

Multidimensional Model Programming

SQL Server 2012 Books Online

Reference



Microsoft

Multidimensional Model Programming

SQL Server 2012 Books Online

Summary: Analysis Services provides several APIs that you can use to program against an Analysis Services instance and the multidimensional databases that it makes available. This section describes the approaches available to developers who want to create custom applications using Analysis Services multidimensional solutions. You can use this information to choose the programming interface that best meets the requirements of a particular project. Analysis Services development projects can be based on managed or non-managed code that runs on a Windows platform, or other platforms that support HTTP access.

Category: Reference

Applies to: SQL Server 2012

Source: SQL Server 2012 Books Online ([link to source content](#))

E-book publication date: January 2013

Copyright © 2012 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft and the trademarks listed at <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Contents

Multidimensional Model Programming.....	4
Understanding Microsoft OLAP Architecture	4
Logical Architecture	5
Logical Architecture Overview	6
Server Objects	13
Database Objects.....	13
Security Roles	15
Dimension Objects	19
Dimensions.....	20
Attributes and Attribute Hierarchies.....	26
Attribute Relationships	27
User Hierarchies.....	29
Write-Enabled Dimensions.....	35
Dimension Translations	36
Database Dimension Properties.....	37
Proactive Caching (Dimensions).....	41
Cube Objects	41
Cube Properties	42
Dimension Relationships.....	44
Calculations	49
Partitions	50
Perspectives	59
Cube Translations	61
Cube Cells	62
Cube Storage.....	65
Aggregations and Aggregation Designs	66
Physical Architecture	68
OLAP Engine Server Components	68
Server Process	71
Object Naming.....	71
Maximum Capacity Specifications.....	74
Data Types in Analysis Services	78
Local Cubes	81
Clients.....	83
International Considerations	88
Languages and Collations	90
Translations.....	96
Currency Conversions	98
Client Applications	104
Developing with ADOMD.NET.....	105

ADOMD.NET Client Programming	106
ADOMD.NET Client Functionality	109
Migrating From ADO MD To ADOMD.NET.....	111
Establishing Connections in ADOMD.NET.....	119
Establishing Secure Connections in ADOMD.NET	120
Working with Connections and Sessions in ADOMD.NET	127
Performing Transactions in ADOMD.NET	130
Retrieving Metadata from an Analytical Data Source.....	131
Working with the ADOMD.NET Object Model	136
Working with Schema Rowsets in ADOMD.NET	137
Executing Commands Against an Analytical Data Source	139
Retrieving Data from an Analytical Data Source.....	140
Retrieving Data Using the CellSet.....	141
Retrieving Data Using the AdomdDataReader.....	143
Retrieving Data Using the XmlReader.....	147
ADOMD.NET Server Programming	148
ADOMD.NET Server Functionality	149
ADOMD.NET Server Object Architecture.....	150
User Defined Functions and Stored Procedures	154
Redistributing ADOMD.NET	156
Developing with Analysis Management Objects (AMO)	157
AMO Concepts and Object Model.....	157
Introducing AMO Classes	165
AMO Fundamental Classes	167
AMO OLAP Classes.....	169
AMO Data Mining Classes.....	174
AMO Security Classes.....	178
AMO Other Classes and Methods.....	180
Programming Administrative Tasks with AMO	185
Programming AMO Fundamental Objects.....	186
Programming AMO OLAP Basic Objects	202
Programming AMO OLAP Advanced Objects.....	218
Programming AMO Data Mining Objects	232
Programming AMO Security Objects.....	235
Programming AMO Complementary Classes and Methods	240
Developing with Analysis Services Scripting Language (ASSL)	247
ASSL Objects and Object Characteristics.....	247
ASSL XML Conventions	249
XMLA Concepts	253
Developing with XMLA in Analysis Services	254
Managing Connections and Sessions (XMLA)	257
Handling Errors and Warnings (XMLA).....	260
Defining and Identifying Objects (XMLA).....	264
Managing Transactions (XMLA)	265
Canceling Commands (XMLA)	266

Performing Batch Operations (XMLA).....	267
Creating and Altering Objects (XMLA).....	271
Locking and Unlocking Databases (XMLA)	274
Processing Objects (XMLA)	275
Merging Partitions (XMLA).....	280
Designing Aggregations (XMLA)	282
Backing Up, Restoring, and Synchronizing Databases (XMLA)	286
Inserting, Updating, and Dropping Members (XMLA).....	292
Updating Cells (XMLA).....	295
Managing Caches (XMLA)	296
Monitoring Traces (XMLA).....	297
Extending OLAP functionality	301
Extending OLAP through personalizations	301
Analysis Services Personalization Extensions	302
Defining Stored Procedures.....	306
Designing Stored Procedures	307
Creating Stored Procedures.....	307
Calling Stored Procedures	310
Accessing Query Context in Stored Procedures	312
Setting Security for Stored Procedures	312
Debugging Stored Procedures.....	312
Analysis Services OLE DB Provider	314

Multidimensional Model Programming

Analysis Services provides several APIs that you can use to program against an Analysis Services instance and the multidimensional databases that it makes available. This section describes the approaches available to developers who want to create custom applications using Analysis Services multidimensional solutions. You can use this information to choose the programming interface that best meets the requirements of a particular project. Analysis Services development projects can be based on managed or non-managed code that runs on a Windows platform, or other platforms that support HTTP access.

In This Section

[Understanding Microsoft OLAP Architecture](#)

[Developing with ADOMD.NET](#)

[Developing with Analysis Management Objects \(AMO\)](#)

[Developing with XMLA in Analysis Services](#)

[Developing with Analysis Services Scripting Language \(ASSL\)](#)

[Extending OLAP functionality](#)

[Analysis Services OLE DB Provider \(Analysis Services - Multidimensional Data\)](#)

See Also

[Tabular Model Programming](#)

[Data Mining Programming](#)

Understanding Microsoft OLAP Architecture

Use these topics to better understand Analysis Services multidimensional databases and plan how to implement multidimensional databases in your business intelligence solution.

Logical Architecture

[Solution Design Considerations \(Analysis Services - Multidimensional Data\)](#)

[Dimension Objects \(Analysis Services - Multidimensional Database\)](#)

[Cube Objects \(Analysis Services - Multidimensional Database\)](#)

[More...](#)

Physical Architecture

[Analysis Services Multidimensional Database Server Cubes](#)

[Analysis Services Multidimensional Database Local Cubes
More...](#)

Programming Architecture

[Developing with Analysis Management Objects \(AMO\)](#)
[Developing with Analysis Services Scripting Language \(ASSL\)](#)
[Developing with ADOMD.NET](#)

International Considerations

[International Considerations \(Analysis Services - Multidimensional Data\)](#)

See Also

[Technical Reference \(Analysis Services - Multidimensional Database\)](#)

Logical Architecture

Microsoft SQL Server Analysis Services uses both server and client components to supply online analytical processing (OLAP) and data mining functionality for business intelligence applications:

- The server component of Analysis Services is implemented as a Microsoft Windows service. SQL Server Analysis Services supports multiple instances on the same computer, with each instance of Analysis Services implemented as a separate instance of the Windows service.
- Clients communicate with Analysis Services using the public standard XML for Analysis (XMLA), a SOAP-based protocol for issuing commands and receiving responses, exposed as a Web service. Client object models are also provided over XMLA, and can be accessed either by using a managed provider, such as ADOMD.NET, or a native OLE DB provider.
- Query commands can be issued using the following languages: SQL; Multidimensional Expressions (MDX), an industry standard query language for analysis; or Data Mining Extensions (DMX), an industry standard query language oriented toward data mining. Analysis Services Scripting Language (ASSL) can also be used to manage Analysis Services database objects.

Analysis Services also supports a local cube engine that enables applications on disconnected clients to browse locally stored multidimensional data. For more information, see [Client Architecture](#)

In This Section

Logical Architecture Overview

[Logical Architecture Overview \(Analysis Services - Multidimensional Database\)](#)

Server Objects

[Server Objects \(Analysis Services - Multidimensional Database\)](#)

Dimension Objects

[Dimension Objects \(Analysis Services - Multidimensional Database\)](#)

Cube Objects

[Cube Objects \(Analysis Services - Multidimensional Database\)](#)

User Access Security

[User Access Security Architecture](#)

See Also

[Planning and Architecture \(Analysis Services - Multidimensional Database\)](#)

[Physical Architecture \(Analysis Services - Multidimensional Database\)](#)

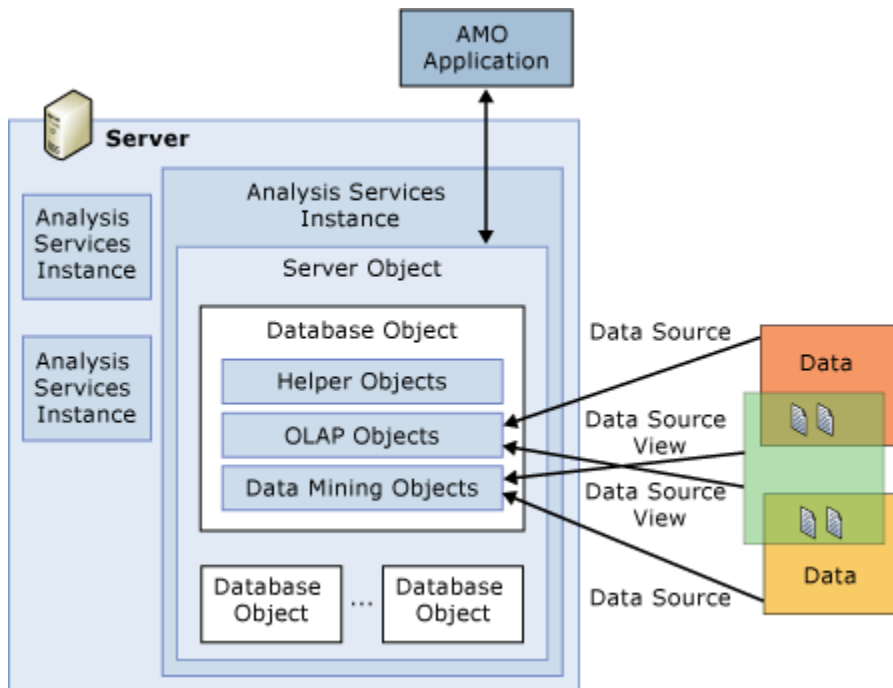
Logical Architecture Overview

In SQL Server 2008 R2, Analysis Services can be operated in two different modes: the standard server installation mode, which supports traditional OLAP and data mining, and SharePoint integrated mode, which uses a special instance of Analysis Services hosted in a SharePoint server to support workbooks created by Microsoft PowerPivot for Excel 2010.

This topic explains the basic architecture of Analysis Services when operating in standard mode. For more information about Sharepoint integrated mode, see [Analysis Services in Vertipaq mode](#). For more information about the PowerPivot client, see [PowerPivot for Excel](#).

Basic Architecture

An instance of Analysis Services can contain multiple databases, and a database can have OLAP objects and data mining objects at the same time. Applications connect to a specified instance of Analysis Services and a specified database. A server computer can host multiple instances of Analysis Services. Instances of Analysis Services are named as "<ServerName>\<InstanceName>". The following illustration shows all mentioned relationships between Analysis Services objects.



Basic classes are the minimum set of objects that are required to build a cube. This minimum set of objects is a dimension, a measure group, and a partition. An aggregation is optional.

Dimensions are built from attributes and hierarchies. Hierarchies are formed by an ordered set of attributes, where each attribute of the set corresponds to a level in the hierarchy.

Cubes are built from dimensions and measure groups. The dimensions in the dimensions collection of a cube belong to the dimensions collection of the database. Measure groups are collections of measures that have the same data source view and have the same subset of dimensions from the cube. A measure group has one or more partitions to manage the physical data. A measure group can have a default aggregation design. The default aggregation design can be used by all partitions in the measure group; also, each partition can have its own aggregation design.

