <\text{lib\_name}>
\]

**Remarks**

When used with the ALL flag, DROP LIBRARY unloads all libraries loaded for that user session. Either a program ID or a file name is used to specify individual libraries.

**Example**

The following example removes MyLib.dll from use for the rest of the session.

DROP LIBRARY MyLib.MyClass
DROP MEMBER Statement

This statement deletes a calculated member that has been defined for the session.

BNF

\[
\text{<drop-member-statement>} \ ::= \text{DROP MEMBER} \text{ <cube-name>}.\text{<fully-qualified-member-name>}
\]

Example

The following example drops a named set created by the first example in CREATE MEMBER Statement:

DROP MEMBER [Warehouse].[Measures].[warehouseprofit]
DROP MINING MODEL Statement

This statement deletes a mining model.

BNF

\[<\text{drop-mining-model-statement}> ::= \text{DROP MINING MODEL} \ <\text{model-name}>\]

Remarks

If the **Mining Location** property is set to a directory path, this statement deletes the model that resides in that directory. This model is specified by \(<\text{model-name}>\).
DROP SET Statement

This statement deletes a user-defined set that has been defined for the session.

BNF

\(<\text{drop-set-statement}\> \::= \text{DROP SET} \ <\text{cube-name}> , \ <\text{set-name}>\)

Example

The following examples drop the named set statements created by the examples in CREATE SET Statement:

DROP SET [Sales].[TopStores]
CREATE SET [Sales].[NorthwesternStores]
Analysis Services Programming

**REFRESH CUBE Statement**

This statement refreshes the client cache for a cube.

**BNF**

<refresh-cube-statement> ::= **REFRESH CUBE** <cube-name>

**Remarks**

For client applications connected to the Analysis server, this statement causes the memory cached on the client application to be synchronized with the server.

For client applications connected to a local cube, the REFRESH CUBE statement causes the local cube file to be rebuilt.

**Example**

The following example refreshes the client cache that pertains to a cube called [Sales]:

REFRESH CUBE [Sales]
Analysis Services Programming

**USE LIBRARY Statement**

This statement loads a function library for use during the session.

**BNF**

<Use-Library-statement> ::= USE LIBRARY <Library-Name-Clause>

**Remarks**

Use this statement to load a user-defined function.

User-defined function libraries should be implemented as COM components. These libraries can be implemented as in-process servers (in a .dll) or as local servers (in an .exe). Before loading a user-defined function library, ensure that the library contains a type library. Additionally, all of the interfaces defined in the type library must be derived from IDISPATCH for automation. User-defined function libraries can be developed in any environment capable of generating COM components.

**Examples**

The following examples demonstrate defining and using a user-defined function library.

**A. Creating a User-Defined Function**

In the following example, a Microsoft® Visual Basic® function is defined that converts currency based upon the exchange rate of a given country:

```vbnet
Public Function Convert(country As String, Value As Double) As Double
    Select Case country
        Case "USA"
            Convert = Value * 1
        Case "Canada"
            Convert = Value * 1.5486
    End Select
End Function
```
Case "Mexico"
   Convert = Value * 9.93
End Select
End Function

**B. Using a User-Defined Function Library**

To use this function with Microsoft SQL Server™ 2000 Analysis Services, place it into a Visual Basic ActiveX® DLL Project. To load the library for use in Analysis Services, use the USE LIBRARY statement. In the following example, a user-defined function library is loaded for use during the session, and a query is defined that uses a query scoped calculated member containing the user-defined function:

```
USE LIBRARY "UDF.Currency"
WITH Member Measures.SalesNC AS
   'UDF!_Currency!Convert(
       [Sales].[Customers].[Country],
       [Sales].[Measures].Members
   )
SELECT {SalesNC} ON COLUMNS FROM Sales'
```

**See Also**

[DROP LIBRARY Statement](#)
Analysis Services Programming
# Data Manipulation Language

The following topics describe the data manipulation language (DML) used by PivotTable® Service.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DRILLTHROUGH Statement</strong></td>
<td>Retrieves the source rowset(s) from the fact table (that is, data source) for a specified tuple.</td>
</tr>
<tr>
<td><strong>INSERT INTO Statement</strong></td>
<td>Describes the INSERT INTO statement, which populates a local cube with dimension members.</td>
</tr>
<tr>
<td><strong>SELECT Statement</strong></td>
<td>Describes the SELECT statement, which is used to create queries that return multidimensional data, either in a Microsoft® ActiveX® Data Objects (ADO) <strong>Cellset</strong> object or in an OLE DB <strong>Dataset</strong> object.</td>
</tr>
<tr>
<td><strong>UPDATE CUBE Statement</strong></td>
<td>Describes the UPDATE CUBE statement, which allocates values from a nonleaf member cell update to all of the children of that member.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**DRILLTHROUGH Statement**

This statement retrieves the source rowset(s) from the fact table (that is, data source) for a specified tuple.

**BNF**

```xml
<drillthrough> ::= DRILLTHROUGH [Max_Rows] [First_Rowset]
<MDX select>
  <Max_Rows>    ::= MAXROWS <positive number>
  <First_Rowset> ::= FIRSTROWSET <positive number>
```

**Remarks**

This statement allows the client application to retrieve the rowsets that were used to create a specified cell in a cube. A Multidimensional Expressions (MDX) statement is used to specify the subject cell. If this cell is at an atomic level (that is, at the lowest level of its hierarchy), only one rowset is returned. If this cube is not at an atomic level, all of the rowsets that make up the source data of that cell are returned. The total number of rowsets returned can also be affected by use of the MAXROWS and FIRSTROWSET modifiers.

The value specified by the MAXROWS modifier indicates the maximum number of rows that should be returned by the resulting rowset. This modifier should only be used if the original source data's OLE DB provider supports the DBPROP_MAXROWS property.

The value specified by the FIRSTROWSET modifier specifies the first rowset to return. Use of this modifier is not recommended unless the client application designer does not wish to use the OLE DB **IMultipleResults** interface or the Microsoft® ActiveX® Data Objects (ADO) **NextRecordset** method to navigate the returned rowsets.

For more information, see [Using DRILLTHROUGH to Retrieve Source Data](#).
**INSERT INTO Statement**

This statement has two functions: It populates local cubes with dimension members, and it trains data mining models. If the local cube is stored in multidimensional OLAP (MOLAP), the INSERT INTO statement also populates the local cube with data.

**BNF**

```
<insert-into-statement> ::= INSERT INTO <target-clause> [<options-clause>] [<bind-clause>] <source-clause>
    | INSERT INTO <model> (<mapped model columns>) <source data query>
    | INSERT INTO <model> (<mapped model columns>) VALUES <constant list>
    | INSERT INTO <model>.COLUMN_VALUES(<mapped model columns>) <source data query>
<mapped model columns> ::= <column identifier> | <table identifier>(<column identifier> | SKIP), ...
<target-clause> ::= <cube-name> <open-paren> <target-element-list> <close-paren>
<target-element-list> ::= <target-element>[, <target-element-list>]
<target-element> ::= [<dim-name>.[<hierarchy-name>].]<level-name>
    | [Measure].<measure-name>
    | SKIPONECOLUMN
<level-name> ::= <simple-level-name>
    | <simple-level-name>.NAME
    | <simple-level-name>.KEY
    | <simple-level-name>.Custom_Rollup
    | .parent
    | <simple-level-name>.SkipLevelColumn
<time-dim-name> ::= <dim-name-type-time>
    | <dim-name-type-time>.NAME
    | <dim-name-type-time>.KEY
<options-clause> ::= OPTIONS <options-list>
```
<options-list> ::= <option>[, <options-list>]
<option> ::= <defer-options>
| < analysis-options>
<defer-options> ::= DEFER_DATA
| ATTEMPT_DEFER
<analysis-options> ::= PASSTHROUGH
| ATTEMPT_ANALYSIS
/bind-clause> ::= BIND (<bind-list>)
/bind-list> ::= <simple-column-name>[,<simple-column-name>]
/simple-column-name> ::= <identifier>
/source-clause> ::= SELECT <columns-list>
  FROM <tables-list>
  [ WHERE <where-clause> ]
  | DIRECTLYFROMCACHEDROWSET <hex-number> | DIRECTLYFROMMARSHALLEDROWSET <hex number>
/columns-list> ::= <column-expression>[,<columns-list>]
/column-expression> ::= <column-expression-name>
/column-expression-name> ::= <column-name> [AS <alias-name>]
| <alias name> <column-name>
/column-name> ::= <table-name>.<column-name>
| <column-function>
| <ODBC scalar function>
| <braced-expression>
/column function> ::= <identifier>(. . .)
/ODBC scalar function> ::= {FN<column-function>}
/braced-expression> ::= ( . . .)
/tables-list> ::= <table-expression>[,<tables-list>]
/table-expression> ::= <table-name> [ [AS] <table-alias>]
/table-alias> ::= <identifier>
/table-name> ::= <identifier>
/where-clause> ::= <where-condition> [AND <where-clause>]
/where-condition> ::= <join-constraint>
  | <application constraint>
/join-constraint> ::= <column-name> = <column-name>
  | <open-paren><column-name> = <column-name><close-paren>
/application-constraint> ::= ( . . .)
  | NOT ( . . .)
Remarks
The behavior and use of this statement depend on whether you use it for OLAP or data mining.

Using INSERT INTO with Local Cubes

Names of elements in an INSERT INTO statement are level and measure names, sometimes qualified with dimension name or the keyword Measures to avoid ambiguity. The Measures keyword is case-sensitive in binary comparisons. If you use binary comparison or are unsure of your comparison method, use Measures as shown with only M in upper case.

Each level and each measure in a cube is derived from a column in the SELECT clause except the (All) level.

The columns specified in the associated SELECT clause are bound to the elements of the INSERT INTO statement in the order specified and in a one-to-one relationship.

Each level can be derived from two columns, with one used as a name column and the other used as a key column. Both columns must be in the same table. If there are two columns associated with a level, use the suffix .NAME or .KEY properties in the INSERT INTO statement after the level name.

If a column specified in the SELECT clause does not have a related element in the INSERT INTO statement, the keyword SKIPONECOLUMN can be used in the INSERT INTO statement as a placeholder for the unused column. SKIPONECOLUMN can be used more than once.

In the INSERT INTO statement, you can specify a dimension of TYPE TIME by using the name of the dimension. The dimension name is used to correlate the entire dimension with a single column in the source table that contains data with
a date/time data type. The levels, of TYPE <level type>, identified for the time dimension in the CREATE_CUBE statement cause the time information to be extracted from the source column specified in the SELECT clause. For more information, see Example D later in this topic.

The WHERE clause can have both application and join constraints. The parser parses only join constraints. It uses the join constraint to find a path from all tables to the fact table and to the dimension tables. The application constraint is used only to specify constraints on a fact table and is passed through without modifications.

Expressions between parentheses are considered to be application constraints. For example, if the expression Sales.Product_ID = Products.Product_ID AND Sales.Customer_ID = Customers.Customer_ID is enclosed in parentheses, it is treated as an application constraint and is not used as a join constraint. It is the responsibility of the client application to ensure that parentheses are used only around application constraints: for example, (Product.Price < 100 AND Product.Category = 1).

The BIND clause is used to bind level and measure names specified in the INSERT INTO statement with column names used to create rowsets.

The AS <alias-name> syntax is not supported for local relational OLAP (ROLAP) cubes.

**Using INSERT INTO with Data Mining**

The INSERT INTO statement inserts training data into the model. The columns from the query are mapped to model columns through the <mapped model columns> section. The keyword **SKIP** is used to instruct the model to ignore columns that appear in the source data query that are not used in the model.

The INSERT INTO <model>.COLUMN_VALUES form inserts data directly into the models columns without training the model's algorithm. This allows you to provide column data to the model in a concise ordered manner that is useful when dealing with datasets containing hierarchies or ordered columns. The period (.) operator is used to specify columns that are part of a nested table. When using this form, columns that are part of a relation (either through RELATE TO or by being a KEY in a nested table) cannot be inserted individually and must be inserted together with all the columns in the relation.
Using Passthrough and Advanced Query Processing

The PASSTHROUGH option causes the SELECT clause to be passed directly to the source database without modification by PivotTable® Service. If PASSTHROUGH is not specified, PivotTable Service parses the query and formulates a set of queries equivalent to the original that is optimized for the source database and index structures. This set of queries is often more efficient than the specified query.

The DEFER_DATA option causes the query to be parsed locally and executed only when necessary to retrieve data to satisfy a user request. DEFER_DATA is used to specify that a local cube be defined in the ROLAP storage mode.

The ATTEMPT_DEFER option causes PivotTable Service to attempt to parse the query and defer data loading if successful, or, if the query cannot be parsed, to process the specified query immediately as if the PASSTHROUGH had been specified.

The ATTEMPT_ANALYSIS option causes PivotTable Service to attempt to parse the query and formulate an optimized set of queries. If the query cannot be parsed, PivotTable Services processes the query immediately as if the PASSTHROUGH had been specified.

Passthrough Compatibility Matrix

These options can be used together in combination. Use the following matrix to determine the effect of a pair of options while building a local cube.

<table>
<thead>
<tr>
<th></th>
<th>Parse</th>
<th>Neither PassThrough nor Attempt_Analysis</th>
<th>PassThrough</th>
<th>Attempt_Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither DEFER_DATA nor ATTEMPT_DEFER</td>
<td>Succeeded</td>
<td>MOLAP</td>
<td>MOLAP(*)</td>
<td>MOLAP</td>
</tr>
<tr>
<td></td>
<td>Failed</td>
<td>Error</td>
<td>n/a</td>
<td>MOLAP(*)</td>
</tr>
<tr>
<td>DEFER_DATA</td>
<td>Succeeded</td>
<td>ROLAP</td>
<td>Error</td>
<td>ROLAP</td>
</tr>
<tr>
<td></td>
<td>Failed</td>
<td>Error</td>
<td>n/a</td>
<td>Error</td>
</tr>
<tr>
<td>ATTEMPT_DEFER</td>
<td>Succeeded</td>
<td>ROLAP</td>
<td>MOLAP(*)</td>
<td>ROLAP</td>
</tr>
<tr>
<td>Failed</td>
<td>MOLAP(*)</td>
<td>n/a</td>
<td>MOLAP(*)</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>-----</td>
<td>----------</td>
<td></td>
</tr>
</tbody>
</table>

* Indicates that this pair of options will cause the SELECT clause to be passed through to the source database.

## Examples

### A. Using an Application Constraint

The following example demonstrates how to use an application constraint:

```
INSERT INTO MyCube (Year, Month.Name, Month.Key, [Product Gr OPTIONS DEFER_DATA
SELECT MyTable.Year, MyTable.Month, MONTH(MyTable.Month), FROM MyTable
WHERE MyTable.SalesRep = "Amir" and MyTable.CustomerGroup =
```

### B. Using the PASSTHROUGH Option

The following example demonstrates how to use the PASSTHROUGH option.

**Note** The PASSTHROUGH option specifies that the SELECT clause that follows it is to be passed directly to the database engine with no parsing by PivotTable Service. This option cannot be used with the DEFERED_DATA option.

```
INSERT INTO MyCube (Year, Month, [Product Group], [Product Narr OPTIONS PASSTHROUGH SELECT MyTable.Year, MyTable.Month,
FROM MyTable
WHERE MyTable.SalesRep = "Amir" and MyTable.CustomerGroup =
```

### C. Using the DIRECTLYFROMCACHEDROWSET Keyword

The following example demonstrates how to use the DIRECTLYFROMCACHEDROWSET keyword.

**Note** The DIRECTLYFROMCACHEDROWSET keyword directs data to be read from the address in memory identified immediately after the keyword. It is the responsibility of the client application to specify the correct address in memory. At run time, the number is assumed to be the in-process address of an
IUnknown pointer to an OLE DB rowset.

INSERT INTO MyCube (Year, Month, [Product Group], [ProductName], Country, Sales, [Units Sold])
DIRECTLYFROMCACHEDROWSET 0x00001284

D. Using the DEFER DATA Option to Create a ROLAP Cube

The following example demonstrates how to create a ROLAP cube by using the DEFER DATA option:

CREATE CUBE MyCube (  
  DIMENSION TimeDim TYPE TIME,  
  LEVEL MyYear TYPE YEAR,  
  LEVEL MyQtr TYPE QUARTER,  
  LEVEL MyMonth TYPE MONTH,  
  DIMENSION Products,  
  LEVEL [Product Group],  
  LEVEL [Product Name],  
  DIMENSION Geography,  
  LEVEL State,  
  LEVEL City,  
  MEASURE [Sales]  
    FUNCTION SUM  
    FORMAT 'Currency',  
  MEASURE [Units Sold]  
    FUNCTION SUM
)

INSERT INTO MyCube (TimeDim, [Product Group], [ProductName], OPTIONS DEFER_DATA
SELECT MyTable.TransDate, MyTable.ProdGroup, MyTable.ProdName, MyTable.Sales, MyTable.UnitsSold
FROM MyTable
WHERE MyTable.SalesRep = "Jacobsen" and MyTable.CustomerGroup = "Industry"

E. Training a Data Mining Model

The following example trains a data mining model called [Age Prediction]. The
training columns for this model are [Gender], [Product Name], [Product Type], and [Month].

```sql
INSERT INTO [Age Prediction].COLUMN_VALUES(Gender)
OPENROWSET('SQLOLEDB', '...', 'SELECT DISTINCT Gender FROM Customers')

INSERT INTO [Age Prediction].COLUMN_VALUES([Product Purchases].[Product Name], [Product Purchases].[Product Type])
OPENROWSET('SQLOLEDB', '...', 'SELECT DISTINCT [Product Name], [Product Type] FROM Sales')

INSERT INTO [Age Prediction].COLUMN_VALUES(SKIP, [Month])
OPENROWSET('SQLOLEDB', '...', 'SELECT MonthID, Month FROM Months ORDER BY MonthID')
```
Analysis Services Programming

**SELECT Statement**

This statement is used to create queries that return multidimensional data, either in a Microsoft® ActiveX® Data Objects (ADO) **Cellset** object or in an OLE DB **Dataset** object.

**BNF**

**BNF for SELECT (OLAP)**

```
<select_statement> ::= [WITH <single_formula_specification>
[<single_formula_specification>...]]
SELECT [<axis_specification> [, <axis_specification>...]]
FROM <cube_specification>
[WHERE <slicer_specification>]
[cell_props]

<single_formula_specification> ::= <member_specification>
| <set_specification>
| <cache_specification>

<member_specification> ::= MEMBER <parent_of_member>.<member_name>
AS '<value_expression>'
[, <solve_order_specification>]
[, <member_property_definition>...]

<solve_order_specification> ::= SOLVE_ORDER = <unsigned_integer>
<member_property_definition> ::= <member_property_name> = <value_expression>

<set_specification> ::= SET <set_name> AS '<set>'
<cache_specification> ::= CACHE AS '(<set>[, <set>])'
[axis_specification] ::= [NON EMPTY] <set> [dim_props] ON <axis_name>
<set> ::= member:member
| <set_value_expression>
| { <set> | <tuple> [, <set> | <tuple>...] }
```
BNF for SELECT (Data Mining)

<column_ref_list>::= <column_ref>
    | <column_ref_list> , <column_ref>

<column_ref>::= <identifier>
    | <identifier>.<column_ref>
    | <column_ref> ( <column_ref_list> )
    | SKIP
    | CLUSTER()
    | $SUPPORT
| $VARIANCE |
| $STDEV |
| $STDDEV |
| $PROBABILITY |
| $PROBABILITY_VARIANCE |
| $PROBABILITY_STDEV |
| $PROBABILITY_STDDEV |
| $ADJUSTEDPROBABILITY |
| $DISTANCE |
| PREDICT ( <column_ref> [, <pred_option_list>] ) |

| <column_ref> AS <identifier> |

| <pred_option_list>-> <pred_option> |
| <pred_option_list>, <pred_option> |

| <pred_option>::= EXCLUDE_NULL |
| INCLUDE_NULL |
| INPUT_ONLY |
| EXCLUSIVE |
| INCLUSIVE |
| INCLUDE_STATISTICS |

| <select>::= <pred_select> |
| <model_select> |

| <pred_select>::= SELECT [FLATTENED] <expression_list> FROM <identifier> [NATURAL] PREDICTION JOIN <query> AS <identifier> [ON <on_list>] [WHERE <where_clause>] |

| SELECT [FLATTENED] <expression_list> FROM <identifier> [NATURAL] PREDICTION JOIN <expression> AS <identifier> [ON <on_list>] [WHERE <where_clause>] |

| <model_select>::= SELECT [DISTINCT] <expression_list> FROM <identifier> [WHERE <where_clause>] |

| SELECT [DISTINCT] <expression_list> FROM <identifier>.PMML |

| SELECT [DISTINCT] <expression_list> FROM <identifier>.CONTENT [WHERE <where_clause>] |
<expression_list>::= <expression> \\
   | <expression_list>, <expression>

<expression>::= <value> \\
   | <column_ref> \\
   | * \\
   | <expression> + <expression> \\
   | <expression> - <expression> \\
   | <expression> * <expression> \\
   | <expression> / <expression> \\
   | -(<expression>) \\
   | +<expression> \\
   | ( <expression> ) \\
   | <expression> OR <expression> \\
   | <expression> AND <expression> \\
   | NOT <expression> \\
   | <expression> = <expression> \\
   | <expression> <> <expression> \\
   | <expression> < <expression> \\
   | <expression> <= <expression> \\
   | <expression> > <expression> \\
   | <expression> >= <expression> \\
   | PREDICTSTDEV ( <column_ref> ) \\
   | PREDICTSTDDEV ( <column_ref> ) \\
   | PREDICTVARIANCE ( <column_ref> ) \\
   | PREDICTSUPPORT ( <column_ref> ) \\
   | PREDICTPROBABILITY ( <column_ref> ) \\
   | PREDICTADJUSTEDPROBABILITY ( <column_ref> ) \\
   | CLUSTERDISTANCE ([<expression>]) \\
   | CLUSTERPROBABILITY ([<expression>]) \\
   | PREDICTHISTOGRAM ( <column_ref> ) \\
   | TOPCOUNT ( <expression>, <column_ref>, <expression> ) \\
   | TOPSUM ( <expression>, <column_ref>, <expression> ) \\
   | TOPPERCENT ( <expression>, <column_ref>, <expression> ) \\
   | BOTTOMCOUNT ( <expression>, <column_ref>, <expression> ) \\
   | BOTTOMSUM ( <expression>, <column_ref>, <expression> ) \\
   | BOTTOMPERCENT ( <expression>, <column_ref>, <expression> )
Note  Microsoft SQL Server™ 2000 Analysis Services data mining algorithms do not support probability variance or probability standard deviation. The columns $PROBABILITY\_VARIANCE$, $PROBABILITY\_STDEV$, and $PROBABILITY\_STDEV$ always contain 0.

The $ADJUSTEDPROBABILITY$ column is an Analysis Services extension to the OLE DB for Data Mining specification.

Remarks

In the \(<\text{tuple}> ::= (<\text{member}>[,<\text{member}>...])\) syntax, each \(<\text{member}>\) value must be from a different dimension.

In the \(<\text{slicer\_specification}>\) syntax, the members in the \(<\text{tuple}>\) value must be in dimensions other than those in the \(<\text{axis\_specification}>\) values.

If a dimension in the cube is omitted from the \(<\text{axis\_specification}>\) values and \(<\text{slicer\_specification}>\) value, the dimension's default member is implicitly added to the \(<\text{slicer\_specification}>\) value.

The DISTINCT keyword is ignored in data mining queries.

Inserting Comments

Like SQL, Multidimensional Expressions (MDX) syntax can contain user-readable comments that are ignored when the commands are processed. The three different character sets that indicate comments are outlined in the following table.
## Characters

<table>
<thead>
<tr>
<th>Characters</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>// (C++ style forward slashes)</td>
<td>All text after the forward slashes (/) and before the end of the same line is ignored.</td>
</tr>
<tr>
<td>-- (SQL hyphens)</td>
<td>All text after the hyphens (-) and before the end of the same line is ignored.</td>
</tr>
<tr>
<td>/<em>...</em>/ (C style slash and asterisk pairs)</td>
<td>All text between the opening forward slash (/) and asterisk and the closing asterisk (*) and closing forward slash (/) is ignored. This type of comment can span multiple lines.</td>
</tr>
</tbody>
</table>

## Example

The following example shows the use of comments in an MDX command:

```
/*
   Using this query to view
   information about units shipped
   and units ordered */
SELECT
  { [Measures].[Units Shipped], [Measures].[Units Ordered] } ON COLUMNS,
  // The next command specifies nonempty members only
  NON EMPTY [Store].[Store Name].Members ON ROWS
FROM Warehouse  -- Pulled from the Warehouse cube
```

For more information, see [Comments in MDX](#).

## See Also

[MDX](#)
Analysis Services Programming

**UPDATE CUBE Statement**

This statement portions out, according to a specified formula, the delta of an updated cell value to all of the children of that member. This method of updating the contents of a cube is called allocation, and is only supported on measures using the SUM aggregation type.

**BNF**

<update_statement> ::= UPDATE [CUBE] <cube_specification>

    SET

    <cell_update>[, <cell_update>...]  

<cell update> ::=  <tuple>.VALUE = <value>  

    [ USE_EQUAL_ALLOCATION |
    USE_EQUAL_INCREMENT |
    USE_WEIGHTED_ALLOCATION [BY < weight value_expression>] |
    USE_WEIGHTED_INCREMENT [BY <weight value_expression>] ]

<Tuple> is a set of coordinates. If the full set of coordinates is not specified, it is assumed that the unspecified coordinates are the default member of the dimension.