Server Objects

Each instance of Analysis Services is seen as a different server object in AMO; each different instance is connected to a **T:Microsoft.AnalysisServices.Server** object by a different connection. Each server object contains one or more data source, data source view, and database objects, as well as assemblies and security roles.

Dimension Objects

Each database object contains multiple dimension objects. Each dimension object contains one or more attributes, which are organized into hierarchies.

Cube Objects

Each database object contains one or more cube objects. A cube is defined by its measures

and dimensions. The measures and dimensions in a cube are derived from the tables and views in the data source view on which the cube is based, or which is generated from the measure and dimension definitions.

Object Inheritance

The ASSL object model contains many repeated element groups. For example, the element group, "**Dimensions** contain **Hierarchies**," defines the dimension hierarchy of an element. Both **Cubes** and **MeasureGroups** contain the element group, "**Dimensions** contain **Hierarchies**."

Unless explicitly overridden, an element inherits the details of these repeated element groups from the higher level. For example, the **Translations** for a **CubeDimension** are the same as the **Translations** for its ancestor element, **Cube**.

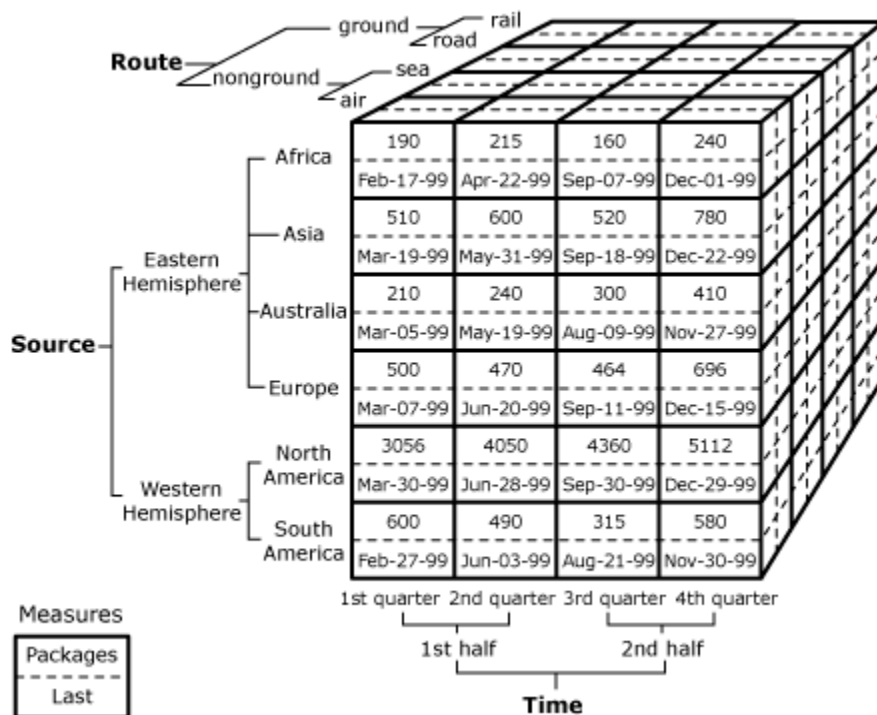
To explicitly override properties inherited from a higher-level object, an object does not need to repeat explicitly the entire structure and properties of the higher-level object. The only properties that an object needs to state explicitly are those properties that the object wants to override. For example, a **CubeDimension** may list only those **Hierarchies** that need to be disabled in the **Cube**, or for which the visibility needs to be changed, or for which some **Level** details have not been provided at the **Dimension** level.

Some properties specified on an object provide default values for the same property on a child or descendant object. For example, **Cube.StorageMode** provides the default value for **Partition.StorageMode**. For inherited default values, ASSL applies the same rules as used in Decision Support Objects (DSO) 8.0. The following list describes these rules for inherited default values:

- When the property for the child object is null in the XML, the property's value defaults to the inherited value. However, if you query the value from the server, the server returns the null value of the XML element.
- It is not possible to determine programmatically whether the property of a child object has been set directly on the child object or inherited.

Example

The Imports cube contains two measures, Packages and Last, and three related dimensions, Route, Source, and Time.



The smaller alphanumeric values around the cube are the members of the dimensions. Example members are ground (member of the Route dimension), Africa (member of the Source dimension), and 1st quarter (member of the Time dimension).

Measures

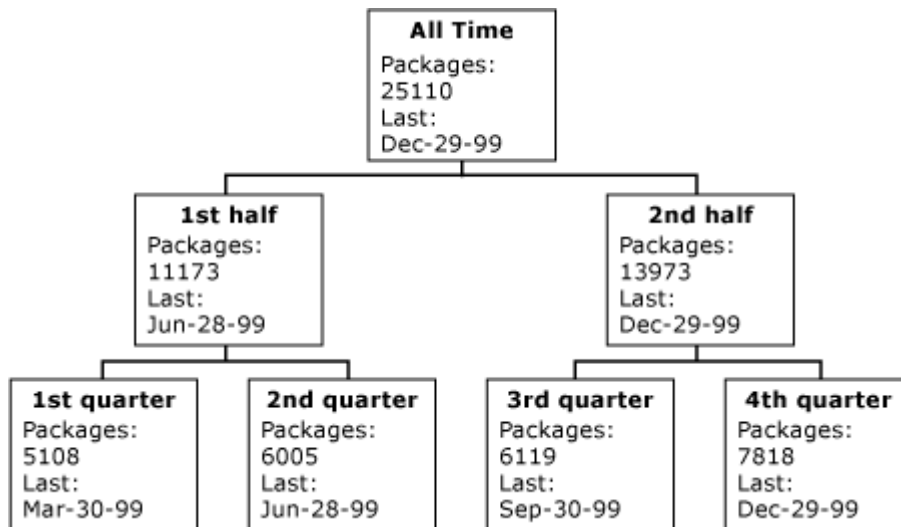
The values within the cube cells represent the two measures, Packages and Last. The Packages measure represents the number of imported packages, and the **Sum** function is used to aggregate the facts. The Last measure represents the date of receipt, and the **Max** function is used to aggregate the facts.

Dimensions

The Route dimension represents the means by which the imports reach their destination. Members of this dimension include ground, nonground, air, sea, road, or rail. The Source dimension represents the locations where the imports are produced, such as Africa or Asia. The Time dimension represents the quarters and halves of a single year.

Aggregates

Business users of a cube can determine the value of any measure for each member of every dimension, regardless of the level of the member within the dimension, because Analysis Services aggregates values at upper levels as needed. For example, the measure values in the preceding illustration can be aggregated according to a standard calendar hierarchy by using the Calendar Time hierarchy in the Time dimension as illustrated in the following diagram.



In addition to aggregating measures by using a single dimension, you can aggregate measures by using combinations of members from different dimensions. This allows business users to evaluate measures in multiple dimensions simultaneously. For example, if a business user wants to analyze quarterly imports that arrived by air from the Eastern Hemisphere and Western Hemisphere, the business user can issue a query on the cube to retrieve the following dataset.

			Packages		Last			
			All Sources	Eastern Hemisphere	Western Hemisphere	All Sources	Eastern Hemisphere	Western Hemisphere
All Time			25110	6547	18563	Dec-29-99	Dec-22-99	Dec-29-99
	1st half		11173	2977	8196	Jun-28-99	Jun-20-99	Jun-28-99
		1st quarter	5108	1452	3656	Mar-30-99	Mar-19-99	Mar-30-99
		2nd quarter	6065	1525	4540	Jun-28-99	Jun-20-99	Jun-28-99
	2nd half		13937	3570	10367	Dec-29-99	Dec-22-99	Dec-29-99
		3rd quarter	6119	1444	4675	Sep-30-99	Sep-18-99	Sep-30-99
		4th	7818	2126	5692	Dec-	Dec-22-99	Dec-29-99

			Packages		Last			
		quarter				29-99		

After a cube is defined, you can create new aggregations, or you can change existing aggregations to set options such as whether aggregations are precalculated during processing or calculated at query time. **Related topic:** [Defining an Analysis Services Database](#).

Mapping Measures, Attributes, and Hierarchies

The measures, attributes, and hierarchies in the example cube are derived from the following columns in the cube's fact and dimension tables.

Measure or attribute (level)	Members	Source table	Source column	Sample column value
Packages measure	Not applicable	ImportsFactTable	Packages	12
Last measure	Not applicable	ImportsFactTable	Last	May-03-99
Route Category level in Route dimension	nonground,ground	RouteDimensionTable	Route_Category	Nonground
Route attribute in Route dimension	air,sea,road,rail	RouteDimensionTable	Route	Sea
Hemisphere attribute in Source dimension	Eastern Hemisphere,Western Hemisphere	SourceDimensionTable	Hemisphere	Eastern Hemisphere
Continent attribute in Source dimension	Africa,Asia,AustraliaEurope, N. America,S. America	SourceDimensionTable	Continent	Europe
Half attribute in Time	1st half,2nd half	TimeDimensionTable	Half	2nd half

Measure or attribute (level)	Members	Source table	Source column	Sample column value
dimension				
Quarter attribute in Time dimension	1st quarter,2nd quarter,3rd quarter,4th quarter	TimeDimensionTable	Quarter	3rd quarter

Data in a single cube cell is usually derived from multiple rows in the fact table. For example, the cube cell at the intersection of the air member, the Africa member, and the 1st quarter member contains a value that is derived by aggregating the following rows in the **ImportsFactTable** fact table.

Import_ReceiptKey	RouteKey	SourceKey	TimeKey	Packages	Last
3516987	1	6	1	15	Jan-10-99
3554790	1	6	1	40	Jan-19-99
3572673	1	6	1	34	Jan-27-99
3600974	1	6	1	45	Feb-02-99
3645541	1	6	1	20	Feb-09-99
3674906	1	6	1	36	Feb-17-99

In the preceding table, each row has the same values for the **RouteKey**, **SourceKey**, and **TimeKey** columns, indicating that these rows contribute to the same cube cell.

The example shown here represents a very simple cube, in that the cube has a single measure group, and all the dimension tables are joined to the fact table in a star schema. Another common schema is a snowflake schema, in which one or more dimension tables join to another dimension table, rather than joining directly to the fact table. **Related topic:** [Dimensions](#).

The example shown here contains only a single fact table. When a cube has multiple fact tables, the measures from each fact table are organized into measure groups, and a measure group is

related to a specific set of dimensions by defined dimension relationships. These relationships are defined by specifying the participating tables in the data source view and the granularity of the relationship. **Related topic:** [Dimension Relationships](#).

See Also

[Working with Cubes and Measures](#)

[Working with Dimensions and Levels](#)

[Working with Analysis Services Databases](#)

Server Objects

Introducing Server Objects

The **T:Microsoft.AnalysisServices.Server** object represents the server and the instance of Microsoft SQL Server Analysis Services that you want to work with.

As soon as you have a connected instance of Analysis Services, you will be able to see:

- All databases that you can access, as a collection.
- All defined server properties, as a collection.
- The connection string, the connection information, and the session ID.
- The product name, edition, and version.
- The roles collections.
- The traces collection.
- The assemblies collection.

Database Objects

A Microsoft SQL Server Analysis Services instance contains database objects and assemblies for use with online analytical processing (OLAP) and data mining.

- Databases contain OLAP and data mining objects, such as data sources, data source views, cubes, measures, measure groups, dimensions, attributes, hierarchies, mining structures, mining models and roles.
- Assemblies contain user-defined functions that extend the functionality of the intrinsic functions provided with the Multidimensional Expressions (MDX) and Data Mining Extensions (DMX) languages.

The **T:Microsoft.AnalysisServices.Database** object is the container for all data objects that are needed for a business intelligence project (such as OLAP cubes, dimensions, and data mining structures), and their supporting objects (such as **T:Microsoft.AnalysisServices.DataSource**, **T:Microsoft.AnalysisServices.Account**, and **T:Microsoft.AnalysisServices.Role**).