The <tuple> can be any cell in the multidimensional space (that is, it does not have to be an atomic cell). However, the cell must be aggregated with the SUM aggregate function and must not use a calculated member as one of its coordinates.

**Remarks**

It may be helpful to think of the UPDATE CUBE statement as a subroutine that will automatically generate a series of individual writeback operations to atomic cells that will roll up into a specified sum.
The following table describes the methods of allocation.

<table>
<thead>
<tr>
<th>Allocation method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE_EQUAL_ALLOCATION</td>
<td>Every atomic cell that contributes to the updated cell will be assigned an equal value that is:</td>
</tr>
<tr>
<td></td>
<td>(&lt;\text{atomic cell value}&gt; = \frac{\text{&lt;value&gt;}}{\text{Count(atomic cells contained in &lt;tuple&gt;)}})</td>
</tr>
<tr>
<td>USE_EQUAL_INCREMENT</td>
<td>Every atomic cell that contributes to the updated cell will be changed according to:</td>
</tr>
<tr>
<td></td>
<td>(&lt;\text{atomic cell value}&gt; = &lt;\text{atomic cell value}&gt; + \frac{(\text{&lt;value&gt; - &lt;existing value&gt;})}{\text{Count(atomic cells contained in &lt;tuple&gt;)}})</td>
</tr>
<tr>
<td>USE_WEIGHTED_ALLOCATION</td>
<td>Every atomic cell that contributes to the updated cell will be assigned an equal value that is:</td>
</tr>
<tr>
<td></td>
<td>(&lt;\text{atomic cell value}&gt; = \text{&lt;value&gt; \times &lt;weight value expression&gt;})</td>
</tr>
<tr>
<td>USE_WEIGHTED_INCREMENT</td>
<td>Every atomic cell that contributes to the updated cell will be changed according to:</td>
</tr>
<tr>
<td></td>
<td>(&lt;\text{atomic cell value}&gt; = &lt;\text{atomic cell value}&gt; + \frac{(\text{&lt;value&gt; - &lt;existing value&gt;})}{\text{&lt;weight value expression&gt;}})</td>
</tr>
</tbody>
</table>

If the value \(<\text{weight value expression}>\) is not provided, the following expression
is assigned to it by default:

\[<\text{weight value expression}> = \frac{<\text{atomic cell value}>}{<\text{existing value}>}\]

The value of \(<\text{weight value expression}>\) should be expressed as a value between 0 and 1. This value specifies the ratio of the allocated value you want to assign to the atomic cells that are affected by the allocation. It is the client application programmer's responsibility to create expressions whose rollup aggregate values will equal the allocated value of the expression.

**CAUTION** The client application must take into account the allocation of all dimensions concurrently to avoid possible unexpected results, including incorrect rollup values or inconsistent data.

Each UPDATE CUBE allocation should be considered to be atomic for transactional purposes. This means that if any one of the allocation operations fails for any reason, such as an error in a formula or a security violation, then the whole UPDATE CUBE operation will fail. Before the calculations of the individual allocation operations are processed, a snapshot of the data is taken to ensure that the resulting calculations are correct.

**CAUTION** When used on a measure containing integers, the USE_WEIGHTED_ALLOCATION method can return imprecise results due to incremental rounding changes.

**Examples**

**Using UPDATE CUBE**

UPDATE CUBE [Budget Cube]
  SET
  ([1999], [Marketing], [Budget], [All Departments]) = 1000

USE_WEIGHTED_ALLOCATION BY
  ([1998], [Sales], [Actual])/
  ([1999], [Sales], [Actual], [All Departments])
Analysis Services Programming
Function Reference

PivotTable® Service has access to an extensive library of OLAP and data mining functions. The following topics cover the functions available to PivotTable Service.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OLAP Functions</strong></td>
<td>Discusses OLAP functions detailed in the MDX Function Reference and in the OLE DB for OLAP specification.</td>
</tr>
<tr>
<td><strong>Data Mining Functions</strong></td>
<td>Covers data mining functions detailed in the OLE DB for Data Mining specification.</td>
</tr>
</tbody>
</table>
Analysis Services Programming

**OLAP Functions**

Microsoft® SQL Server™ 2000 Analysis Services supplies a wide variety of functions, through the use of Multidimensional Expressions (MDX) function libraries.

For more information about OLAP functions, see [MDX Function Reference](#).
Data Mining Functions

Microsoft® SQL Server™ 2000 Analysis Services supplies a number of functions which retrieve and manipulate statistical information from a data mining model.

The following topics discuss these functions in greater detail.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BottomCount</strong></td>
<td>Returns a table containing a specified number of bottommost rows in increasing order of rank based on a rank expression.</td>
</tr>
<tr>
<td><strong>BottomPercent</strong></td>
<td>Returns a table containing the smallest number of bottommost rows, in increasing order of rank based on a rank expression, that meet a specified percent expression.</td>
</tr>
<tr>
<td><strong>BottomSum</strong></td>
<td>Returns a table containing the smallest number of bottommost rows, in increasing order of rank based on a rank expression, that meet a specified sum expression.</td>
</tr>
<tr>
<td><strong>Cluster</strong></td>
<td>For clustering data mining models, returns the cluster identifier containing the highest probability of the input case.</td>
</tr>
<tr>
<td><strong>ClusterDistance</strong></td>
<td>Returns the distance between the input case and the center of the cluster that has the highest probability.</td>
</tr>
<tr>
<td><strong>ClusterProbability</strong></td>
<td>Returns the probability that the input case belongs to the cluster that has the highest probability.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Predict</strong></td>
<td>Performs a prediction based on a specified column.</td>
</tr>
<tr>
<td><strong>PredictAdjustedProbability</strong></td>
<td>Retrieves the adjusted probability of the topmost histogram entry for a specified column.</td>
</tr>
<tr>
<td><strong>PredictHistogram</strong></td>
<td>Retrieves a table representing the histogram for a specified column.</td>
</tr>
<tr>
<td><strong>PredictProbability</strong></td>
<td>Retrieves the probability of the topmost histogram entry for a specified column.</td>
</tr>
<tr>
<td><strong>PredictStdev</strong></td>
<td>Retrieves the standard deviation value of the topmost histogram entry for a specified column.</td>
</tr>
<tr>
<td><strong>PredictSupport</strong></td>
<td>Retrieves the support value of the topmost histogram entry for a specified column.</td>
</tr>
<tr>
<td><strong>PredictVariance</strong></td>
<td>Retrieves the variance value of the topmost histogram entry for a specified column.</td>
</tr>
<tr>
<td><strong>RangeMax</strong></td>
<td>Retrieves the upper value of the predicted bucket discovered for a specified discretized column.</td>
</tr>
<tr>
<td><strong>RangeMid</strong></td>
<td>Retrieves the midpoint value of the predicted bucket discovered for a specified discretized column.</td>
</tr>
<tr>
<td><strong>RangeMin</strong></td>
<td>Retrieves the lower value of the predicted bucket discovered for a specified discretized column.</td>
</tr>
<tr>
<td><strong>Sub-SELECT</strong></td>
<td>Returns a table from a specified table expression.</td>
</tr>
<tr>
<td><strong>TopCount</strong></td>
<td>Returns a table containing a specified number of topmost rows in a decreasing order of rank based on a rank expression.</td>
</tr>
<tr>
<td><strong>TopPercent</strong></td>
<td>Returns a table containing the</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>smallest number of topmost rows, in a decreasing order of rank based on a rank expression, that meet a specified percent expression.</td>
<td><strong>TopSum</strong> Returns a table containing the smallest number of topmost rows, in a decreasing order of rank based on a rank expression, that meet a specified sum expression.</td>
</tr>
</tbody>
</table>
**BottomCount**

The **BottomCount** function returns the specified number of bottommost rows in increasing order of rank as specified by an expression.

**Syntax**

`BottomCount(<table expression>, <rank expression>, <count>)`

**Applies to**

An expression that returns a table, such as a `<table column reference>` or a function that returns a table.

**Return Type**

`<table expression>`

**Remarks**

The value supplied by the `<rank expression>` argument is used to determine the increasing order of rank for the rows supplied in the `<table expression>` argument, and the number of bottommost rows specified in the `<count>` argument is returned.

For more information, see [TopCount](#).
**BottomPercent**

The **BottomPercent** function returns, in order of increasing rank, the bottommost rows of a table whose cumulative total is at least a specified percentage.

**Syntax**

```plaintext
BottomPercent(<table expression>, <rank expression>, <percent>)
```

**Applies to**

An expression that returns a table, such as a `<table column reference>` or a function that returns a table.

**Return Type**

`<table expression>`

**Remarks**

The **BottomPercent** function returns the bottommost rows in increasing order of rank based on the evaluated value of the `<rank expression>` argument for each row, such that the sum of the `<rank expression>` values is at least the given percentage specified by the `<percent>` argument. **BottomPercent** returns the smallest number of elements possible while still meeting the specified percent value.

For more information, see [TopPercent](#).
BottomSum

The **BottomSum** function returns, in order of increasing rank, the bottommost rows of a table whose cumulative total is at least a specified value.

**Syntax**

```
BottomSum(<table expression>, <rank expression>, <sum>)
```

**Applies to**

An expression that returns a table, such as a `<table column reference>` or a function that returns a table.

**Return Type**

`<table expression>`

**Remarks**

The **BottomSum** function returns the bottommost rows in increasing order of rank based on the evaluated value of the `<rank expression>` argument for each row, such that the sum of the `<rank expression>` values is at least the given total specified by the `<sum>` argument. **BottomSum** returns the smallest number of elements possible while still meeting the specified sum value.

For more information, see [TopSum](#).
Cluster

The Cluster function identifies the cluster to which the input case belongs with the highest probability.

Syntax

Cluster

Applies to

This function does not require any parameter, but it can be used only when the underlying data mining model supports clustering.

Return Type

This function returns a scalar value of a cluster identifier, referred to in other data mining functions as a clusterID. However, if this function is used as an argument of other functions, it must be regarded as a cluster column reference.

Remarks

Cluster can also be used as a cluster column reference for a PredictHistogram function.

See Also

ClusterDistance
ClusterProbability
ClusterDistance

The ClusterDistance function returns the distance between the input case and the center of the cluster that has the highest probability.

Syntax

ClusterDistance([<ClusterID expression>])

Applies to

This function can be used only when the underlying data mining model supports clustering.

Return Type

Scalar value

Remarks

If <ClusterID expression> is specified, the cluster is identified by the evaluation of the expression.

See Also

Cluster

ClusterProbability
ClusterProbability

The ClusterProbability function returns the probability that the input case belongs to the cluster that has the highest probability.

Syntax

ClusterProbability([<ClusterID expression>])

Applies to

This function can be used only when the underlying data mining model supports clustering.

Return Type

Scalar value

Remarks

If <ClusterID expression> is specified, the cluster is identified by the evaluation of the expression.

See Also

Cluster
ClusterDistance
**Predict**

The **Predict** function is a general prediction function that modifies the behavior of a prediction such as missing value control, association control, and so on.

**Syntax**

Predict(<scalar column reference>, option1, option2, ...)

Predict(<table column reference>, option1, option2, ...)

**Applies to**

Either a scalar column or table column reference.

**Return Type**

<scalar column reference>

or

<table column reference>

The return type depends on the type of column to which this function is applied.

**Remarks**

Possible options include EXCLUDE_NULL (default), INCLUDE_NULL, INCLUSIVE, EXCLUSIVE (default), INPUT_ONLY, and INCLUDE_STATISTICS.

**Note**  INCLUSIVE, EXCLUSIVE, INPUT_ONLY, and INCLUDE_STATISTICS are applicable only for a table column reference, and EXCLUDE and INCLUDE_NULL apply only for scalar values columns.

The following alternative abbreviated forms are often used:

- [Gender] is shorthand for **Predict**([Gender], EXCLUDE_NULL).
- [Products Purchases] is an alternative for Predict([Products Purchases], EXCLUDE_NULL, EXCLUSIVE_ASSOCIATION).

**Note** The return type of this function is itself regarded as a column reference. This means that this function can be used as an argument in other functions that take a column reference as an argument (except the Predict function itself).

Passing INCLUDE_STATISTICS to a prediction on a TABLE-valued column will add the metacolumns $Probability and $Support to the resulting table. These columns describe the likelihood of existence for the associated nested table record.
**PredictAdjustedProbability**

The *PredictAdjustedProbability* function returns the adjusted probability for the histogram entry that has the highest probability.