A **T:Microsoft.AnalysisServices.Database** object provides access to objects and attributes that include the following:

- All cubes that you can access, as a collection.

- All dimensions that you can access, as a collection.
- All mining structures that you can access, as a collection.
- All data sources and data source views, as two collections.
- All roles and database permissions, as two collections.
- The collation value for the database.
- The estimated size of the database.
- The language value of the database.
- The visible setting for the database.

In This Section

The following topics describe objects shared by both OLAP and data mining features in Analysis Services.

Topic	Description
Working with Data Mining	Describes a data source in Analysis Services.
Data Source Views (Analysis Services)	Describes a logical data model based on one or more data sources, in Analysis Services.
Cubes	Describes cubes and cube objects, including measures, measure groups, dimension usage relationships, calculations, key performance indicators, actions, translations, partitions, and perspectives.
Dimensions	Describes dimensions and dimension objects, including attributes, attribute relationships, hierarchies, levels, and members.
Mining Structures	Describes mining structures and mining objects, including mining models.
Roles	Describes a role, the security mechanism used to control access to objects in Analysis Services.
Assemblies (Analysis Services)	Describes an assembly, a collection of user-defined functions used to extend the MDX and DMX languages, in Analysis Services.

See Also

[Working with Data Sources](#)

[Data Source Views](#)

[Designing Analysis Services Multidimensional Database Objects](#)

[Working with Data Mining](#)

Security Roles

Roles are used in Microsoft SQL Server Analysis Services to manage security for Analysis Services objects and data. In basic terms, a role associates the security identifiers (SIDs) of Microsoft Windows users and groups that have specific access rights and permissions defined for objects managed by an instance of Analysis Services. Two types of roles are provided in Analysis Services:

- The server role, a fixed role that provides administrator access to an instance of Analysis Services.
- Database roles, roles defined by administrators to control access to objects and data for non-administrator users.

Security in Microsoft SQL Server Analysis Services security is managed by using roles and permissions. Roles are groups of users. Users, also called members, can be added or removed from roles. Permissions for objects are specified by roles, and all members in a role can use the objects for which the role has permissions. All members in a role have equal permissions to the objects. Permissions are particular to objects. Each object has a permissions collection with the permissions granted on that object, different sets of permissions can be granted on an object. Each permission, from the permissions collection of the object, has a single role assigned to it.

Role and Role Member Objects

A role is a containing object for a collection of users (members). A Role definition establishes the membership of the users in Analysis Services. Because permissions are assigned by role, a user must be a member of a role before the user has access to any object.

A **T:Microsoft.AnalysisServices.Role** object is composed of the parameters Name, Id, and Members. Members is a collection of strings. Each member contains the user name in the form of "domain\username". Name is a string that contains the name of the role. ID is a string that contains the unique identifier of the role.

Server Role

The Analysis Services server role defines administrative access of Windows users and groups to an instance of Analysis Services. Members of this role have access to all Analysis Services databases and objects on an instance of Analysis Services, and can perform the following tasks:

- Perform server-level administrative functions using SQL Server Management Studio or SQL Server Data Tools (SSDT), including creating databases and setting server-level properties.
- Perform administrative functions programmatically with Analysis Management Objects (AMO).
- Maintain Analysis Services database roles.

- Start traces (other than for processing events, which can be performed by a database role with Process access).

Every instance of Analysis Services has a server role that defines which users can administer that instance. The name and ID of this role is Administrators, and unlike database roles, the server role cannot be deleted, nor can permissions be added or removed. In other words, a user either is or is not an administrator for an instance of Analysis Services, depending on whether he or she is included in the server role for that instance of Analysis Services. **Related topics:** [Granting User Access](#), [Setting Server Configuration Properties](#).

Database Roles

An Analysis Services database role defines user access to objects and data in an Analysis Services database. A database role is created as a separate object in an Analysis Services database, and applies only to the database in which that role is created. Windows users and groups are included in the role by an administrator, who also defines permissions within the role.

The permissions of a role may allow members to access and administer the database, in addition to the objects and data within the database. Each permission has one or more access rights associated with it, which in turn give the permission finer control over access to a particular object in the database. **Related topics:** [Permissions and Access Rights](#), [Granting User Access](#)

Permission Objects

Permissions are associated with an object (cube, dimension, others) for a particular role. Permissions specify what operations the member of that role can perform on that object.

The **T:Microsoft.AnalysisServices.Permission** class is an abstract class. Therefore, you must use the derived classes to define permissions on the corresponding objects. For each object, a permission derived class is defined.

Object	Class
T:Microsoft.AnalysisServices.Database	T:Microsoft.AnalysisServices.DatabasePermission
T:Microsoft.AnalysisServices.DataSource	T:Microsoft.AnalysisServices.DataSourcePermission
T:Microsoft.AnalysisServices.Dimension	T:Microsoft.AnalysisServices.DimensionPermission
T:Microsoft.AnalysisServices.Cube	T:Microsoft.AnalysisServices.CubePermission
T:Microsoft.AnalysisServices.MiningStructure	T:Microsoft.AnalysisServices.MiningStructurePermission
T:Microsoft.AnalysisServices.MiningModel	T:Microsoft.AnalysisServices.MiningModelPermission

Possible actions enabled by permissions are shown in the list:

Action	Values	Explanation
Process	{ true , false } Default= false	If true , members can process the object and any object that is contained in the object. Process permissions do not apply to mining models. T:Microsoft.AnalysisServices.MiningModel permissions are always inherited from T:Microsoft.AnalysisServices.MiningStructure .
ReadDefinition	{ None , Basic , Allowed } Default= None	Specifies whether members can read the data definition (ASSL) associated with the object. If Allowed , members can read the ASSL associated with the object. Basic and Allowed are inherited by objects that are contained in the object. Allowed overrides Basic and None . Allowed is required for DISCOVER_XML_METADATA on an object. Basic is required to create linked objects and local cubes.
Read	{ None , Allowed } Default= None (Except for DimensionPermission, where default= Allowed)	Specifies whether members have read access to schema rowsets and data content. Allowed gives read access on a database, which lets you discover a database. Allowed on a cube gives read access in schema rowsets and access to cube content (unless constrained by T:Microsoft.AnalysisServices.CellPermission and T:Microsoft.AnalysisServices.CubeDimensionPermission). Allowed on a dimension grants that read permission on all attributes in the dimension (unless constrained by T:Microsoft.AnalysisServices.CubeDimensionPermission). Read permission is used for static inheritance to the T:Microsoft.AnalysisServices.CubeDimensionPermission only. None on a dimension hides the dimension and gives access to the default member only for aggregatable attributes; an error is raised if the dimension contains a non-aggregatable attribute. Allowed on a

Action	Values	Explanation
		T:Microsoft.AnalysisServices.MiningModelPermission grants permissions to see objects in schema rowsets and to perform predict joins. Note Allowed is required to read or write to any object in the database.
Write	{None, Allowed} Default=None	Specifies whether members have write access to data of the parent object. Access applies to T:Microsoft.AnalysisServices.Dimension, T:Microsoft.AnalysisServices.Cube, and T:Microsoft.AnalysisServices.MiningModel subclasses. It does not apply to database T:Microsoft.AnalysisServices.MiningStructure subclasses, which generates a validation error. Allowed on a T:Microsoft.AnalysisServices.Dimension grants write permission on all attributes in the dimension. Allowed on a T:Microsoft.AnalysisServices.Cube grants write permission on the cells of the cube for partitions defined as Type=writeback. Allowed on a T:Microsoft.AnalysisServices.MiningModel grants permission to modify model content. Allowed on a T:Microsoft.AnalysisServices.MiningStructure has no specific meaning in Analysis Services.  Note Write cannot be set to Allowed unless read is also set to Allowed
Administer  Note Only in Database permissions	{true, false} Default=false	Specifies whether members can administer a database. true grants members access to all objects in a database. A member can have Administer permissions for a specific database, but not for others.

See Also

[Permissions and Access Rights](#)
[Security and Protection](#)
[Granting Administrative Access](#)
[Granting User Access](#)

Dimension Objects

A simple **T:Microsoft.AnalysisServices.Dimension** object is composed of basic information, attributes, and hierarchies. Basic information includes the name of the dimension, the type of the dimension, the data source, the storage mode, and others. Attributes define the actual data in the dimension. Attributes do not necessarily belong to a hierarchy, but hierarchies are built from attributes. A hierarchy creates ordered lists of levels, and defines the ways a user can explore the dimension.

In This Section

The following topics provide more information about how to design and implement dimension objects.

Topic	Description
Dimensions (Analysis Services)	In Microsoft SQL Server Analysis Services, dimensions are a fundamental component of cubes. Dimensions organize data with relation to an area of interest, such as customers, stores, or employees, to users.
Attributes and Attribute Hierarchies	Dimensions are collections of attributes, which are bound to one or more columns in a table or view in the data source view.
Attribute Relationships	In Microsoft SQL Server Analysis Services, attributes within a dimension are always related either directly or indirectly to the key attribute. When you define a dimension based on a star schema, which is where all dimension attributes are derived from the same relational table, an attribute relationship is automatically defined between the key attribute and each non-key attribute of the dimension. When you define a dimension based on a snowflake schema, which is where dimension attributes are derived from multiple related tables, an attribute relationship is

Topic	Description
	automatically defined as follows: • Between the key attribute and each non-key attribute bound to columns in the main dimension table. • Between the key attribute and the attribute bound to the foreign key in the secondary table that links the underlying dimension tables. • Between the attribute bound to foreign key in the secondary table and each non-key attribute bound to columns from the secondary table.

Dimensions

In Microsoft SQL Server Analysis Services, dimensions are a fundamental component of cubes. Dimensions organize data with relation to an area of interest, such as customers, stores, or employees, to users. Dimensions in Analysis Services contain attributes that correspond to columns in dimension tables. These attributes appear as attribute hierarchies and can be organized into user-defined hierarchies, or can be defined as parent-child hierarchies based on columns in the underlying dimension table. Hierarchies are used to organize measures that are contained in a cube. The following topics provide an overview of dimensions, attributes, and hierarchies.

In This Section

Topic	Description
Solution Overview (Analysis Services - Multidimensional Data)	Provides an overview of dimension concepts.
Attributes and Attribute Hierarchies	Describes attributes and attribute hierarchies.
User-defined Hierarchies	Describes user-defined hierarchies of attributes.
Write-Enabled Dimensions	Describes write-enabled dimensions.
Dimension Translations	Describes translations of dimension meta data.