**Syntax**

`PredictAdjustedProbability(<scalar column reference>)`

**Applies to**

Scalar column

**Return Type**

Scalar value

**Remarks**

*PredictAdjustedProbability* returns the top row in the histogram obtained by *PredictHistogram(<column reference>).*

The *PredictAdjustedProbability* function is a Microsoft® SQL Server™ 2000 Analysis Services extension to the OLE DB for Data Mining specification.
**PredictHistogram**

The **PredictHistogram** function returns a table representing a histogram for prediction of the given column.

**Syntax**

`PredictHistogram(<scalar column reference> | <cluster column reference>)`

**Applies to**

A scalar or cluster column reference.

**Return Type**

Table

**Remarks**

A histogram generates statistics columns. The column structure of the returned histogram depends on the type of column reference used with the **PredictHistogram** function.

**Scalar Columns**

For a `<scalar column reference>`, the histogram returned by the **PredictHistogram** function consists of the following seven columns:

- The column being predicted
- $Support
- $Variance
- $Stdev$ (standard deviation)

- $Probability$

- $ProbabilityVariance$
  
  Microsoft® SQL Server™ 2000 Analysis Services data mining algorithms do not support $ProbabilityVariance$. This column always contains 0.

- $ProbabilityStdev$
  
  Analysis Services data mining algorithms do not support $ProbabilityStdev$. This column always contains 0.

- $AdjustedProbability$
  
  The $AdjustedProbability$ column is an Analysis Services extension to the OLE DB for Data Mining specification.

**Cluster Columns**

The histogram returned by the PredictHistogram function for a $<cluster
column reference>$ consists of the following columns:

- **Cluster** (represents the cluster identifier)

- $Distance$

- $Probability$

- $Support$

**See Also**

Cluster
ClusterDistance
ClusterProbability
PredictAdjustedProbability
PredictProbability
PredictStdev
PredictSupport
PredictVariance
**PredictProbability**

The **PredictProbability** function returns the probability for the histogram entry that has the highest probability, which is the top row in the histogram obtained by **PredictHistogram(<column reference>)**.

**Syntax**

```
PredictProbability(<scalar column reference>)
```

**Applies to**

Scalar column

**Return Type**

Scalar value
**PredictStdev**

The **PredictStdev** function returns the standard deviation for the histogram entry that has the highest probability, which is the top row in the histogram obtained by **PredictHistogram(<column reference>)**.

**Syntax**

**PredictStdev(<scalar column reference>)**

**Applies to**

Scalar column

**Return Type**

Scalar value
The **PredictSupport** function returns the support value for the histogram entry that has the highest probability, which is the top row in the histogram obtained by **PredictHistogram**(*<column reference>*).

**Syntax**

**PredictSupport**(*<scalar column reference>*)

**Applies to**

Scalar column

**Return Type**

Scalar value
**PredictVariance**

The **PredictVariance** function returns the variance value for the histogram entry that has the highest probability, which is the top row in the histogram obtained by **PredictHistogram(</column reference>)**.

**Syntax**

**PredictVariance(<scalar column reference>)**

**Applies to**

Scalar column

**Return Type**

Scalar value
RangeMax

The **RangeMax** function returns the upper end of the predicted bucket that was discovered for a discretized column.

**Syntax**

```
RangeMax(<scalar column reference>)
```

**Applies to**

Discretized scalar columns

**Return Type**

Scalar value
**RangeMid**

The **RangeMid** function returns the midpoint of the predicted bucket that was discovered for a discretized column.

**Syntax**

`RangeMid(<scalar column reference>)`

**Applies to**

Discretized scalar columns

**Return Type**

Scalar value
RangeMin

The **RangeMin** function returns the lower end of the predicted bucket that was discovered for a discretized column.

**Syntax**

`RangeMin(<scalar column reference>)`

**Applies to**

Discretized scalar columns

**Return Type**

Scalar value
Sub-SELECT

A Sub-SELECT selects columns (or expressions containing columns) from the given table-returning expression.

Syntax

(SELECT <SELECT-expressions> FROM <table expression> [WHERE <WHERE-clause>])

Applies to

A table-returning expression that includes <table column reference> and functions that return a table.

Return Type

<table expression>

Remarks

An optional WHERE clause can be used to filter returned rows.
### TopCount

The **TopCount** function returns the specified number of topmost rows in a decreasing order of rank as specified by an expression.

#### Syntax

```
TopCount(<table expression>, <rank expression>, <count>)
```

#### Applies to

An expression that returns a table, such as a `<table column reference>` or a function that returns a table.

#### Return Type

`<table expression>`

#### Remarks

The value supplied by the `<rank expression>` argument is used to determine the decreasing order of rank for the rows supplied in the `<table expression>` argument, and the number of topmost rows specified in the `<count>` argument is returned.

For example, assume that this **Sub-SELECT** contains the following table:

```
(SELECT [Product Name], $Probability AS [Probability] FROM Predict([Products Purchases], INCLUDE_STATISTICS))
```

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apples</td>
<td>0.4</td>
</tr>
<tr>
<td>Kiwi</td>
<td>0.1</td>
</tr>
<tr>
<td>Oranges</td>
<td>0.5</td>
</tr>
<tr>
<td>Lemons</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Using the **TopCount** function with the Sub-SELECT as a parameter as shown
yields the following results:

\textbf{TopCount}((SELECT [Product Name], $Probability$ AS [Probability] FROM Predict([Products Purchases], INCLUDE_STATISTICS)), [Probability], 2

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oranges</td>
<td>0.5</td>
</tr>
<tr>
<td>Apples</td>
<td>0.4</td>
</tr>
</tbody>
</table>
**TopPercent**

The **TopPercent** function returns, in order of decreasing rank, the topmost rows of a table whose cumulative total is at least a specified percentage.

**Syntax**

TopPercent(<table expression>, <rank expression>, <percent>)

**Applies to**

An expression that returns a table, such as a `<table column reference>` or a function that returns a table.

**Return Type**

`<table expression>`

**Remarks**

The **TopPercent** function returns the topmost rows in decreasing order of rank based on the evaluated value of the `<rank expression>` argument for each row, such that the sum of the `<rank expression>` values is at least the given percentage specified by the `<percent>` argument. **TopPercent** returns the smallest number of elements possible while still meeting the specified percent value.

For example, assume that a table column named [Products] contains this table:

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Unit Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apples</td>
<td>30</td>
</tr>
<tr>
<td>Kiwi</td>
<td>10</td>
</tr>
<tr>
<td>Oranges</td>
<td>40</td>
</tr>
<tr>
<td>Lemons</td>
<td>20</td>
</tr>
</tbody>
</table>

TopPercent([Products], [Unit Sales], 60) function returns the following
Table:

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Unit Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oranges</td>
<td>40</td>
</tr>
<tr>
<td>Apples</td>
<td>30</td>
</tr>
</tbody>
</table>

Note that Apples was selected instead of Lemons.
TopSum

The TopSum function returns, in order of decreasing rank, the topmost rows of a table whose cumulative total is at least a specified value.

Syntax

TopSum(<table expression>, <rank expression>, <sum>)

Applies to

An expression that returns a table, such as a <table column reference> or a function that returns a table.

Return Type

<table expression>

Remarks

The TopSum function returns the topmost rows in decreasing order of rank based on the evaluated value of the <rank expression> argument for each row, such that the sum of the <rank expression> values is at least the given total specified by the <sum> argument. TopSum returns the smallest number of elements possible while still meeting the specified sum value.

For example, assume that a table column named [Products] contains this table:

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Unit Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apples</td>
<td>1200</td>
</tr>
<tr>
<td>Kiwi</td>
<td>500</td>
</tr>
<tr>
<td>Oranges</td>
<td>1500</td>
</tr>
<tr>
<td>Lemons</td>
<td>750</td>
</tr>
</tbody>
</table>

TopSum([Products], [Unit Sales], 2500) returns the following table:
<table>
<thead>
<tr>
<th>Product Name</th>
<th>Unit Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oranges</td>
<td>1500</td>
</tr>
<tr>
<td>Apples</td>
<td>1200</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Schema Rowsets

In OLE DB, the schema for an object is a description of the object's structure (that is, the contents of that object's meta data). A schema rowset is an OLE DB rowset that encapsulates that description for all objects of particular type within the database. Each row in the rowset corresponds to an individual object. The individual properties of the objects contained in the rowset are contained within the columns of the rowset.

In addition to the columns returned by the schema rowset, OLE DB provides a mechanism, called a restriction column, for filtering these schema rowsets based upon the content of certain columns. For each schema rowset, a set of restriction columns is specified; the client application can use these columns to filter the results of the schema rowset. When more than one restriction column is specified for a schema rowset, the columns are combined using a logical AND statement. For instance, if a user is interested only in dimensions that are contained within the Sales cube of the FoodMart 2000 database, the client application can set the CATALOG_NAME restriction column to equal "FoodMart 2000" and the CUBE_NAME restriction column to equal "Sales".

Schema rowsets used for online analytical processing (OLAP) are documented in the OLE DB specification. Microsoft® SQL Server™ 2000 Analysis Services provides additional rowsets and additional columns for some specified rowsets to provide functionality beyond that addressed in the OLE DB specification. For information about the schema rowsets used by Analysis Services for OLAP, see the OLE DB documentation and OLAP Schema Rowsets.

Schema rowsets used for data mining are documented in the OLE DB for Data Mining specification. All data mining schema rowsets implemented by Analysis Services in this release are described in this documentation, regardless of whether they are also documented in the new OLE DB for Data Mining specification. For information about the schema rowsets used by Analysis Services for data mining, see Data Mining Schema Rowsets.

A C++ header file, Msmd.h, contains the GUIDs for the schema rowsets that are supported in Analysis Services beyond those defined in OLE DB. Msmd.h is installed with Analysis Services samples. The default installation folder is C:\Program Files\Analysis Services\Samples\Include.
For more information about OLE DB schema rowsets, search on "OLE DB schema rowsets" in the Platform SDK portion of the MSDN® library at Microsoft Web site.
OLAP Schema Rowsets

The following table describes the schema rowsets that are used by Microsoft® SQL Server™ 2000 Analysis Services for online analytical processing (OLAP).

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDSHEMA_ACTIONS</td>
<td>Contains information about the actions schema rowset, which describes the actions that may be available to the client application</td>
</tr>
<tr>
<td>MDSHEMA_CELL_FORMULAS</td>
<td>Contains information about the calculated cells schema rowset, which describes the calculated cells that may be contained within a database</td>
</tr>
<tr>
<td>MDSHEMA_CUBES</td>
<td>Contains information about the cubes schema rowset, which describes the structure of cubes that are contained within a database</td>
</tr>
<tr>
<td>MDSHEMA_DIMENSIONS</td>
<td>Contains information about the dimensions schema rowset, which describes the shared and private dimensions that are contained within a database</td>
</tr>
<tr>
<td>MDSHEMA_FUNCTIONs</td>
<td>Contains information about the functions schema rowset, which describes the functions that are available to client applications connected to the database</td>
</tr>
<tr>
<td>MDSHEMA_HIERARCHIES</td>
<td>Contains information about the hierarchies schema rowset, which describes each hierarchy that is contained within a particular dimension</td>
</tr>
<tr>
<td>MDSHEMA_LEVELS</td>
<td>Contains information about the levels schema rowset, which describes the levels of each hierarchy</td>
</tr>
<tr>
<td>Table Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MDSchema_Measures</td>
<td>Contains information about the measures schema rowset, which describes each measure contained within a cube</td>
</tr>
<tr>
<td>MDSchema_Members</td>
<td>Contains information about the members schema rowset, which describes the members contained within a database</td>
</tr>
<tr>
<td>MDSchema_Properties</td>
<td>Contains information about the properties schema rowset, which describes the properties of members contained within a database</td>
</tr>
<tr>
<td>MDSchema_Sets</td>
<td>Contains information about the sets schema rowset, which describes any sets that are currently defined</td>
</tr>
</tbody>
</table>
**MDSCHEMA_ACTIONS**

This schema rowset describes the actions that may be available to the client application.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the MDSCHEMA_ACTIONS schema rowset. The following table describes this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the catalog to which this action belongs.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>DBTYPE_WSTR</td>
<td>This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the cube to which this action belongs.</td>
</tr>
<tr>
<td>ACTION_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of this action.</td>
</tr>
<tr>
<td>ACTION_TYPE</td>
<td>DBTYPE_I4</td>
<td>A bitmap that is used to specify the action's triggering method. The following bit value constants are defined in Msmd.h for this bitmap:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_URL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_HTML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_STATEMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_DATASET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_ROWSET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_COMMANDLINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDACTION_TYPE_PROPRIETARY</td>
</tr>
<tr>
<td>COORDINATE</td>
<td>DBTYPE_WSTR</td>
<td>A Multidimensional Expressions (MDX) expression that…</td>
</tr>
<tr>
<td>Field</td>
<td>DBTYPE</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COORDINATE</td>
<td>WSTR</td>
<td>A Multidimensional Expressions (MDX) expression that specifies an object or a coordinate in the multidimensional space in which the action is executed. It is the responsibility of the client application to provide the value of this restriction column.</td>
</tr>
<tr>
<td>COORDINATE_TYPE</td>
<td>I4</td>
<td>A bitmap that specifies how the COORDINATE restriction column is interpreted. The following bit value constants are defined in Msmd.h for this bitmap:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION COORDINATE_CUBE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION COORDINATE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION COORDINATE_LEVEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION COORDINATE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION COORDINATE_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION COORDINATE_CELL</td>
</tr>
<tr>
<td>ACTION.Caption</td>
<td>WSTR</td>
<td>The label or a caption associated with this action.</td>
</tr>
<tr>
<td>Description</td>
<td>WSTR</td>
<td>A user-friendly description of the action.</td>
</tr>
<tr>
<td>Content</td>
<td>WSTR</td>
<td>The expression or content of the action that is to be executed.</td>
</tr>
<tr>
<td>Application</td>
<td>WSTR</td>
<td>The name of the application that is to be used to execute the action.</td>
</tr>
<tr>
<td>Invocation</td>
<td>I4</td>
<td>Provides information about how the action should be invoked:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION_INVOCATION_INTERACTIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Regular action used during normal operations. This is the default value for this column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD ACTION_INVOCATION_ON_OPEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action should be executed when the cube is first opened.</td>
</tr>
</tbody>
</table>
Action should be executed when the cube is first opened.