See Also

[Defining and Configuring Dimensions, Attributes and Hierarchies](#)

[Cube Objects \(Analysis Services - Multidimensional Data\)](#)

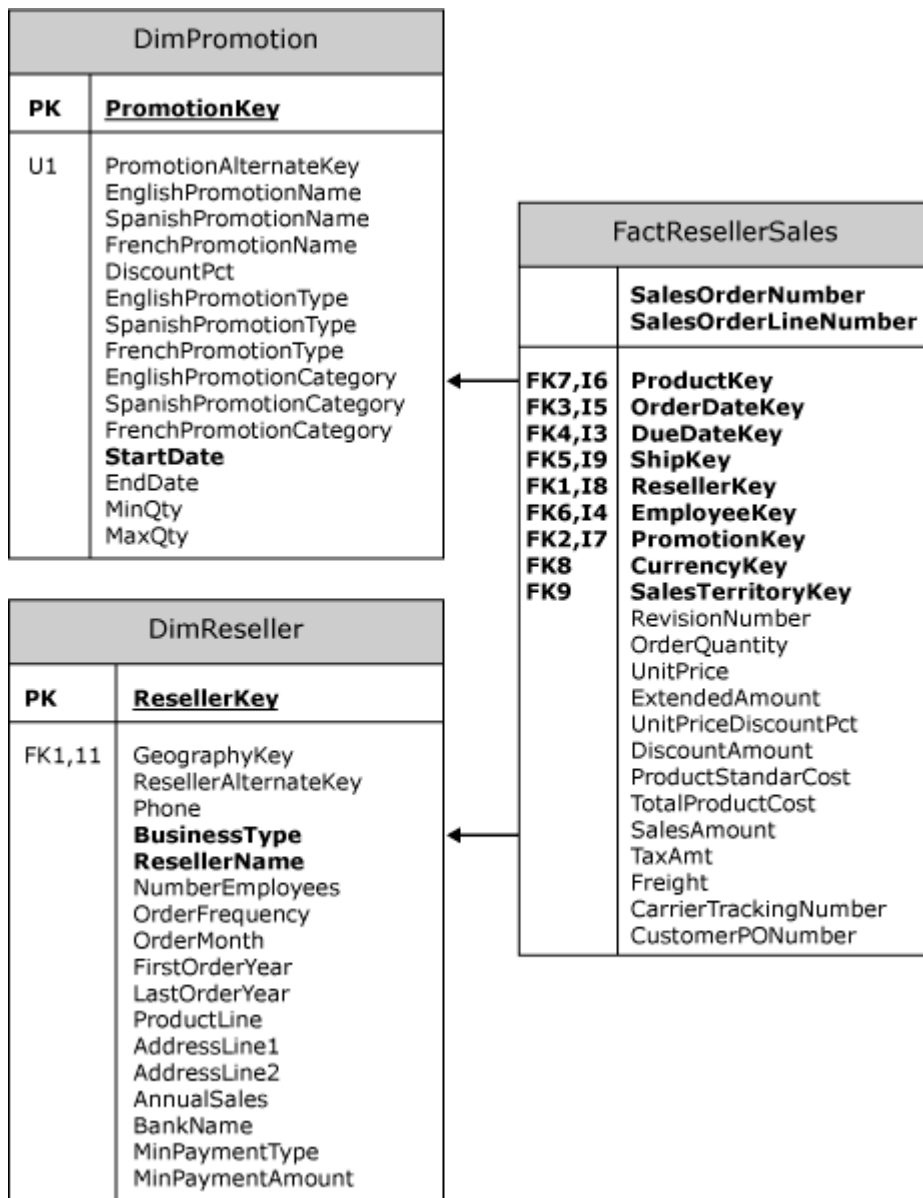
Introduction to Dimensions (Analysis Services - Multidimensional Data)

All Microsoft SQL Server Analysis Services dimensions are groups of attributes based on columns from tables or views in a data source view. Dimensions exist independent of a cube, can be used in multiple cubes, can be used multiple times in a single cube, and can be linked between Analysis Services instances. A dimension that exists independent of a cube is called a database dimension and an instance of a database dimension within a cube is called a cube dimension.

Dimension based on a Star Schema Design

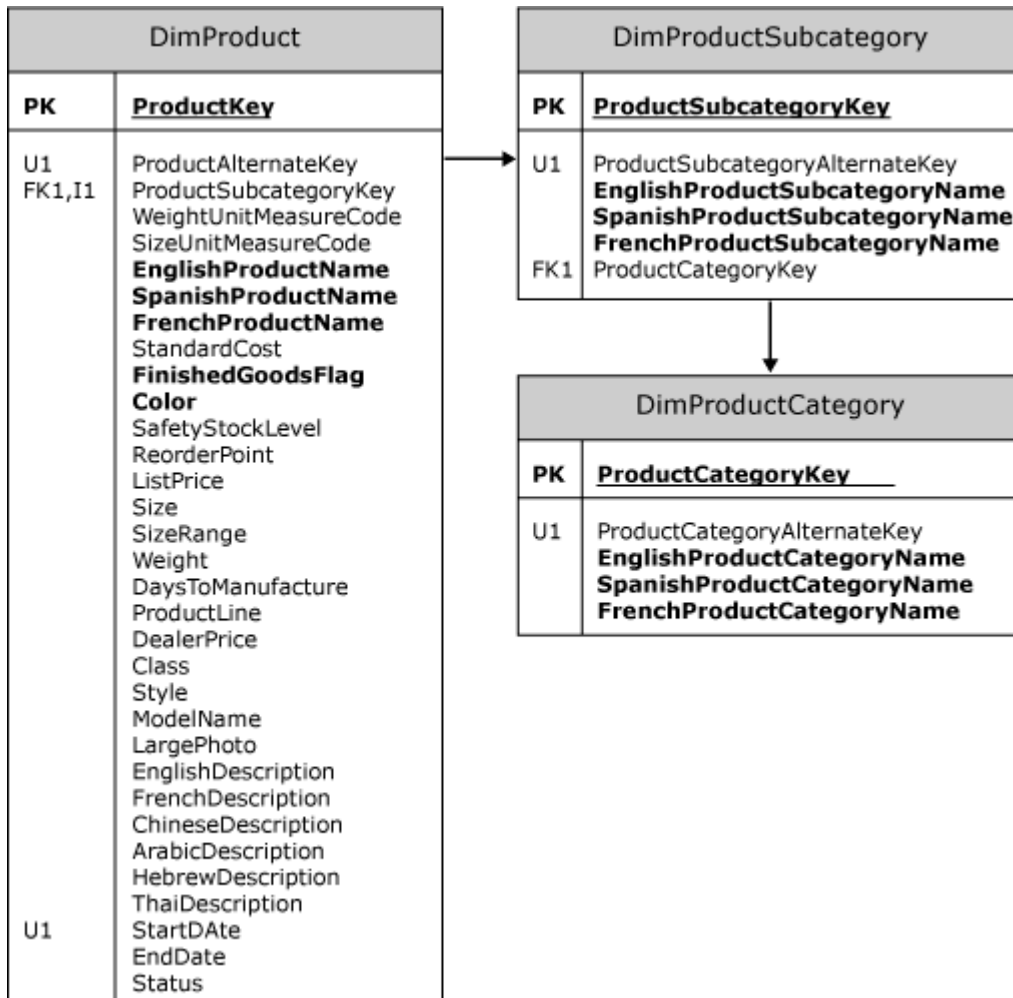
The structure of a dimension is largely driven by the structure of the underlying dimension table or tables. The simplest structure is called a star schema, where each dimension is based on a single dimension table that is directly linked to the fact table by a primary key - foreign key relationship.

The following diagram illustrates a subsection of the sample database, in which the **FactResellerSales** fact table is related to two dimension tables, **DimReseller** and **DimPromotion**. The **ResellerKey** column in the **FactResellerSales** fact table defines a foreign key relationship to the **ResellerKey** primary key column in the **DimReseller** dimension table. Similarly, the **PromotionKey** column in the **FactResellerSales** fact table defines a foreign key relationship to the **PromotionKey** primary key column in the **DimPromotion** dimension table.



Dimension based on a Snowflake Schema Design

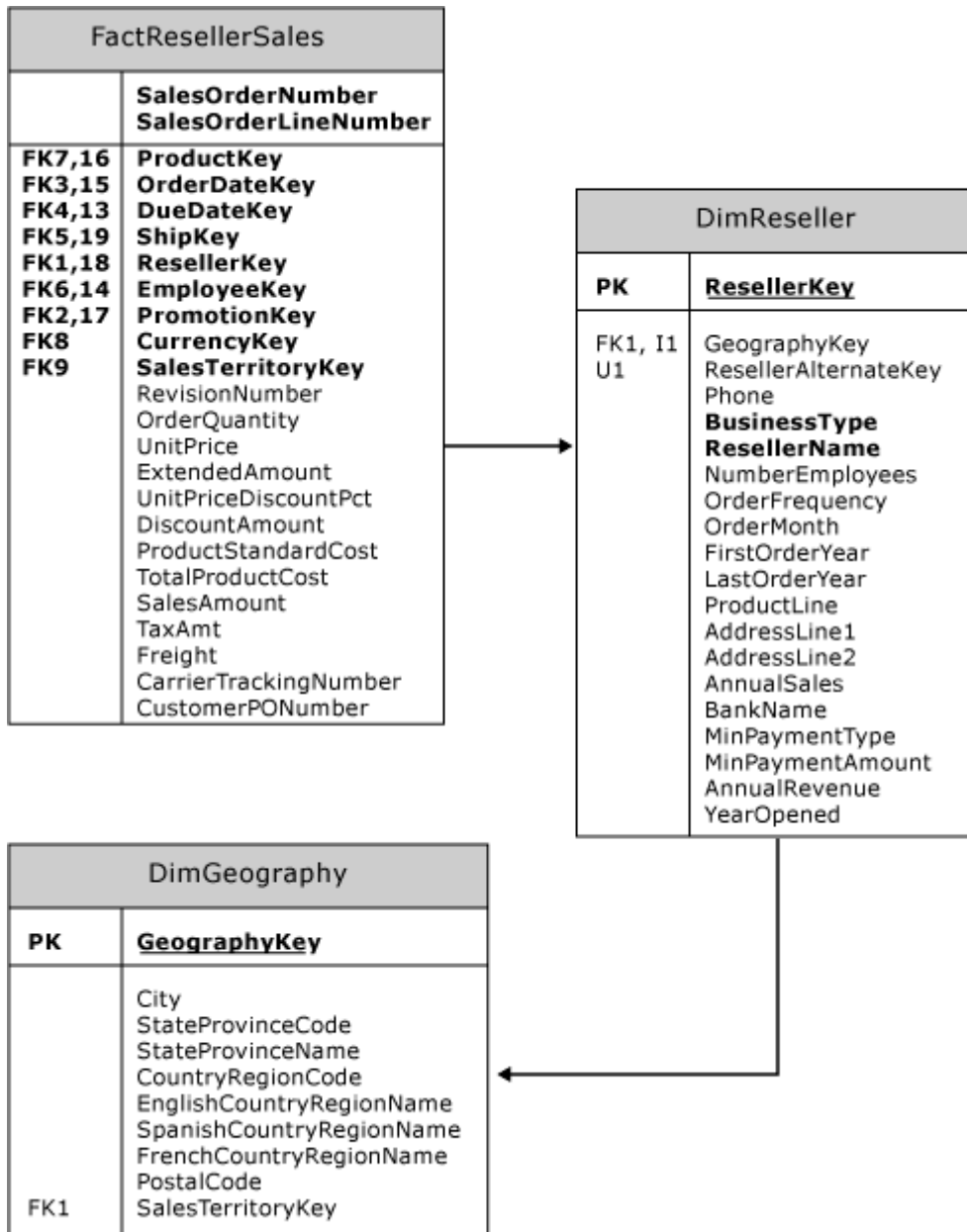
Frequently, a more complex structure is required because information from multiple tables is required to define the dimension. In this structure, called a snowflake schema, each dimension is based on attributes from columns in multiple tables linked to each other and ultimately to the fact table by primary key - foreign key relationships. For example, the following diagram illustrates the tables required to completely describe the Product dimension in the **AdventureWorksDW** sample project:



To completely describe a product, the product's category and subcategory must be included in the Product dimension. However, that information does not reside directly in the main table for the **DimProduct** dimension. A foreign key relationship from **DimProduct** to **DimProductSubcategory**, which in turn has a foreign key relationship to the **DimProductCategory** table, makes it possible to include the information for product categories and subcategories in the Product dimension.

Snowflake Schema versus Reference Relationship

Sometimes, you may have a choice between using a snowflake schema to define attributes in a dimension from multiple tables, or defining two separate dimensions and defining a reference dimension relationship between them. The following diagram illustrates such a scenario.



In the previous diagram, the **FactResellerSales** fact table does not have a foreign key relationship with the **DimGeography** dimension table. However, the **FactResellerSales** fact table does have a foreign key relationship with the **DimReseller** dimension table, which in turn has a foreign key relationship with the **DimGeography** dimension table. To define a Reseller dimension that contains geography information about each reseller, you would have to retrieve these attributes from the **DimGeography** and the **DimReseller** dimension tables. However, in Analysis Services, you can achieve the same result by creating two separate dimensions and linking them in a measure group by defining a reference dimension relationship between the

two dimensions. For more information about reference dimension relationships, see [Using the Cube Wizard to Define a Cube, Dimensions, Hierarchies and Attributes](#).