- **MDACTION_INVOCATION_BATCH**
  Action executes as part of a batch operation or DTS task.

These enumeration values are defined in MSMD.h.

The sort order for this schema rowset is the same as the definition for this schema rowset.

**Note** Actions of **MDACTION_TYPE_PROPRIETARY** type must provide a value for the **APPLICATION** column.

### Restriction Columns

The actions schema rowset contains three mandatory restrictions that must be specified when retrieving a schema rowset. Failing to specify a mandatory restriction column results in an error. The following table contains a list of restriction columns and describes whether they are mandatory.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Restriction state</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Optional</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>Optional</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Mandatory</td>
</tr>
<tr>
<td>ACTION_NAME</td>
<td>Optional</td>
</tr>
<tr>
<td>ACTION_TYPE</td>
<td>Optional</td>
</tr>
<tr>
<td>COORDINATE</td>
<td>Mandatory</td>
</tr>
<tr>
<td>COORDINATE_TYPE</td>
<td>Mandatory</td>
</tr>
<tr>
<td>INVOCATION</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Important** The **INVOCATION** restriction column has a default value of **MDACTION_INVOCATION_INTERACTIVE**. Any schema rowset that does not explicitly specify a value for this column contains only rows with this value.
If you want the rowset to contain the entire set of actions, use the MDACTION_INVOCATION_ALL constant in the INVOCATION restriction column.

Client applications can define more than one ACTION_TYPE by using the OR operator.

See Also

Actions

CREATE ACTION Statement
**MDSCHEMA_CELL_FORMULAS**

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the MDSCHEMA_CELL_FORMULAS schema rowset. The following table describes this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the catalog to which a set belongs. If the provider does not support catalogs, this column contains VT_NULL.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the schema to which a calculated cell formula belongs. This column is not supported by Analysis Services. It always contains VT_NULL.</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the cube to which the calculated cell formula belongs.</td>
</tr>
<tr>
<td>FORMULA_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the calculated cell formula, as specified in the CREATE CELL FORMULA statement.</td>
</tr>
<tr>
<td>SCOPE</td>
<td>DBTYPE_I4</td>
<td>The scope of the calculated cell formula. Only MDSET_SCOPE_SESSION is supported. The calculated cell formula lasts only as long as the current session is active.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>A user-friendly description of the calculated cell formula. This column is not supported.</td>
</tr>
</tbody>
</table>
This column is not supported by Analysis Services. It always contains VT_NULL.

<table>
<thead>
<tr>
<th>EXPERSSION</th>
<th>DBTYPE_WSTR</th>
<th>The Multidimensional Expressions (MDX) expression specified in the &lt;formula body&gt; clause of the CREATE CELL FORMULA statement.</th>
</tr>
</thead>
</table>

### Restriction Columns

<table>
<thead>
<tr>
<th>CATALOG_NAME</th>
<th>SCHEMA_NAME</th>
<th>CUBE_NAME</th>
<th>FORMULA_NAME</th>
<th>SCOPE</th>
</tr>
</thead>
</table>

### See Also

- [Calculated Cells](#)
- [CREATE CELL CALCULATION Statement](#)
Analysis Services Programming

**MDSHEMA_CUBES**

This schema rowset describes the structure of cubes that are contained in a database.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_DRILLTHROUGH_ENABLED</td>
<td>DBTYPE_BOOL</td>
<td>Describes whether DRILLTHROUGH can be performed on the members of a cube</td>
</tr>
<tr>
<td>IS_WRITE_ENABLED</td>
<td>DBTYPE_BOOL</td>
<td>Describes whether a cube is write-enabled</td>
</tr>
<tr>
<td>IS_LINKABLE</td>
<td>DBTYPE_BOOL</td>
<td>Describes whether a cube can be used in a linked cube</td>
</tr>
<tr>
<td>IS_SQL_ALLOWED</td>
<td>DBTYPE_BOOL</td>
<td>Describes whether or not SQL can be used on the cube</td>
</tr>
</tbody>
</table>

In Analysis Services, the CUBE_TYPE column can contain one the following string values: "CUBE", "VIRTUAL CUBE", or "LINKED CUBE". The value of the column depends on the type of cube the row is describing.

For local cubes, this column contains "CUBE".

For more information about the variety of cubes supported by Analysis Services, see [Introduction to Cubes](#).

**Restriction Columns**

CATALOG_NAME
See Also

Regular Cubes
Virtual Cubes
Linked Cubes
Local Cubes
MDSCHEMA_DIMENSIONS

This schema rowset describes the shared and private dimensions that are contained within a database.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_READWRITE</td>
<td>DBTYPE_BOOL</td>
<td>Contains TRUE if the dimension is write-enabled.</td>
</tr>
<tr>
<td>DIMENSION_UNIQUE_SETTINGS</td>
<td>DBTYPE_I4</td>
<td>If the dimension contains members with unique names or keys, this column contains a bitmap that specifies which columns contain unique values. The following bit value constants are defined in Msmd.h for this bitmap:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDDIMENSIONS_MEMBER_KEY_UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDDIMENSIONS_MEMBER_NAME_UNIQUE</td>
</tr>
<tr>
<td>DIMENSION_MASTER_UNIQUE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>If the value in this column in the schema rowset is set to TRUE (that is, if the dimension is virtual), this column contains the dimension on which that virtual dimension is based.</td>
</tr>
<tr>
<td>DIMENSION_IS_VISIBLE</td>
<td>DBTYPE_BOOL</td>
<td>Contains TRUE if the dimension is visible.</td>
</tr>
</tbody>
</table>

The meaning of the following column has changed since SQL Server version 7.0 OLAP Services.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_HIERARCHY</td>
<td>DBTYPE_WSTR</td>
<td>Contains the unique name of the hierarchy regardless of the number of hierarchies in the dimension. In earlier releases, this column contained VT_NULL if the</td>
</tr>
</tbody>
</table>
dimension had only one hierarchy.

**Restriction Columns**

CATALOG_NAME  
SCHEMA_NAME  
CUBE_NAME  
DIMENSION_NAME  
DIMENSION_UNIQUE_NAME

For more information about the MDSHEMA_DIMENSIONS schema rowset, see the OLE DB documentation.
This schema rowset describes the functions that are available to client applications connected to the database.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the function.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>A user-friendly description of the function.</td>
</tr>
<tr>
<td>PARAMETER_LIST</td>
<td>DBTYPE_WSTR</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>RETURN_TYPE</td>
<td>DBTYPE_I4</td>
<td>The VARTYPE of the return data type of the function.</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>DBTYPE_I4</td>
<td>For Multidimensional Expressions (MDX) functions, returns MSMD_SCHEMA_FUNCTIONS_ORIGIN_MSSQL. For user-defined functions, returns MSMD_FUNCTIONS_ORIGIN_UDF.</td>
</tr>
<tr>
<td>INTERFACE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the interface for user-defined functions. The group name for the MDX functions.</td>
</tr>
<tr>
<td>LIBRARY_NAME</td>
<td>DBTYPE_WSTR</td>
<td>(Optional.) For user-defined functions, the type library. For MDX functions, returns VT_NULL.</td>
</tr>
<tr>
<td>DLL_NAME</td>
<td>DBTYPE_WSTR</td>
<td>(Optional.) For user-defined functions, contains the name of the .dll or .exe file in which the function is implemented. For MDX functions, returns VT_NULL.</td>
</tr>
<tr>
<td>HELP_FILE</td>
<td>DBTYPE_WSTR</td>
<td>(Optional.) Contains the name of the file that contains this function's documentation. For MDX functions, returns VT_NULL.</td>
</tr>
<tr>
<td>HELP_CONTEXT</td>
<td>DBTYPE_WSTR</td>
<td>(Optional.) Returns the Help context ID for this function.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>DBTYPE_WSTR</td>
<td>(Optional). The generic name of the object class to which the function applies. For example, the rowset function returns VT_NULL if the function applies to the &lt;Level_Name&gt;.Members function.</td>
</tr>
</tbody>
</table>
The default sort order for this schema rowset is ORIGIN, INTERFACE_NAME, and FUNCTION_NAME.

**Restriction Columns**

LIBRARY_NAME
INTERFACE_NAME
FUNCTION_NAME
ORIGIN
MDSCHEMA_HIERARCHIES

This schema rowset describes each hierarchy that is contained within a particular dimension.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>DBTYPE_I2</td>
<td>The type of hierarchy. It can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MD_STRUCTURE_FULLY_BALANCED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MD_STRUCTURE_RAGGED_BALANCED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MD_STRUCTURE_UNBALANCED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MD_STRUCTURE_NETWORK</td>
</tr>
<tr>
<td>IS_VIRTUAL</td>
<td>DBTYPE_BOOL</td>
<td>Returns TRUE if this hierarchy represents a virtual dimension.</td>
</tr>
<tr>
<td>IS_READWRITE</td>
<td>DBTYPE_BOOL</td>
<td>Returns TRUE if the hierarchy is enabled.</td>
</tr>
<tr>
<td>HIERARCHY_UNIQUE_SETTINGS</td>
<td>DBTYPE_I4</td>
<td>A bitmap that specifies which columns contain unique values, if the hierarchy only has members with unique names or keys. The following bit value constants are defined in Msmd.h for this bitmap:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MDDIMENSIONS_MEMBER_KEY_UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MDDIMENSIONS_MEMBER_NAME_UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- HIERARCHY_MASTER_UNIQUE_NAME</td>
</tr>
<tr>
<td>HIERARCHY_MASTER_UNIQUE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>If the value in this set to TRUE (that is, if the dimension contains the current hierarchy), this column contains the dimension on which the virtual dimension is based.</td>
</tr>
<tr>
<td>HIERARCHY_IS_VISIBLE</td>
<td>DBTYPE_BOOL</td>
<td>Returns TRUE if dimension is visible.</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>HIERARCHY_ORDINAL</td>
<td>DBTYPE_UI4</td>
<td>Returns the ordinal number of the hierarchy across all hierarchies of the cube.</td>
</tr>
<tr>
<td>DIMENSION_IS_SHARED</td>
<td>DBTYPE_BOOL</td>
<td>Returns TRUE if the parent dimension is shared.</td>
</tr>
</tbody>
</table>

**Restriction Columns**

CATALOG_NAME  
SCHEMA_NAME  
CUBE_NAME  
DIMENSION_UNIQUE_NAME  
HIERARCHY_NAME  
HIERARCHY_UNIQUE_NAME

For more information about the MDSHEMA_HIERARCHIES schema rowset, see the OLE DB documentation.
MDSCHEMA_LEVELS

This schema rowset describes each level that is contained within a particular hierarchy.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL_UNIQUE_SETTINGS</td>
<td>DBTYPE_I4</td>
<td>A bitmap that specifies which columns contain unique values, if the level only has member with unique names or keys. The following bit value constants are defined in Msmd.h for this bitmap:</td>
</tr>
<tr>
<td>LEVEL_IS_VISIBLE</td>
<td>DBTYPE_BOOL</td>
<td>Returns TRUE if the dimension is visible.</td>
</tr>
<tr>
<td>LEVEL_ORDERING_PROPERTY</td>
<td>DBTYPE_WSTR</td>
<td>If the level is sorted by member property, this column returns the name of that property.</td>
</tr>
<tr>
<td>LEVEL_DBTYPE</td>
<td>DBTYPE_I4</td>
<td>The DBType was used to build members for the level.</td>
</tr>
<tr>
<td>LEVEL_MASTER_UNIQUE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>For levels that are members of a virtual dimension but not (All) levels, specifies the unique name of the level.</td>
</tr>
<tr>
<td>LEVEL_NAME_SQL_COLUMN_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the column in the SQL query that corresponds to the level's name.</td>
</tr>
<tr>
<td>LEVEL_KEY_SQL_COLUMN_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the column in the SQL query that corresponds to the level's key.</td>
</tr>
<tr>
<td>LEVEL_UNIQUE_NAME_SQL_COLUMN_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the column in the SQL query that corresponds to the level's unique name.</td>
</tr>
</tbody>
</table>