One advantage of using reference dimension relationships in this scenario is that you could create a single geography dimension and then create multiple cube dimensions based on the geography dimension, without requiring any additional storage space. For example, you could link one of the geography cube dimensions to a reseller dimension and another of the geography cube dimensions to a customer dimension. **Related topics:** [Dimension Relationships](#), [Defining a Referenced Relationship](#) and [Referenced Relationship Properties](#)

Processing a Dimension

After you create a dimension, you must process the dimension before you can view the members of the attributes and hierarchies in the dimension. After the structure of a dimension is changed or the information in its underlying tables is updated, you have to process the dimension again before you can view the changes. When you process a dimension after structural changes, you must also process any cubes that include the dimension - or the cube will not be viewable.

Security

All the subordinate objects of a dimension, including hierarchies, levels, and members, are secured using roles in Analysis Services. Dimension security can be applied for all the cubes in the database that use the dimension, or for only a specific cube. For more information about dimension security, see [Granting Dimension Access](#).

See Also

[Dimension Storage](#)

[Dimension Translations](#)

[Write-Enabled Dimensions](#)

[Using the Dimension Wizard to Define a Dimension, Hierarchies, and Attributes](#)

Dimension Storage

Dimensions in Microsoft SQL Server Analysis Services support two storage modes:

- Relational OLAP (ROLAP)
- Multidimensional OLAP (MOLAP)

The storage mode determines the location and form of a dimension's data. MOLAP is the default storage mode for dimensions. **Related topics:** [Partition Storage Modes and Processing](#)

MOLAP

Data for a dimension that uses MOLAP is stored in a multidimensional structure in the instance of Analysis Services. This multidimensional structure is created and populated when the dimension is processed. MOLAP dimensions provide better query performance than ROLAP dimensions.

ROLAP

Data for a dimension that uses ROLAP is actually stored in the tables used to define the dimension. The ROLAP storage mode can be used to support large dimensions without duplicating large amounts of data, but at the expense of query performance. Because the dimension relies directly on the tables in the data source view used to define the dimension, the ROLAP storage mode also supports real-time OLAP.

Important

If a dimension uses the ROLAP storage mode and the dimension is included in a cube that uses MOLAP storage, any schema changes to its source table must be followed by immediate processing of the cube. Failure to do this may result in inconsistent results when querying the cube. **Related topic:** [Processing Objects Using Integration Services](#).

See Also

[Partition Storage Modes and Processing](#)

Attributes and Attribute Hierarchies

Dimensions are collections of attributes, which are bound to one or more columns in a table or view in the data source view.

Key Attribute

Each dimension contains a key attribute. Each attribute bound to one or more columns in a dimension table. The key attribute is the attribute in a dimension that identifies the columns in the dimension main table that are used in foreign key relationships to the fact table. Typically, the key attribute represents the primary key column or columns in the dimension table. You can define a logical primary key on a table in a data source view which has no physical primary key in the underlying data source. **For more information**, see [Defining Dimension Attributes](#). When defining key attributes, the Cube Wizard and Dimension Wizard try to use the primary key columns of the dimension table in the data source view. If the dimension table does not have a logical primary key or physical primary key defined, the wizards may not be able to correctly define the key attributes for the dimension.

Binding an Attribute to Columns in Data Source View Tables or Views

An attribute is bound to columns in one or more data source view tables or views. An attribute is always bound to one or more key columns, which determines the members that are contained by the attribute. By default, this is the only column to which an attribute is bound. An attribute can also be bound to one or more additional columns for specific purposes. For example, an attribute's **NameColumn** property determines the name that appears to the user for each attribute member - this property of the attribute can be bound to a particular dimension column through a data source view or can be bound to a calculated column in the data source view. For more information, see [Defining and Configuring Dimension Attributes](#).

Attribute Hierarchies

By default, attribute members are organized into two level hierarchies, consisting of a leaf level and an All level. The All level contains the aggregated value of the attribute's members across the measures in each measure group to which the dimension of which the attribute is related is

a member. However, if the **IsAggregatable** property is set to False, the All level is not created. For more information, see [Defining and Configuring Dimension Attributes](#).

Attributes can be, and typically are, arranged into user-defined hierarchies that provide the drill-down paths by which users can browse the data in the measure groups to which the attribute is related. In client applications, attributes can be used to provide grouping and constraint information. When attributes are arranged into user-defined hierarchies, you define relationships between hierarchy levels when levels are related in a many-to-one or a one-to-one relationship (called a *natural* relationship). For example, in a Calendar Time hierarchy, a Day level should be related to the Month level, the Month level related to the Quarter level, and so on. Defining relationships between levels in a user-defined hierarchy enables Analysis Services to define more useful aggregations to increase query performance and can also save memory during processing performance, which can be important with large or complex cubes. For more information, see [User-defined Hierarchies](#), [Defining and Configuring a User-defined Hierarchy](#), and [Defining and Configuring an Attribute Relationship](#).

Attribute Relationships, Star Schemas, and Snowflake Schemas

By default, in a star schema, all attributes are directly related to the key attribute, which enables users to browse the facts in the cube based on any attribute hierarchy in the dimension. In a snowflake schema, an attribute is either directly linked to the key attribute if their underlying table is directly linked to the fact table or is indirectly linked by means of the attribute that is bound to the key in the underlying table that links the snowflake table to the directly linked table.

See Also

[Defining and Configuring a User-defined Hierarchy](#)
[Defining and Configuring an Attribute Relationship](#)
[Defining and Configuring Dimension Attributes](#)

Attribute Relationships

In Microsoft SQL Server Analysis Services, attributes within a dimension are always related either directly or indirectly to the key attribute. When you define a dimension based on a star schema, which is where all dimension attributes are derived from the same relational table, an attribute relationship is automatically defined between the key attribute and each non-key attribute of the dimension. When you define a dimension based on a snowflake schema, which is where dimension attributes are derived from multiple related tables, an attribute relationship is automatically defined as follows:

- Between the key attribute and each non-key attribute bound to columns in the main dimension table.
- Between the key attribute and the attribute bound to the foreign key in the secondary table that links the underlying dimension tables.
- Between the attribute bound to foreign key in the secondary table and each non-key attribute bound to columns from the secondary table.

However, there are a number of reasons why you might want to change these default attribute relationships. For example, you might want to define a natural hierarchy, a custom sort order, or dimension granularity based on a non-key attribute. For more information, see [User Hierarchy Properties](#).



Note

Attribute relationships are known in Multidimensional Expressions (MDX) as member properties.

Natural Hierarchy Relationships

A hierarchy is a natural hierarchy when each attribute included in the user-defined hierarchy has a one to many relationship with the attribute immediately below it. For example, consider a Customer dimension based on a relational source table with eight columns:

- CustomerKey
- CustomerName
- Age
- Gender
- Email
- City
- Country
- Region

The corresponding Analysis Services dimension has seven attributes:

- Customer (based on CustomerKey, with CustomerName supplying member names)
- Age, Gender, Email, City, Region, Country

Relationships representing natural hierarchies are enforced by creating an attribute relationship between the attribute for a level and the attribute for the level below it. For Analysis Services, this specifies a natural relationship and potential aggregation. In the Customer dimension, a natural hierarchy exists for the Country, Region, City, and Customer attributes. The natural hierarchy for {Country, Region, City, Customer} is described by adding the following attribute relationships:

- The Country attribute as an attribute relationship to the Region attribute.
- The Region attribute as an attribute relationship to the City attribute.
- The City attribute as an attribute relationship to the Customer attribute.

For navigating data in the cube, you can also create a user-defined hierarchy that does not represent a natural hierarchy in the data (which is called an *ad hoc* or *reporting* hierarchy). For example, you could create a user-defined hierarchy based on {Age, Gender}. Users do not see any difference in how the two hierarchies behave, although the natural hierarchy benefits from aggregating and indexing structures — hidden from the user — that account for the natural relationships in the source data.

The **SourceAttribute** property of a level determines which attribute is used to describe the level. The **KeyColumns** property on the attribute specifies the column in the data source view that supplies the members. The **NameColumn** property on the attribute can specify a different name column for the members.

To define a level in a user-defined hierarchy using SQL Server Data Tools (SSDT), the **Dimension Designer** allows you to select a dimension attribute, a column in a dimension table, or a column from a related table included in the data source view for the cube. For more information about creating user-defined hierarchies, see [Defining and Configuring a User-defined Hierarchy](#).

In Analysis Services, an assumption is usually made about the content of members. Leaf members have no descendents and contain data derived from underlying data sources. Nonleaf members have descendents and contain data derived from aggregations performed on child members. In aggregated levels, members are based on aggregations of subordinate levels. Therefore, when the **IsAggregatable** property is set to **False** on a source attribute for a level, no aggregatable attributes should be added as levels above it.

Defining an Attribute Relationship

The main constraint when you create an attribute relationship is to make sure that the attribute referred to by the attribute relationship has no more than one value for any member in the attribute to which the attribute relationship belongs. For example, if you define a relationship between a City attribute and a State attribute, each city can only relate to a single state.

Attribute Relationship Queries

You can use MDX queries to retrieve data from attribute relationships, in the form of member properties, with the **PROPERTIES** keyword of the MDX **SELECT** statement. For more information about how to use MDX to retrieve member properties, see [Using Member Properties \(MDX\)](#).

See Also

[Attributes and Attribute Hierarchies](#)

[Configuring Dimension Attribute Properties](#)

[User-defined Hierarchies](#)

[Configuring User-defined Hierarchy Properties](#)

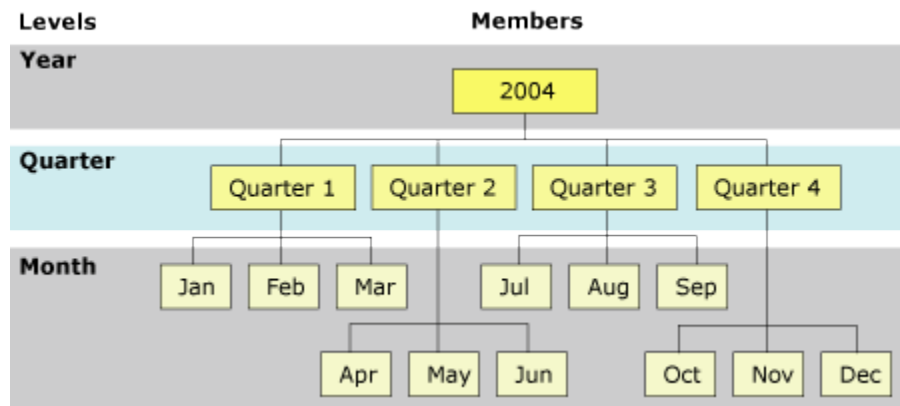
User Hierarchies

User-defined hierarchies are user-defined hierarchies of attributes that are used in Microsoft SQL Server Analysis Services to organize the members of a dimension into hierarchical structures and provide navigation paths in a cube. For example, the following table defines a dimension table for a time dimension. The dimension table supports three attributes, named Year, Quarter, and Month.

Year	Quarter	Month
1999	Quarter 1	Jan

Year	Quarter	Month
1999	Quarter 1	Feb
1999	Quarter 1	Mar
1999	Quarter 2	Apr
1999	Quarter 2	May
1999	Quarter 2	Jun
1999	Quarter 3	Jul
1999	Quarter 3	Aug
1999	Quarter 3	Sep
1999	Quarter 4	Oct
1999	Quarter 4	Nov
1999	Quarter 4	Dec

The Year, Quarter, and Month attributes are used to construct a user-defined hierarchy, named Calendar, in the time dimension. The relationship between the levels and members of the Calendar dimension (a regular dimension) is shown in the following diagram.