Restriction Columns
CATALOG_NAME
SCHEMA_NAME
CUBE_NAME
DIMENSION_UNIQUE_NAME
HIERARCHY_UNIQUE_NAME
LEVEL_NAME
LEVEL_UNIQUE_NAME

For more information about the MDSHEMA_LEVELS schema rowset, see the OLE DB documentation.
**MDSCHEMA_MEASURES**

This schema rowset describes each measure contained within a cube.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEASURE_IS_VISIBLE</td>
<td>DBTYPE_BOOL</td>
<td>Returns TRUE if dimension is visible.</td>
</tr>
<tr>
<td>LEVELS_LIST</td>
<td>DBTYPE_WSTR</td>
<td>Returns a comma-delimited list of unique names of the levels that are used in this measure. This column can be used for writeback when the end user needs to find out which levels can be written to for a virtual...</td>
</tr>
</tbody>
</table>
cube. If the measure is calculated, this column returns VT_NULL.

| MEASURE_NAME_SQL_COLUMN_NAME | DBTYPE_WSTR | Returns the name of the column in the SQL query that corresponds to the measure's name. |

### Restriction Columns

- CATALOG_NAME
- SCHEMA_NAME
- CUBE_NAME
- MEASURE_NAME
- MEASURE_UNIQUE_NAME

For more information about the MDSHEMA_MEASURES schema rowset, see the OLE DB documentation.
MDSCHEMA_MEMBERS

This schema rowset describes the members contained in a database.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBER_KEY</td>
<td>DBTYPE_WSTR</td>
<td>Contains the key property for the member.</td>
</tr>
<tr>
<td>IS_PLACEHOLDERMEMBER</td>
<td>DBTYPE_BOOL</td>
<td>Indicates whether a member is a placeholder member for an empty position in a dimension hierarchy. It is valid only if the MDX Compatibility property has been set to 1.</td>
</tr>
<tr>
<td>IS_DATAMEMBER</td>
<td>DBTYPE_BOOL</td>
<td>Contains TRUE if the member is a data member.</td>
</tr>
</tbody>
</table>

Restriction Columns

CATALOG_NAME
SCHEMA_NAME
CUBE_NAMEDIMENSION_UNIQUE_NAME
HIERARCHY_UNIQUE_NAME
IS_EMPTYMEMBER
IS_DATAMEMBER
LEVEL_UNIQUE_NAME
LEVEL_NUMBER
MEMBER_NAME
MEMBER_UNIQUE_NAME
MEMBER_CAPTION
MEMBER_TYPE
Tree operator

For more information about the MDSHEMA_MEMBERS schema rowset, see the OLE DB documentation.
MDSCHEMA_PROPERTIES

This schema rowset describes the properties of members contained in a database.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following column to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPERTY_CONTENT_TYPE</td>
<td>DBTYPE_I2</td>
<td>Property type</td>
</tr>
</tbody>
</table>

Restriction Columns

CATALOG_NAME
SCHEMA_NAME
CUBE_NAME
DIMENSION_UNIQUE_NAME
HIERARCHY_UNIQUE_NAME
LEVEL_UNIQUE_NAME
MEMBER_UNIQUE_NAME
PROPERTY_NAME
PROPERTY_TYPE

For more information about the MDSCHEMA_PROPERTIES schema rowset, see the OLE DB documentation.
MDSCHEMA_SETS

This schema rowset describes any sets that are currently defined within a database, including session-scoped sets.

Microsoft® SQL Server™ 2000 Analysis Services extends the OLE DB specification with the addition of the following columns to this schema rowset.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the catalog to which this set belongs. This column contains VT_NULL if the provider does not support catalogs.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>DBTYPE_WSTR</td>
<td>This column is not supported by Analysis Services. It always contains VT_NULL.</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the cube to which the set belongs. This column always contains a value and can never be VT_NULL.</td>
</tr>
<tr>
<td>SET_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the set, as specified in the CREATE SET statement.</td>
</tr>
<tr>
<td>SCOPE</td>
<td>DBTYPE_I4</td>
<td>The scope of the set. Only MDSET_SCOPE_SESSION is supported.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>This column is not supported by Analysis Services. It always contains VT_NULL.</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>DBTYPE_WSTR</td>
<td>The expression for this set.</td>
</tr>
<tr>
<td>DIMENSIONS</td>
<td>DBTYPE_WSTR</td>
<td>A comma-delimited list of dimensions used by the set.</td>
</tr>
</tbody>
</table>

The default sort order for this schema rowset is: CATALOG_NAME, SCHEMA_NAME, CUBE_NAME, SET_NAME, and SCOPE.
Restriction Columns

CATALOG_NAME
SCHEMA_NAME
CUBE_NAME
SET_NAME
SCOPE
Data Mining Schema Rowsets

The following table describes the schema rowsets that are used by Microsoft® SQL Server™ 2000 Analysis Services for browsing data mining models. Unlike the OLAP schema rowsets defined in previous topics, these rowsets are described in their entirety. For more information about these schema rowsets, see the OLE DB for Data Mining specification.

<table>
<thead>
<tr>
<th>Schema rowset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINING_COLUMNS</td>
<td>Describes the individual columns of all defined data mining models known to the provider</td>
</tr>
<tr>
<td>MINING_MODEL_CONTENT</td>
<td>Allows browsing of the content of a data mining model</td>
</tr>
<tr>
<td>MINING_MODEL_PMML</td>
<td>Stores the Predictive Model Markup Language (PMML) standard XML representation of the mining model</td>
</tr>
<tr>
<td>MINING_MODELS</td>
<td>Exposes <a href="#">data mining models</a></td>
</tr>
<tr>
<td>MINING_SERVICE_PARAMETERS</td>
<td>Provides a list of parameters that can be supplied when generating a mining model using the CREATE MINING MODEL statement</td>
</tr>
<tr>
<td>MINING_SERVICES</td>
<td>Provides a description of each data mining algorithm that is supported by that provider</td>
</tr>
</tbody>
</table>
MINING_COLUMNS

The individual columns in a data mining model are exposed in the MINING_COLUMNS schema rowset for every mining model in the database. Structurally, this rowset is similar to the COLUMNS schema rowset and can be used in the same manner. For example, if you provide a MODEL_NAME restriction, you can obtain all of the columns for a particular model.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name. Microsoft® SQL Server™ 2000 Analysis Services populates this column with the name of the database that the model is a member of.</td>
</tr>
<tr>
<td>MODEL_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>The unqualified schema name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>MODEL_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The mining model name. This column contains the name of the mining model with which a column is associated, and it is never empty.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>COLUMN_GUID</td>
<td>DBTYPE_GUID</td>
<td>The column GUID. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>COLUMN_PROPID</td>
<td>DBTYPE_UI4</td>
<td>The column property ID.</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>DBTYPE_UI4</td>
<td>The ordinal position. Columns are numbered starting from 1. This column contains VT_NULL if there is no stable ordinal.</td>
</tr>
<tr>
<td>COLUMN_HASDEFAULT</td>
<td>DBTYPE_BOOL</td>
<td>Contains VARIANT_TRUE if the column has a default value, otherwise False.</td>
</tr>
<tr>
<td></td>
<td>DBTYPE</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COLUMN_DEFAULT</td>
<td>DBTYPE_WSTR</td>
<td>The default value of the column. If the default value is the NULL value,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COLUMN_HASDEFAULT contains VARIANT_TRUE, and this column contains VT_NULL.</td>
</tr>
<tr>
<td>COLUMN_FLAGS</td>
<td>DBTYPE_UI4</td>
<td>A bitmask that describes characteristics of the column. The DBCOLUMNFLAGS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enumerated type specifies the bits in the bitmask. This column is never</td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty.</td>
</tr>
<tr>
<td>IS_NULLABLE</td>
<td>DBTYPE_BOOL</td>
<td>Contains VARIANT_FALSE if the column is known not to be nullable, otherwise</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIANT_TRUE.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>DBTYPE_UI2</td>
<td>The indicator of the column's data type, for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;TABLE&quot; = DBTYPE_HCHAPTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;TEXT&quot; = DBTYPE_WCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;LONG&quot; = DBTYPE_I8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;DOUBLE&quot; = DBTYPE_R8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;DATE&quot; = DBTYPE_DATE</td>
</tr>
<tr>
<td>TYPE_GUID</td>
<td>DBTYPE_GUID</td>
<td>The GUID of the column's data type. This column is not supported by Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Services; it always contains an empty GUID.</td>
</tr>
</tbody>
</table>
| CHARACTER_MAXIMUM_LENGTH | DBTYPE_UI4 | The maximum possible length in the column. For character, binary, or bit columns, this is on
- The maximum length of the column in characters, bytes, or bits, respective to the column type, if a length is defined. For example, a `CHAR(5)` column in an SQL table has a length of 5.

- The maximum length of the data type in characters, bytes, or bits, respective to the column type, if the column does not have a defined length.

- 0 if neither the column nor the data type has a defined length.

- NULL for all other types of columns.

<table>
<thead>
<tr>
<th>CHARACTER_OCTET_LENGTH</th>
<th>DBTYPE_UI4</th>
<th>The maximum length in octets (bytes) of the column, if the type of the column is character or binary. If the column has no maximum length, the column contains Vague types of columns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC_PRECISION</td>
<td>DBTYPE_UI2</td>
<td>If the column's data type other than DBTYPE_VARCHAR or DBTYPE_NUMERIC, column contains the maximum precision of the column. The column contains the maximum precision of the column definition. If the column's data type is DBTYPE_VARCHAR or DBTYPE_NUMERIC, column contains the maximum precision of the column definition.</td>
</tr>
<tr>
<td>Column Name</td>
<td>DB Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUMERIC_SCALE</td>
<td>DBTYPE_I2</td>
<td>If the column's type indicator is DBTYPE_DECIMAL, DBTYPE_NUMERIC, or DBTYPE_VARNUMERIC, this column contains the number of digits to the right of the decimal point. Otherwise, it contains VT_NULL.</td>
</tr>
<tr>
<td>DATETIME_PRECISION</td>
<td>DBTYPE_UI4</td>
<td>The date/time precision (number of digits in the fractional seconds portion) of the column if the column data type is a datetime or interval type. Otherwise, it contains VT_NULL.</td>
</tr>
<tr>
<td>CHARACTER_SET_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name in which the character set is defined. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>CHARACTER_SET_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>An unqualified schema name in which the character set is defined. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>CHARACTER_SET_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The character set name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>COLLATION_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name in which the collation is defined. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>COLLATION_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>An unqualified schema name in which the collation is defined. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>COLLATION_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The collation name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOMAIN_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name in which the domain is defined. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>DOMAIN_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>The unqualified schema name in which the domain is defined. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>DOMAIN_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The domain name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>A user-friendly description. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>DISTRIBUTION_FLAG</td>
<td>DBTYPE_WSTR</td>
<td>A description of the statistical distribution of the column. This column contains one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;NORMAL&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;LOG NORMAL&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;UNIFORM&quot;</td>
</tr>
<tr>
<td>CONTENT_TYPE</td>
<td>DBTYPE_WSTR</td>
<td>A description of the content of the column. This column contains one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;KEY&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;DISCRETE&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;CONTINUOUS&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;DISCRETED([arguments])&quot;</td>
</tr>
</tbody>
</table>
Provider-specific flags can also be defined.

<table>
<thead>
<tr>
<th>MODELING_FLAG</th>
<th>DBTYPE_WSTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A comma-delimited list of defined flags are:</td>
<td></td>
</tr>
<tr>
<td>• &quot;MODEL_EXISTENCE_ONLY&quot;</td>
<td></td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>IS_RELATED_TO_KEY</td>
<td>DBTYPE_BOOL</td>
</tr>
<tr>
<td>RELATED_ATTRIBUTE</td>
<td>DBTYPE_WSTR</td>
</tr>
<tr>
<td>IS_INPUT</td>
<td>DBTYPE_BOOL</td>
</tr>
<tr>
<td>IS_PREDICTABLE</td>
<td>DBTYPE_BOOL</td>
</tr>
<tr>
<td>CONTAINING_COLUMN</td>
<td>DBTYPE_WSTR</td>
</tr>
<tr>
<td>PREDICTION_SCALAR_FUNCTIONS</td>
<td>DBTYPE_WSTR</td>
</tr>
</tbody>
</table>
| PREDICTION_TABLE_FUNCTIONS        | DBTYPE_WSTR       | A comma-delimited list of functions that can be applied to the column. The functions should return a table. The list has the following format:  
  <function name>(<column1>, <column2>, ...)
  The format allows the client application to determine which columns will be present in the table the function returns. |
<table>
<thead>
<tr>
<th><strong>IS_POPULATED</strong></th>
<th>DBTYPE_BOOL</th>
<th>Contains TRUE if trained with a set of possible values. Contains FALSE if not populated.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PREDICTION_SCORE</strong></td>
<td>DBTYPE_UI4</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

**Default Sort Order**

- MODEL_CATALOG
- MODEL_SCHEMA
- MODEL_NAME
- COLUMN_NAME

**Restriction Columns**

- MODEL_CATALOG
- MODEL_SCHEMA
- MODEL_NAME
- COLUMN_NAME
This schema rowset allows the client application to browse the content of a data mining model. Client applications can use the special tree operation restrictions described at the end of this topic to navigate the content of the mining model.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name. Microsoft® SQL Server™ 2000 Analysis Services populates this column with the name of the database of which the model is a member.</td>
</tr>
<tr>
<td>MODEL_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>The unqualified schema name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>MODEL_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the model with which the content described by this row is associated.</td>
</tr>
<tr>
<td>ATTRIBUTE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name(s) of the attribute(s) corresponding to this node. For a model node, this is a list of predictable attributes. For a leaf distribution node, this is an attribute to which the distribution corresponds.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the node. Currently, this column contains the same value as NODE_UNIQUE_NAME, though this may change in future releases.</td>
</tr>
<tr>
<td>NODE_UNIQUE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The unique name of the node.</td>
</tr>
<tr>
<td>NODE_TYPE</td>
<td>DBTYPE_I4</td>
<td>The type of the node. It can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DM_NODE_TYPE_MODEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DM_NODE_TYPE_TREE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DM_NODE_TYPE_INTERIOR</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NODE_GUID</td>
<td>DBTYPE_GUID</td>
<td>The node GUID. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>NODE_CAPTION</td>
<td>DBTYPE_WSTR</td>
<td>A label or a caption associated with the node. This property is used primarily for display purposes. If a caption does not exist, the contents of the NODE_NAME column is returned.</td>
</tr>
<tr>
<td>CHILDREN_CARDINALITY</td>
<td>DBTYPE_UI4</td>
<td>An estimate of the number of children that the node has.</td>
</tr>
<tr>
<td>PARENT_UNIQUE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The unique name of the node's parent. VT_NULL is returned for any nodes at the root level.</td>
</tr>
<tr>
<td>NODE_DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>A user-friendly description of the node.</td>
</tr>
<tr>
<td>NODE_RULE</td>
<td>DBTYPE_WSTR</td>
<td>An XML description of the rule that is embedded in the node.</td>
</tr>
<tr>
<td>MARGINAL_RULE</td>
<td>DBTYPE_WSTR</td>
<td>An XML description of the rule that is moving to the node from the parent node.</td>
</tr>
<tr>
<td>NODE_PROBABILITY</td>
<td>DBTYPE_R8</td>
<td>The probability associated with this node.</td>
</tr>
<tr>
<td>MARGINAL_PROBABILITY</td>
<td>DBTYPE_R8</td>
<td>The probability of reaching the node from the parent node.</td>
</tr>
<tr>
<td>NODE_DISTRIBUTION</td>
<td>DBTYPE_HCHAPTER</td>
<td>A table that contains the probability histogram of the node.</td>
</tr>
<tr>
<td>NODE_SUPPORT</td>
<td>DBTYPE_R8</td>
<td>The number of cases that support this node.</td>
</tr>
<tr>
<td>MSOLAP_MODEL_COLUMN</td>
<td>DBTYPE_WSTR</td>
<td>The name of the column from the model definition that this node pertains to.</td>
</tr>
<tr>
<td>MSOLAP_NODE_SCORE</td>
<td>DBTYPE_R8</td>
<td>The score that was computed for this node.</td>
</tr>
</tbody>
</table>
**Default Sort Order**

MODEL_CATALOG
MODEL_SCHEMA
MODEL_NAME
ATTRIBUTE_NAME

**Restriction Columns**

The MINING_MODEL_CONTENT schema rowset can have ten restrictions. The first nine are columns in the rowset described in the table.

MODEL_CATALOG
MODEL_SCHEMA
MODEL_NAME
ATTRIBUTE_NAME
NODE_NAME
NODE_UNIQUE_NAME
NODE_TYPE
NODE_GUID
NODE_CAPTION

The tenth restriction, TREE_OPERATION, is not on any particular column of the MINING_MODEL_CONTENT rowset; rather, it specifies a tree operator. The consumer can specify a NODE_UNIQUE_NAME restriction and the tree operator (ANCESTORS, CHILDREN, SIBLINGS, PARENT, DESCENDANTS, SELF) to obtain the requested set of members. The SELF operator includes the row for the node itself in the list of returned rows. The following table describes the constants that make up the bitmap definition for the TREE_OPERATION restriction. They can be combined using the logical OR operator.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMTREEOP_ANCESTORS</td>
<td>0x00000020</td>
</tr>
<tr>
<td>DMTREEOP_CHILDREN</td>
<td>0x00000001</td>
</tr>
<tr>
<td>DM_TREEOP_NAME</td>
<td>DWORD</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DM_TREEOP_SIBLINGS</td>
<td>0x00000002</td>
</tr>
<tr>
<td>DM_TREEOP_PARENT</td>
<td>0x00000004</td>
</tr>
<tr>
<td>DM_TREEOP_SELF</td>
<td>0x00000008</td>
</tr>
<tr>
<td>DM_TREEOP_DESCENDANTS</td>
<td>0x00000010</td>
</tr>
</tbody>
</table>

**See Also**

[Data Mining Columns](#)
MINING_MODEL_CONTENT_PMML

This schema rowset stores the Extensible Markup Language (XML) structure of the mining model. The format of the XML string follows the Predictive Model Markup Language (PMML) standard.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name. Microsoft® SQL Server™ 2000 Analysis Services populates this column with the name of the database of which the model is a member.</td>
</tr>
<tr>
<td>MODEL_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>The unqualified schema name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>MODEL_NAME</td>
<td>DBTYPE_WSTR</td>
<td>Model name. This column cannot contain VT_NULL.</td>
</tr>
<tr>
<td>MODEL_TYPE</td>
<td>DBTYPE_WSTR</td>
<td>The model type. It is a provider-specific string. It can be VT_NULL.</td>
</tr>
<tr>
<td>MODEL_GUID</td>
<td>DBTYPE_GUID</td>
<td>The GUID that identifies the model. Providers that do not use GUIDs to identify tables return VT_NULL.</td>
</tr>
<tr>
<td>MODEL_PMML</td>
<td>DBTYPE_WSTR</td>
<td>An XML representation of the model's content in PMML format.</td>
</tr>
<tr>
<td>SIZE</td>
<td>DMTYPE_UI4</td>
<td>Number of bytes in the</td>
</tr>
</tbody>
</table>
The location of the XML file. It is VT_NULL if the file is stored in the default directory.

**Default Sort Order**

MODEL_CATALOG
MODEL_SCHEMA
MODEL_NAME

**Restriction Columns**

MODEL_CATALOG
MODEL_SCHEMA
MODEL_NAME
MODEL_TYPE
MINING_MODELS

Data mining models are exposed in the MINING_MODELS schema rowset. This schema rowset is very similar to the TABLES schema rowset and can be used the same way. Use this rowset to obtain information about the mining models contained within a database. This rowset can include information such as the names, types and mining algorithms associated with each mining model.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_CATALOG</td>
<td>DBTYPE_WSTR</td>
<td>The catalog name. Microsoft® SQL Server™ Analysis Services populates this column with the name of the database of which the model is a member.</td>
</tr>
<tr>
<td>MODEL_SCHEMA</td>
<td>DBTYPE_WSTR</td>
<td>The unqualified schema name. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>MODEL_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The mining model name. This column contains the name of the mining model, and it is never empty.</td>
</tr>
<tr>
<td>MODEL_TYPE</td>
<td>DBTYPE_WSTR</td>
<td>The model type. This value is set to &quot;OLAP&quot; if the mining model is an OLAP model and the model is relational.</td>
</tr>
<tr>
<td>MODEL_GUID</td>
<td>DBTYPE_GUID</td>
<td>The GUID of the model.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>A user-friendly description of the model. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>MODEL_PROPID</td>
<td>DBTYPE_UI4</td>
<td>The property ID of the model. This column is not supported by Analysis Services; it always contains VT_NULL.</td>
</tr>
<tr>
<td>DATE_CREATED</td>
<td>DBTYPE_DATE</td>
<td>The date on which the model was created.</td>
</tr>
<tr>
<td>DATE_MODIFIED</td>
<td>DBTYPE_DATE</td>
<td>The date on which the model definition was last modified.</td>
</tr>
<tr>
<td>SERVICE_TYPE_ID</td>
<td>DBTYPE_UI4</td>
<td>Contains an enumerated type that identifies the data mining algorithm used by the model. This type may be one of the following:</td>
</tr>
</tbody>
</table>

**DM_SERVICETYPE**
## Data Mining Models

### Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>A string that contains the provider-specific name for the data mining algorithm used by the model.</td>
</tr>
<tr>
<td>CREATION_STATEMENT</td>
<td>DBTYPE_WSTR</td>
<td>Contains a string that contains the statement used to create the mining model.</td>
</tr>
<tr>
<td>PREDICTION_ENTITY</td>
<td>DBTYPE_WSTR</td>
<td>A string that contains a comma-delimited list indicating which mining columns can be predicted.</td>
</tr>
<tr>
<td>IS_POPULATED</td>
<td>DBTYPE_BOOL</td>
<td>Contains <strong>VARIANT_TRUE</strong> if the model is populated. Otherwise, it contains <strong>VARIANT_FALSE</strong>.</td>
</tr>
<tr>
<td>MSOLAP_MODEL_SOURCE</td>
<td>DBTYPE_WSTR</td>
<td>For OLAP mining models, this column contains the name of the cube on which the model is based.</td>
</tr>
</tbody>
</table>

### Default Sort Order

```
MODEL_CATALOG
MODEL_SCHEMA
MODEL_NAME
MODEL_TYPE
SERVICE_NAME
SERVICE_TYPE_ID
```

### Restrictions

```
MODEL_CATALOG
MODEL_SCHEMA
MODEL_NAME
MODEL_TYPE
SERVICE_NAME
SERVICE_TYPE_ID
```

### See Also

[Data Mining Models](#)
MINING_SERVICE_PARAMETERS

This schema rowset provides a list of parameters that can be supplied when you are generating a mining model using the CREATE MINING MODEL statement. The client application will often restrict by SERVICE_NAME to obtain the parameters that are supported by the provider and are applicable to the type of mining model being generated.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the algorithm.</td>
</tr>
<tr>
<td>PARAMETER_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>PARAMETER_TYPE</td>
<td>DBTYPE_WSTR</td>
<td>The OLE DB data type of the parameter.</td>
</tr>
<tr>
<td>IS_REQUIRED</td>
<td>DBTYPE_BOOL</td>
<td>TRUE if the parameter is required.</td>
</tr>
<tr>
<td>PARAMETER_FLAGS</td>
<td>DBTYPE_UI4</td>
<td>A bitmap that describes parameter characteristics. The following bit value constants are defined in Msmd.h for this bitmap:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DM_PARAMETER_TRAINING (0x00000001) For training</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DM_PARAMETER_PREDICTION (0x00000002) For prediction</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>Text that describes the purpose and format of the parameter.</td>
</tr>
</tbody>
</table>