Note

Any hierarchy other than the default two-level attribute hierarchy is called a user-defined hierarchy. For more information about attribute hierarchies, see [Defining Dimension Attributes](#).

Member Structures

With the exception of parent-child hierarchies, the positions of members within the hierarchy are controlled by the order of the attributes in the hierarchy's definition. Each attribute in the

hierarchy definition constitutes a level in the hierarchy. The position of a member within a level is determined by the ordering of the attribute used to create the level. The member structures of user-defined hierarchies can take one of four basic forms, depending on how the members are related to each other.

Balanced Hierarchies

In a balanced hierarchy, all branches of the hierarchy descend to the same level, and each member's logical parent is the level immediately above the member. The Product Categories hierarchy of the Product dimension in the Adventure Works DW Multidimensional 2012 sample Analysis Services database is a good example of a balanced hierarchy. Each member in the Product Name level has a parent member in the Subcategory level, which in turn has a parent member in the Category level. Also, every branch in the hierarchy has a leaf member in the Product Name level.

Unbalanced Hierarchies

In an unbalanced hierarchy, branches of the hierarchy descend to different levels. Parent-child hierarchies are unbalanced hierarchies. For example, the Organization dimension in the Adventure Works DW Multidimensional 2012 sample Analysis Services database contains a member for each employee. The CEO is the top member in the hierarchy, and the division managers and executive secretary are immediately beneath the CEO. The division managers have subordinate members but the executive secretary does not.

It may be impossible for end users to distinguish between unbalanced and ragged hierarchies. However, you employ different techniques and properties in Analysis Services to support these two types of hierarchies. For more information, see [Working with Attributes in Ragged Hierarchies](#), and [Working with Attributes in Parent-Child Hierarchies](#).

Ragged Hierarchies

In a ragged hierarchy, the logical parent member of at least one member is not in the level immediately above the member. This can cause branches of the hierarchy to descend to different levels. For example, in a Geography dimension defined with the levels Continent, CountryRegion, and City, in that order, the member Europe appears in the top level of the hierarchy, the member France appears in the middle level, and the member Paris appears in the bottom level. France is more specific than Europe, and Paris is more specific than France. To this regular hierarchy, the following changes are made:

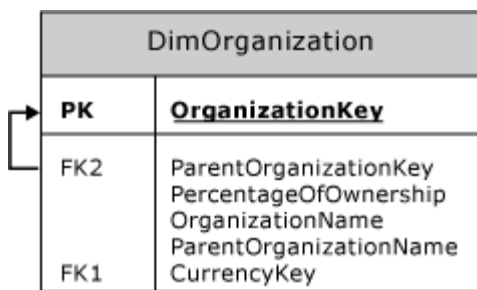
- The Vatican City member is added to the CountryRegion level.
- Members are added to the City level and are associated with the Vatican City member in the CountryRegion level.
- A level, named Province, is added between the CountryRegion and City levels.

The Province level is populated with members associated with other members in the CountryRegion level, and members in the City level are associated with their corresponding members in the Province level. However, because the Vatican City member in the CountryRegion level has no associated members in the Province level, members must be associated from the City level directly to the Vatican City member in the CountryRegion level. Because of the

changes, the hierarchy of the dimension is now ragged. The parent of the city Vatican City is the country/region Vatican City, which is not in the level immediately above the Vatican City member in the City level. For more information, see [Working with Attributes in Ragged Hierarchies](#).

Parent-Child Hierarchies

Parent-child hierarchies for dimensions are defined by using a special attribute, called a parent attribute, to determine how members relate to each other. A parent attribute describes a *self-referencing relationship*, or *self-join*, within a dimension main table. Parent-child hierarchies are constructed from a single parent attribute. Only one level is assigned to a parent-child hierarchy, because the levels present in the hierarchy are drawn from the parent-child relationships between members associated with the parent attribute. The dimension schema of a parent-child hierarchy depends on a self-referencing relationship present on the dimension main table. For example, the following diagram illustrates the **DimOrganization** dimension main table in the Adventure Works DW Multidimensional 2012 Analysis Services sample database.



In this dimension table, the **ParentOrganizationKey** column has a foreign key relationship with the **OrganizationKey** primary key column. In other words, each record in this table can be related through a parent-child relationship with another record in the table. This kind of self-join is generally used to represent organization entity data, such as the management structure of employees in a department.

When you create a parent-child hierarchy, the columns represented by both attributes must have the same data type. Both attributes must also be in the same table. By default, any member whose parent key equals its own member key, null, 0 (zero), or a value absent from the column for member keys is assumed to be a member of the top level (excluding the (All) level).

The depth of a parent-child hierarchy can vary among its hierarchical branches. In other words, a parent-child hierarchy is considered an unbalanced hierarchy.

Unlike user-defined hierarchies, in which the number of levels in the hierarchy determines the number of levels that can be seen by end users, a parent-child hierarchy is defined with the single level of an attribute hierarchy, and the values in this single level produce the multiple levels seen by users. The number of displayed levels depends on the contents of the dimension table columns that store the member keys and the parent keys. The number of levels can change when the data in the dimension tables change. For more information, see [Defining a Parent-Child Hierarchy](#), and [Working with Attributes in Parent-Child Hierarchies](#).

See Also

[Defining and Configuring a User-defined Hierarchy](#)

[Configuring User-defined Hierarchy Properties](#)

[Configuring Dimension Attribute Properties](#)

User Hierarchy Properties

The following table describes the properties of a user-defined hierarchy.

Property	Description
AllMemberName	Contains the caption in the default language for the All member of the hierarchy.
AllowDuplicateNames	Determines whether duplicate names are allowed in the hierarchy. Values are True and False. Default value is True.
Description	Contains the description of the hierarchy.
DisplayFolder	Specifies the folder in which to list the hierarchy for users.
ID	Contains the unique identifier (ID) of the hierarchy.
MemberNamesUnique	Determines whether member names in the hierarchy must be unique. Values are True and False. Default value is False.
Name	Contains the name of the hierarchy.

See Also

[Level Properties](#)

[Configuring Level Properties](#)

Level Properties

The following table lists and describes the properties of a level in a user-defined hierarchy.

Property	Description
Description	Contains the description of the level.

Property	Description
HideMemberIf	Indicates whether and when a member in a level should be hidden from client applications. This property can have the following values: Never Members are never hidden. This is the default value. OnlyChildWithNoName A member is hidden when the member is the only child of its parent and the member's name is empty. OnlyChildWithParentName A member is hidden when the member is the only child of its parent and the member's name is identical to that of its parent. NoName A member is hidden when the member's name is empty. ParentName A member is hidden when the member's name is identical to that of its parent.
ID	Contains the unique identifier (ID) of the level.
Name	Contains the friendly name of the level. By default, the name of a level is the same as the name of the source attribute.
SourceAttribute	Contains the name of the source attribute on which the level is based.

See Also

[User Hierarchy Properties](#)

Write-Enabled Dimensions

The data in a dimension is generally read-only. However, for certain scenarios, you may want to write-enable a dimension. In Microsoft SQL Server Analysis Services, write-enabling a dimension enables business users to modify the contents of the dimension and see the immediate affect of changes on the hierarchies of the dimension. Any dimension that is based on a single table can be write-enabled. In a write-enabled dimension, business users and administrators can change, move, add, and delete attribute members within the dimension. These updates are referred to collectively as *dimension writeback*.

Analysis Services supports dimension writeback on all dimension attributes and any member of a dimension may be modified. For a write-enabled cube or partition, updates are stored in a writeback table separate from the cube's source tables. However, for a write-enabled dimension, updates are recorded directly in the dimension's table. Also, if the write-enabled dimension is included in a cube with multiple partitions where some or all their data sources have copies of the dimension table, only the original dimension table is updated during a writeback process.

Write-enabled dimensions and write-enabled cubes have different but complementary features. A write-enabled dimension gives business users the ability to update members, whereas a write-enabled cube gives them the ability to update cell values. Although these two features are complementary, you do not have to use both features in combination. A dimension does not have to be included in a cube for dimension writeback to occur. A write-enabled dimension can also be included in a cube that is not write-enabled. You use different procedures to write-enable dimensions and cubes, and to maintain their security.

The following restrictions apply to dimension writeback:

- When you create a new member, you must include every attribute in a dimension. You cannot insert a member without specifying a value for the key attribute of the dimension. Therefore, creating members is subject to any constraints (such as non-null key values) that are defined on the dimension table.
- Dimension writeback is supported only for star schemas. In other words, a dimension must be based on a single dimension table directly related to a fact table. After you write-enable a dimension, Analysis Services validates this requirement when you deploy to an existing Analysis Services database or when you build an Analysis Services project.

Any existing member of a writeback dimension can be modified or deleted. When a member is deleted, the deletion cascades to all child members. For example, in a Customer dimension that contains CountryRegion, Province, City, and Customer attributes, deleting a country/region would delete all provinces, cities, and customers that belong to the deleted country/region. If a country/region has only one province, deleting that province would delete the country/region also.

Members of a writeback dimension can only be moved within the same level. For example, a city could be moved to the City level in a different country/region or province, but a city cannot be moved to the Province or CountryRegion level. In a parent-child hierarchy, all members are leaf members, and therefore a member may be moved to any level other than the **(All)** level.

If a member of a parent-child hierarchy is deleted, the member's children are moved to the member's parent. Update permissions on the relational table are required on the deleted member, but no permissions are required on the moved members. When an application moves a member in a parent-child hierarchy, the application can specify in the UPDATE operation whether descendants of the member are moved with the member or are moved to the member's parent. To recursively delete a member in a parent-child hierarchy, a user must have update permissions on the relational table for the member and all the member's descendants.



Note

Updates to the parent attribute in a parent-child hierarchy must not include updates to any other properties or attributes.

All changes to a dimension cause the dimension structure to be modified. Each change to a dimension is considered a single transaction, requiring incremental processing to update the dimension structure. Write-enabled dimensions have the same processing requirements as any other dimension.



Note

Dimension writeback is not supported by linked dimensions. For more information about linked dimensions, see [Dimensions \(Analysis Services - Multidimensional Data\)](#).

Security

The only business users who can update a write-enabled dimension are those in Analysis Services database roles that have been granted read/write permission to the dimension. For each role, you can control which members can and cannot be updated. For business users to update write-enabled dimensions, their client application must support this capability. For such users, a write-enabled dimension must be included in a cube that was processed since the dimension last changed. For more information, see [Granting User Access](#).

Users and groups included in the Administrators role can update the attribute members of a write-enabled dimension, even if the dimension is not included in a cube.

See Also

[Configuring Database Dimension Properties](#)

[Write-Enabled Partitions](#)

[Dimensions \(Analysis Services\)](#)

Dimension Translations

A translation is a simple mechanism to change the displayed labels and captions from one language to another. Each translation is defined as a pair of values: a string with the translated text, and a number with the language ID. Translations are available for all objects in Analysis Services. Dimensions can also have the attribute values translated. The client application is responsible for finding the language setting that the user has defined, and switch to display all captions and labels to that language. An object can have as many translations as you want.

A simple **T:Microsoft.AnalysisServices.Translation** object is composed of: language ID number, and translated caption. The language ID number is an **Integer** with the language ID. The translated caption is the translated text.

In Microsoft SQL Server Analysis Services, a dimension translation is a language-specific representation of the name of a dimension, the name of an Analysis Services object or one of its members, such as a caption, member, or hierarchy level. SQL Server Analysis Services also supports translations of cube objects.

Translations provide server support for client applications that can support multiple languages. Frequently, users from different countries view a cube and its dimensions. It is useful to be able to translate various elements of a cube and its dimensions into a different language so that these users can view and understand the cube. For example, a business user in France can access a cube from a workstation with a French locale setting, and see the object property values in French. However, a business user in Germany who accesses the same cube from a workstation with a German locale setting sees the same object property values in German.