Default Sort Order

SERVICE_NAME
PARAMETER_NAME
Restriction Columns

SERVICE_NAME
PARAMETER_NAME
## MINING_SERVICES

This schema rowset provides a description of each data mining algorithm the provider supports.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_NAME</td>
<td>DBTYPE_WSTR</td>
<td>The name of the algorithm. This column is provider-specific.</td>
</tr>
<tr>
<td>SERVICE_TYPE_ID</td>
<td>DBTYPE_UI4</td>
<td>This column contains a bitmap that describes the mining service. Microsoft® SQL Server™ 2000 Analysis Services populates this column with one of the following values:</td>
</tr>
<tr>
<td>SERVICE_DISPLAY_NAME</td>
<td>DBTYPE_WSTR</td>
<td>A localizable display name for the algorithm.</td>
</tr>
<tr>
<td>SERVICE_GUID</td>
<td>DBTYPE_GUID</td>
<td>The GUID for the algorithm.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DBTYPE_WSTR</td>
<td>A user-friendly description of the algorithm.</td>
</tr>
<tr>
<td>PREDICTION_LIMIT</td>
<td>DBTYPE_UI4</td>
<td>The maximum number of predictions the model and algorithm can provide.</td>
</tr>
<tr>
<td>SUPPORTED_DISTRIBUTION_FLAGS</td>
<td>DBTYPE_WSTR</td>
<td>A comma-delimited list of flags that describe the statistical distributions supported by the algorithm. This column contains one or more of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Provider-specific flags can also be defined.</td>
</tr>
<tr>
<td>SUPPORTED_INPUT_CONTENT_TYPES</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>A comma-delimited list of flags that describe the input content types that are supported by the algorithm. This column contains one or more of the following values:</td>
<td>A comma-delimited list of flags that describe the input content types that are supported by the algorithm. This column contains one or more of the following values:</td>
<td></td>
</tr>
<tr>
<td>SUPPORTED_PREDICTION_CONTENT_TYPES</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>A comma-delimited list of flags that describe the prediction content types that are supported by the algorithm. This column contains one or more of the following values:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| SUPPORTED_MODELING_FLAGS       | DBTYPE_WSTR | A comma-delimited list of the modeling flags that are supported by the algorithm. This column contains one or more of the following values:
| SUPPORTED_SOURCE_QUERY        | DBTYPE_WSTR | The `<source_data_query>` types that the provider supports. This is a comma-delimited list of one or more of the following syntax descriptions. These descriptions can be used as the source of data for `INSERT INTO`, or can be joined using a `PREDICTION JOIN` to a mining model for `SELECT`. The following values are available:
<p>| TRAINING_COMPLEXITY           | DBTYPE_I4   | Indicates the length of time training is expected to take: |</p>
<table>
<thead>
<tr>
<th>Feature</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREDICTION_COMPLEXITY</td>
<td>DBTYPE_I4</td>
<td>Indicates the length of time prediction is expected to take.</td>
</tr>
<tr>
<td>EXPECTED_QUALITY</td>
<td>DBTYPE_I4</td>
<td>Indicates the expected quality of the model produced with this algorithm.</td>
</tr>
<tr>
<td>SCALING</td>
<td>DBTYPE_I4</td>
<td>Indicates the scalability of the algorithm.</td>
</tr>
<tr>
<td>Feature</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>ALLOW_INCREMENTAL_INSERT</td>
<td>DBTYPE_BOOL</td>
<td>Contains <code>VARIANT_TRUE</code> if additional <code>INSERT</code> statements are allowed after the initial training.</td>
</tr>
<tr>
<td>ALLOW_PMML_INITIALIZATION</td>
<td>DBTYPE_BOOL</td>
<td>Contains <code>VARIANT_TRUE</code> if the mining models (including structure and content) can be created based on an XML string.</td>
</tr>
<tr>
<td>CONTROL</td>
<td>DBTYPE_I4</td>
<td>Contains one of the following values, which determine whether the service supports training interruption.</td>
</tr>
<tr>
<td>ALLOW_DUPLICATE_KEY</td>
<td>DBTYPE_BOOL</td>
<td>Contains <code>VARIANT_TRUE</code> if cases are allowed to contain duplicate keys.</td>
</tr>
</tbody>
</table>

**Default Sort Order**

SERVICE_NAME

**Restriction Columns**
SERVICE_NAME
SERVICE_TYPE_ID
Analysis Services Programming
Analysis Services Programming Samples

The following samples illustrate Microsoft® SQL Server™ 2000 Analysis Services application development in Microsoft Visual Basic® Scripting Edition (VBScript), Microsoft Visual C++®, and Visual Basic.

Each sample demonstrates a different technique for working with cube data. Prerequisites for viewing or running each sample vary depending on the development tool and methodology used to create the code.

Installing Sample Files

The sample applications are installed with the optional Samples component of Analysis Services. They are located in the Samples folder under Microsoft Analysis Services (installed by default to C:\Program Files\Microsoft Analysis Services\Samples). Each sample program is contained in a subfolder at this location.

General Requirements

Many samples require a SQL Server database that provides cube data. For testing purposes, you can use the cubes in the **FoodMart 2000** database. You can modify sample source files to point to this database or to specific cubes.

Sample Scripts and Programs

The following table lists and describes the categories of samples you can work with. Sample source code has comments to help you learn how the application works.

<table>
<thead>
<tr>
<th>Sample category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Cube Creation</td>
<td>A set of samples that demonstrate how to create cubes from client and server applications.</td>
</tr>
<tr>
<td>Cube Query and Result Set Manipulation</td>
<td>A set of samples that demonstrate how to query a cube and manipulate the result set. Samples are provided for both client and server applications.</td>
</tr>
<tr>
<td>Cube Schema</td>
<td>A set of samples that demonstrate how to obtain</td>
</tr>
<tr>
<td><strong>Retrieval and Manipulation</strong></td>
<td>Cube schema data and manipulate the data. Samples are provided for both client and server applications.</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Complex Cube Creation and Manipulation</strong></td>
<td>A three-part integrated sample that demonstrates how to create a cube, write-enable the cube, and write back to the cube. In addition, a different sample shows how to retrieve cube data and schema information and then manipulate the results using dynamic HTML (DHTML).</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Simple Cube Creation

Sample code for creating a cube is provided in two different samples. Each sample illustrates a different implementation, depending on whether the cube is created by a client or a server application.

Sample Programs

The following table lists and describes the samples in this section. For more information about installation and general requirements, see Analysis Services Programming Samples.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VbAdoCreateCube</td>
<td>This Microsoft® Visual Basic® project creates a client-side cube using Microsoft ActiveX® Data Objects (ADO) and ActiveX Data Objects (Multidimensional) (ADO MD). This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, ADO, and ADO MD. This sample is located in the VbAdoCreateCube folder. It consists of the FrmVbAdoCreateCube.frm, VbAdoCreateCube.vbp, and VbAdoCreateCube.vbw files.</td>
</tr>
<tr>
<td>VbDSOExample</td>
<td>This Visual Basic project creates a server-side cube and demonstrates much of the available Decision Support Objects (DSO) functionality. This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, and DSO. This sample is located in the VbDSOExample folder. It consists of the FrmMain.frm, AdvancedSampleCode.bas, Writeback.bas, DSOSample.vbp, and DSOSample.vbw files.</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Cube Query and Result Set Manipulation

Sample code for creating and manipulating a query-based cube is provided in five different samples. Each sample illustrates a different implementation, depending on the development tool and whether the cube is created by a client application or a server application.

Sample Programs

The following table provides the names of and details about the samples in this section. For more information about installation and general requirements, see Analysis Services Programming Samples.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
</table>
| AspAdoSimple    | This Microsoft® Visual Basic® Scripting Edition (VBScript) sample executes a Multidimensional Expressions (MDX) query using the Sales cube on the local computer and displays the results in a simple table format. The sample demonstrates the basic steps for querying a database and displaying the results.  
This sample requires Microsoft Internet Information Server 4.0 or Microsoft Internet Information Services (IIS) 5.0 or later, ADO, and ADO MD.  
This sample is located in the AspAdoSimple folder. It consists of the AspAdoSimple.asp file. |
| AspAdoComplex   | This VBScript sample executes an MDX query using the server, database, and cube the user specifies in fields of the form provided by an Active Server Pages (ASP) page. The sample uses the HTML COLSPAN attribute to present the resulting cellset as a table on the same HTML pane as the form.  
This sample requires IIS 4.0 or later, Microsoft ActiveX® Data Objects (ADO), and ADO Multidimensional (ADO MD). |
<table>
<thead>
<tr>
<th>Sample Name</th>
<th>Description</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>AspAdoComplex</td>
<td>This sample is located in the AspAdoComplex folder. It consists of the AspAdoComplex.asp file.</td>
<td></td>
</tr>
<tr>
<td>VbAdoSimple</td>
<td>This Visual Basic project executes a query using ADO MD. The sample displays the result set in the Immediate window.</td>
<td>This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, and ADO MD.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This sample is located in the VbAdoSimple folder. It consists of the FrmVbAdoSimple.frm, VbAdoSimple.vbp, and VbAdoSimple.vbw files.</td>
</tr>
<tr>
<td>VbAdoComplex</td>
<td>This VBScript sample executes an MDX query. The sample displays the resulting cellset in a Microsoft Excel spreadsheet.</td>
<td>This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, ADO, and Excel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This sample is located in the VbAdoComplex folder. It consists of the FrmVbAdoComplex.frm, VbAdoComplex.vbp, and VbAdoComplex.vbw files.</td>
</tr>
<tr>
<td>CppOlapDemo</td>
<td>This Microsoft Visual C++® project creates a server-side cube based on a query, using OLE DB for OLAP to connect to Microsoft SQL Server™ 2000 Analysis Services.</td>
<td>This sample requires Visual C++ 5.0 or later, Msmd.h (which is located in the C:\Program Files\Microsoft Analysis Services\Samples\Include folder), and the Microsoft Data Access Software Development Kit (SDK) version 2.1 or later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This sample is located in the CppOlapDemo folder. It consists of the OLAPApp.cpp, OLAPDemo.cpp, OLAPTab.cpp, OLAPDemo.dsp, OLAPApp.hpp, and OLAPTab.hpp files.</td>
</tr>
</tbody>
</table>
Analysis Services Programming
Cube Schema Retrieval and Manipulation

Sample code for retrieving and manipulating a cube schema is provided in three different samples. Each sample illustrates a different implementation, depending on the development tool and whether the cube is created by a client application or a server application.

Sample Programs

The following table provides the names of and details about the samples in this section. For more information about installation and general requirements, see Analysis Services Programming Samples.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AspAdoCubeDoc</td>
<td>This Microsoft® Visual Basic® Scripting Edition (VBScript) sample retrieves a cube schema and displays the data on a Web page. This sample requires Microsoft Internet Information Server 4.0 or Microsoft Internet Information Services (IIS) 5.0 or later, Microsoft ActiveX® Data Objects (ADO) and ADO (Multidimensional) (ADO MD). This sample is located in the AspAdoCubeDoc folder. It consists of the AspAdoCubeDoc.asp file.</td>
</tr>
<tr>
<td>AspAdoCubeTree</td>
<td>This VBScript and JScript client application retrieves a cube schema and displays the data on a Web page using dynamic HTML (DHTML). This sample requires IIS 4.0 or later, ADO, and ADO MD. This sample is located in the AspAdoCubeTree folder. It consists of AspAdoCubeTree.asp, DimensionDrop.js, and fourteen image files.</td>
</tr>
<tr>
<td>VbAdoCubeDoc</td>
<td>This Visual Basic project retrieves cube schema information and stores it in a Microsoft Word</td>
</tr>
</tbody>
</table>
This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, ADO MD, and Microsoft Excel.

This sample is located in the VbAdoCubeDoc folder. It consists of the FrmVbAdoCubeDoc.frm, VbAdoCubeDoc.vbp, VbAdoCubeDoc.vbw files.
Analysis Services Programming
Complex Cube Creation and Manipulation

Sample code for creating and manipulating a complex cube is provided in a three-part integrated sample and a stand-alone sample that demonstrates manipulation in dynamic HTML (DHTML).

Sample Programs

The following table provides the names of and details about the samples in this section. For more information about installation and general requirements, see Analysis Services Programming Samples.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VbDsoCreateSmallCube (part 1 of 3)</td>
<td>This Microsoft® Visual Basic® project uses Decision Support Objects (DSO) to create a server-side cube that can be used in other applications (specifically, VbAdoWriteback). This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, and Decision Support Objects (DSO). This sample is located in the VbDsoCreateSmallCube folder. It consists of the FrmMain.frm and DsoMakeCube.vbp files.</td>
</tr>
<tr>
<td>VbDsoWriteEnableCube (part 2 of 3)</td>
<td>This Visual Basic project demonstrates the steps involved in creating a writeback partition table and write-enabling a cube. This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, and DSO. It also requires that you run VbDsoCreateSmallCube before running this sample. This sample is located in the VbDsoWriteEnableCube folder. It consists of the</td>
</tr>
</tbody>
</table>
| VbAdoWriteBack (part 3 of 3) | This Visual Basic project populates a client-side cube using the writeback cube defined by VbDsoCreateSmallCube.  

This sample requires Visual Basic 5.0 (with Service Pack 3) or Visual Basic 6.0, ADO, and ADO MD. You must run VbDsoCreateSmallCube and VbDsoWriteEnableCube before you run this sample.  

This sample is located in the VbAdoWriteBack folder. It consists of the FrmMain.frx, FrmMain.frm, and SimpleWriteback.vbp files. |
|---|---|
| VbMdHTMLdll | This Visual Basic project creates MdHtml.dll, which displays cube data on a Web page.  

This sample requires Visual Basic 6.0 and Microsoft Internet Information Server 4.0 or Microsoft Internet Information Services (IIS) 5.0 or later.  

This sample is located in the VbMdHTMLdll folder. It consists of the MdHtmlDll.asp, MdHtmlDll.dll, MdHtmlDll.exp, MdHtmlDll.lib, Table.cls, MdHtmlDll.vbp, and MdHtmlDll.vbw files. |