The collation and language information for the client computer is stored in the form of a locale identifier (LCID). Upon connection, the client passes the LCID to the instance of Analysis Services. The instance uses the LCID to determine which set of translations to use when providing metadata for Analysis Services objects. If an Analysis Services object does not contain the specified translation, the default language is used to return the content back to the client.

See Also

[Client Applications \(Analysis Services - Multidimensional Data\)](#)

[Working with Translations \(SSAS\)](#)

[Working with Client Applications \(SSAS\)](#)

Database Dimension Properties

In Microsoft SQL Server Analysis Services, the characteristics of a dimension are defined by the metadata for the dimension, based on the settings of various dimension properties, and on the attributes or hierarchies that are contained by the dimension. The following table describes the dimension properties in Analysis Services.

Property	Description
AttributeAllMemberName	Specifies the name of the All member for attributes in a dimension.
Collation	Determines the collation used by the dimension.
CurrentStorageMode	Contains the current storage mode for the dimension.
DependsOnDimension	Contains the ID of another dimension on

Property	Description
	which the dimension depends, if any.
Description	Contains the description of the dimension.
ErrorConfiguration	Configurable error handling settings for handling of duplicate keys, unknown keys, error limits, action upon error detection, error log file, and null key handling.
ID	Contains the unique identifier (ID) of the dimension.
Language	Specifies the default language for the dimension.
MdxMissingMemberMode	Determines how missing members are handled for Multidimensional Expressions (MDX) statements.
MiningModelID	Contains the ID of the mining model with which the data mining dimension is associated. This property is applicable only if the dimension is a mining model dimension.
Name	Specifies the name of the dimension.
ProactiveCaching	Defines the proactive cache settings for the dimension.
ProcessingGroup	Specifies the processing group. Values are ByAttribute or ByTable. Default is ByAttribute .
ProcessingMode	Indicates whether Analysis Services should index and aggregate during or after processing.
ProcessingPriority	Determines the processing priority of the dimension during background operations such as lazy aggregation, indexing, or clustering.
Source	Identifies the data source view to which the dimension is bound.
StorageMode	Determines the storage mode for the dimension.

Property	Description
Type	Specifies the type of the dimension.
UnknownMember	Indicates whether the unknown member is visible.
UnknownMemberName	Specifies the caption, in the default language of the dimension, for the unknown member of the dimension.
WriteEnabled	Indicates whether dimension writebacks are available (subject to security permissions).



Note

For more information about setting values for the ErrorConfiguration and UnknownMember properties when working with null values and other data integrity issues, see [Handling Data Integrity Issues in Analysis Services 2005](#).

See Also

[Dimensions \(Analysis Services - Multidimensional Data\)](#)

[Hierarchies](#)

[Dimension Relationships and Usage](#)

[Dimensions \(SSAS\)](#)

Dimension Types

The **Type** property setting provides information about the contents of a dimension to server and client applications. In some cases, the **Type** setting only provides guidance for client applications and is optional. In other cases, such as **Accounts** or **Time** dimensions, the **Type** property settings for the dimension and its attributes determine specific server-based behaviors and may be required to implement certain behaviors in the cube. For example, the **Type** property of a dimension can be set to **Accounts** to indicate to client applications that the standard dimension contains account attributes. For more information about time, account, and currency dimensions, see [Dimensions \(Analysis Services - Multidimensional Data\)](#), [Account](#), and [Currency](#).

The default setting for the dimension type is **Regular**, which makes no assumptions about the contents of the dimension. This is the default setting for all dimensions when you initially define a dimension unless you specify **Time** when defining the dimension using the Dimension Wizard. You should also leave **Regular** as the dimension type if the Dimension Wizard does not list an appropriate type for Dimension type.

Available Dimension Types

The following table describes the dimension types available in Microsoft SQL Server Analysis Services.

Dimension type	Description
Regular	A dimension whose type has not been set to a special dimension type.
Time	A dimension whose attributes represent time periods, such as years, semesters, quarters, months, and days.
Organization	A dimension whose attributes represent organizational information, such as employees or subsidiaries.
Geography	A dimension whose attributes represent geographic information, such as cities or postal codes.
BillOfMaterials	A dimension whose attributes represent inventory or manufacturing information, such as parts lists for products.
Accounts	A dimension whose attributes represent a chart of accounts for financial reporting purposes.
Customers	A dimension whose attributes represent customer or contact information.
Products	A dimension whose attributes represent product information.
Scenario	A dimension whose attributes represent planning or strategic analysis information.
Quantitative	A dimension whose attributes represent quantitative information.
Utility	A dimension whose attributes represent miscellaneous information.
Currency	This type of dimension contains currency data and metadata.
Rates	A dimension whose attributes represent currency rate information.

Dimension type	Description
Channel	A dimension whose attributes represent channel information.
Promotion	A dimension whose attributes represent marketing promotion information.

See Also

[Creating a Standard Dimension](#)

[Dimensions \(SSAS\)](#)

Proactive Caching (Dimensions)

Proactive caching provides automatic MOLAP cache creation and management for OLAP objects. The cubes immediately incorporate changes that are made to the data in the database, based upon notifications received from the database. The goal of proactive caching is to provide the performance of traditional MOLAP, while retaining the immediacy and ease of management offered by ROLAP.

A simple **T:Microsoft.AnalysisServices.ProactiveCaching** object is composed of: timing specification, and table notification. The timing specification defines the timeframe for updating the cache after a change notification has been received. The table notification defines the notification schema between the data table and the **T:Microsoft.AnalysisServices.ProactiveCaching** object.

Cube Objects

Introducing Cube Objects

A simple **T:Microsoft.AnalysisServices.Cube** object is composed of: basic information, dimensions, and measure groups. Basic information includes the name of the cube, the default measure of the cube, the data source, the storage mode, and others.

The Dimensions collection contains the actual set of dimensions used in the cube from the database dimensions collection. All dimensions have to be defined in the dimensions collection of the database before being referenced in the cube. Private dimensions are not available in Microsoft SQL Server Analysis Services.

Measure groups are sets of measures in the cube. A measure group is a collection of measures that have a common data source view and a common set of dimensions. A measure group is the unit of process for measures; measure groups can be processed individually and then browsed.

In this section

Topic	

Actions (Analysis Services - Multidimensional Data)	
Aggregations and Aggregation Designs	
Calculations	
Cube Cells (Analysis Services - Multidimensional Data)	
Cube Properties	
Cube Storage (Analysis Services - Multidimensional Data)	
Cube Translations	
Dimension Relationships	
Key Performance Indicators (KPIs)	
Measures and Measure Groups	
Partitions (Analysis Services - Multidimensional Data)	
Perspectives	

Cube Properties

Cubes have a number of properties that you can set to affect cube-wide behavior. These properties are summarized in the following table.



Note

Some properties are set automatically when the cube is created and cannot be changed. For more information about how to set cube properties, see [Proactive Caching \(Partitions\)](#).

Property	Description
AggregationPrefix	Specifies the common prefix that is used for aggregation names.
Collation	Specifies the locale identifier (LCID) and the comparison flag, separated by an underscore: for example, Latin1_General_C1_AS.
DefaultMeasure	Contains a Multidimensional Expressions

Property	Description
	(MDX) expression that defines the default measure for the cube.
Description	Provides a description of the cube, which may be exposed in client applications.
ErrorConfiguration	Contains configurable error handling settings for handling of duplicate keys, unknown keys, error limits, action upon error detection, error log file, and null key handling.
EstimatedRows	Specifies the number of estimated rows in the cube.
ID	Contains the unique identifier (ID) of the cube.
Language	Specifies the default language identifier of the cube.
Name	Specifies the user-friendly name of the cube.
ProactiveCaching	Defines proactive cache settings for the cube.
ProcessingMode	Indicates whether indexing and aggregating should occur during or after processing. Options are regular or lazy .
ProcessingPriority	Determines the processing priority of the cube during background operations, such as lazy aggregations and indexing. The default value is 0 .
ScriptCacheProcessingMode	Indicates whether the script cache should be built during or after processing. Options are regular and lazy .
ScriptErrorHandlingMode	Determines error handling. Options are IgnoreNone or IgnoreAll
Source	Displays the data source view used for the cube.
StorageLocation	Specifies the file system storage location for the cube. If none is specified, the

Property	Description
	location is inherited from the database that contains the cube object.
StorageMode	Specifies the storage mode for the cube. Values are MOLAP , ROLAP , or HOLAP .
Visible	Determines the visibility of the cube.



Note

For more information about setting values for the ErrorConfiguration property when working with null values and other data integrity issues, see [Handling Data Integrity Issues in Analysis Services 2005](#).

See Also

[Proactive Caching \(SSAS\)](#)

Dimension Relationships

Dimension usage defines the relationships between a cube dimension and the measure groups in a cube. A cube dimension is an instance of a database dimension that is used in a specific cube. A cube can, and frequently does, have cube dimensions that are not directly related to a measure group, but which might be indirectly related to the measure group through another dimension or measure group. When you add a database dimension or measure group to a cube, Microsoft SQL Server Analysis Services tries to determine dimension usage by examining relationships between the dimension tables and fact tables in the cube's data source view, and by examining the relationships between attributes in dimensions. Analysis Services automatically sets the dimension usage settings for the relationships that it can detect.

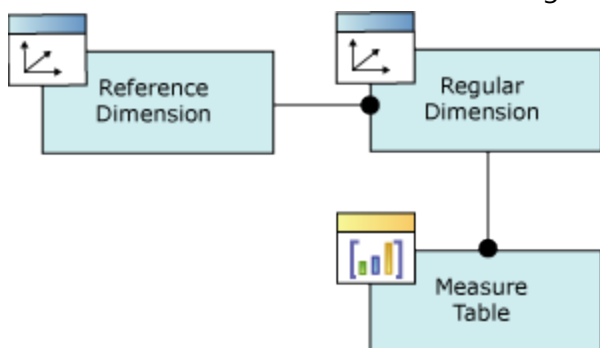
A relationship between a dimension and a measure group consists of the dimension and fact tables participating in the relationship and a granularity attribute that specifies the granularity of the dimension in the particular measure group.

Regular Dimension Relationships

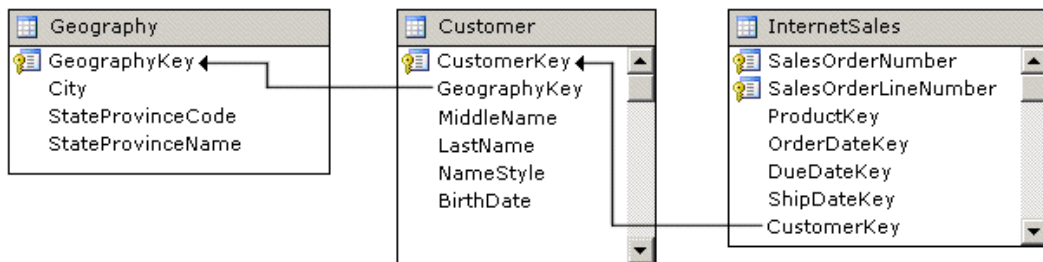
A regular dimension relationship between a cube dimension and a measure group exists when the key column for the dimension is joined directly to the fact table. This direct relationship is based on a primary key–foreign key relationship in the underlying relational database, but might also be based on a logical relationship that is defined in the data source view. A regular dimension relationship represents the relationship between dimension tables and a fact table in a traditional star schema design. For more information about regular relationships, see [Defining Dimension Usage Relationships](#).

Reference Dimension Relationships

A reference dimension relationship between a cube dimension and a measure group exists when the key column for the dimension is joined indirectly to the fact table through a key in another dimension table, as shown in the following illustration.



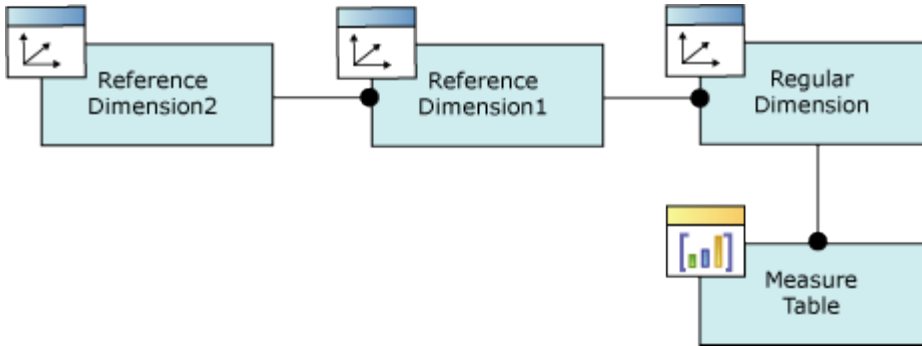
A reference dimension relationship represents the relationship between dimension tables and a fact table in a snowflake schema design. When dimension tables are connected in a snowflake schema, you can define a single dimension using columns from multiple tables, or you can define separate dimensions based on the separate dimension tables and then define a link between them using the reference dimension relationship setting. The following figure shows one fact table named **InternetSales**, and two dimension tables called **Customer** and **Geography**, in a snowflake schema.



You can create a dimension with the **Customer** table as the dimension main table and the **Geography** table included as a related table. A regular relationship is then defined between the dimension and the InternetSales measure group.

Alternatively, you can create two dimensions related to the InternetSales measure group: a dimension based on the **Customer** table, and a dimension based on the **Geography** table. You can then relate the Geography dimension to the InternetSales measure group using a reference dimension relationship using the Customer dimension. In this case, when the facts in the InternetSales measure group are dimensioned by the Geography dimension, the facts are dimensioned by customer and by geography. If the cube contained a second measure group named Reseller Sales, you would be unable to dimension the facts in the Reseller Sales measure group by Geography because no relationship would exist between Reseller Sales and Geography.

There is no limit to the number of reference dimensions that can be chained together, as shown in the following illustration.



For more information about referenced relationships, see [Defining a Referenced Relationship and Referenced Relationship Properties](#).

Fact Dimension Relationships

Fact dimensions, frequently referred to as degenerate dimensions, are standard dimensions that are constructed from attribute columns in fact tables instead of from attribute columns in dimension tables. Useful dimensional data is sometimes stored in a fact table to reduce duplication. For example, the following diagram displays the **FactResellerSales** fact table, from the Adventure Works DW Multidimensional 2012 sample database.

FactResellerSales (dbo.FactResellerSales)	
	SalesOrderNumber
	SalesOrderLineNumber
	ProductKey
	OrderDateKey
	DueDateKey
	ShipDateKey
	ResellerKey
	EmployeeKey
	PromotionKey
	CurrencyKey
	SalesTerritoryKey
	RevisionNumber
	OrderQuantity
	UnitPrice
	ExtendedAmount
	UnitPriceDiscountPct
	DiscountAmount
	ProductStandardCost
	TotalProductCost
	SalesAmount
	TaxAmt
	Freight
	CarrierTrackingNumber
	CustomerPONumber

The table contains attribute information not only for each line of an order issued by a reseller, but about the order itself. The attributes circled in the previous diagram identify the information in the **FactResellerSales** table that could be used as attributes in a dimension. In this case, two additional pieces of information, the carrier tracking number and the purchase order number issued by the reseller, are represented by the CarrierTrackingNumber and CustomerPONumber attribute columns. This information is interesting—for example, users would definitely be interested in seeing aggregated information, such as the total product cost, for all the orders being shipped under a single tracking number. But, without a dimension data for these two attributes cannot be organized or aggregated.

In theory, you could create a dimension table that uses the same key information as the FactResellerSales table and move the other two attribute columns, CarrierTrackingNumber and CustomerPONumber, to that dimension table. However, you would be duplicating a significant portion of data and adding unnecessary complexity to the data warehouse to represent just two attributes as a separate dimension.



Note

Fact dimensions are frequently used to support drillthrough actions. For more information about actions, see [Actions](#).

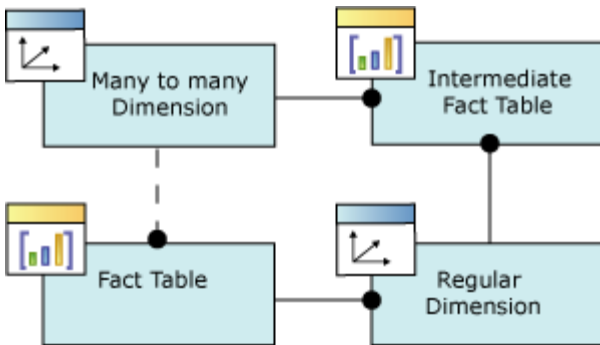
Note

Fact dimensions must be incrementally updated after every update to the measure group that is referenced by the fact relationship. If the fact dimension is a ROLAP dimension, the Analysis Services processing engine drops any caches and incrementally processes the measure group.

For more information about fact relationships, see [Defining a Fact Relationship and Fact Relationship Properties](#).

Many to Many Dimension Relationships

In most dimensions, each fact joins to one and only one dimension member, and a single dimension member can be associated with multiple facts. In relational database terminology, this is referred to as a one-to-many relationship. However, it is frequently useful to join a single fact to multiple dimension members. For example, a bank customer might have multiple accounts (checking, saving, credit card, and investment accounts), and an account can also have joint or multiple owners. The Customer dimension constructed from such relationships would then have multiple members that relate to a single account transaction.



SQL Server Analysis Services lets you define a many-to-many relationship between a dimension and a fact table.

Note

To support a many-to-many dimension relationship, the data source view must have established a foreign key relationship between all the tables involved, as shown in the previous diagram. Otherwise, you will be unable to select the correct intermediate measure group when establishing the relationship in the **Dimension Usage** tab of Dimension Designer.

For more information about many-to-many relationships, see [Defining a Many-to-Many Relationship and Many-to-Many Relationship Properties](#).

See Also

[Dimensions](#)

[Defining and Configuring Dimension Usage and Dimension Relationships](#)

Calculations

A calculation is a Multidimensional Expressions (MDX) expression or script that is used to define a calculated member, a named set, or a scoped assignment in a cube in Microsoft SQL Server Analysis Services. Calculations let you add objects that are defined not by the data of the cube, but by expressions that can reference other parts of the cube, other cubes, or even information outside the Analysis Services database. Calculations let you extend the capabilities of a cube, adding flexibility and power to business intelligence applications. For more information about scripting calculations, see [Introduction to MDX Scripting in Microsoft SQL Server 2005](#). For more information about performance issues related to MDX queries and calculations, see the [SQL Server 2005 Analysis Services Performance Guide](#).

Calculated Members

A calculated member is a member whose value is calculated at run time using a Multidimensional Expressions (MDX) expression that you specify when you define the calculated member. A calculated member is available to business intelligence applications just like any other member. Calculated members do not increase the size of the cube because only the definitions are stored in the cube; values are calculated in memory as required to answer a query.

Calculated members can be defined for any dimension, including the measures dimension. Calculated members created on the Measures dimension are called calculated measures.

Although calculated members are typically based on data that already exists in the cube, you can create complex expressions by combining data with arithmetic operators, numbers, and functions. You can also use MDX functions, such as `LookupCube`, to access data in other cubes in the Analysis Services database. Analysis Services includes standardized Visual Studio function libraries, and you can use stored procedures to retrieve data from sources other than the current Analysis Services database. For more information about stored procedures, see [Defining Calculations](#).

For example, suppose executives in a shipping company want to determine which types of cargo are more profitable to carry, based on profit per unit of volume. They use a Shipments cube that contains the dimensions Cargo, Fleet, and Time and the measures Price_to_Ship, Cost_to_Ship, and Volume_in_Cubic_Meters; however, the cube does not contain a measure for profitability. You can create a calculated member as a measure named Profit_per_Cubic_Meter in the cube by combining the existing measures in the following expression:

```
([Measures].[Price_to_Ship] - [Measures].[Cost_to_Ship]) /  
[Measures].[Volume_in_Cubic_Meters]
```

After you create the calculated member, the Profit_per_Cubic_Meter appears together with the other measures the next time that the Shipments cube is browsed.

To create calculated members, use the **Calculations** tab in Cube Designer. For more information, see [Defining a Calculated Member](#)

Named Sets

A named set is a CREATE SET MDX statement expression that returns a set. The MDX expression is saved as part of the definition of a cube in Microsoft SQL Server Analysis Services. A named set is created for reuse in Multidimensional Expressions (MDX) queries. A named set enables business users to simplify queries, and use a set name instead of a set expression for complex, frequently used set expressions. **Related topic:** [Defining a Named Set](#)

Script Commands

A script command is an MDX script, included as part of the definition of the cube. Script commands let you perform almost any action that is supported by MDX on a cube, such as scoping a calculation to apply to only part of the cube. In SQL Server Analysis Services, MDX scripts can apply either to the whole cube or to specific sections of the cube, at specific points throughout the execution of the script. The default script command, which is the CALCULATE statement, populates cells in the cube with aggregated data based on the default scope.

The default scope is the whole cube, but you can define a more limited scope, known as a subcube, and then apply an MDX script to only that particular cube space. The SCOPE statement defines the scope of all subsequent MDX expressions and statements in the calculation script until the scope is terminated or redefined. The THIS statement is then used to apply an MDX expression to the current scope. You can use the BACK_COLOR statement to specify a background cell color for the cells in the current scope, to help you during debugging.

For example, you can use a script command to allocate sales quotas to employees across time and sales territory based on the weighted values of sales for a prior time period.

See Also

[Defining and Configuring a Calculation](#)

Partitions

A partition is a container for a portion of the measure group data. Partitions are not seen from MDX queries; all queries reflect the whole content of the measure group, regardless of how many partitions are defined for the measure group. The data content of a partition is defined by the query bindings of the partition, and by the slicing expression.

A simple **T:Microsoft.AnalysisServices.Partition** object is composed of: basic information, slicing definition, aggregation design, and others. Basic information includes the name of the partition, the storage mode, the processing mode, and others. The slicing definition is an MDX expression specifying a tuple or a set. The slicing definition has the same restrictions as the StrToSet MDX function. Together with the CONSTRAINED parameter, the slicing definition can use dimension, hierarchy, level and member names, keys, unique names, or other named objects in the cube, but cannot use MDX functions. The aggregation design is a collection of aggregation definitions that can be shared across multiple partitions. The default is taken from the parent cube's aggregation design.

Partitions are used by Microsoft SQL Server Analysis Services to manage and store data and aggregations for a measure group in a cube. Every measure group has at least one partition; this partition is created when the measure group is defined. When you create a new partition for a

measure group, the new partition is added to the set of partitions that already exist for the measure group. The measure group reflects the combined data that is contained in all its partitions. This means that you must ensure that the data for a partition in a measure group is exclusive of the data for any other partition in the measure group to ensure that data is not reflected in the measure group more than once. The original partition for a measure group is based on a single fact table in the data source view of the cube. When there are multiple partitions for a measure group, each partition can reference a different table in either the data source view or in the underlying relational data source for the cube. More than one partition in a measure group can reference the same table, if each partition is restricted to different rows in the table.

Partitions are a powerful and flexible means of managing cubes, especially large cubes. For example, a cube that contains sales information can contain a partition for the data of each past year and also partitions for each quarter of the current year. Only the current quarter partition needs to be processed when current information is added to the cube; processing a smaller amount of data will improve processing performance by decreasing processing time. At the end of the year the four quarterly partitions can be merged into a single partition for the year and a new partition created for the first quarter of the new year. Further, this new partition creation process can be automated as part of your data warehouse loading and cube processing procedures.

Partitions are not visible to business users of the cube. However, administrators can configure, add, or drop partitions. Each partition is stored in a separate set of files. The aggregate data of each partition can be stored on the instance of Analysis Services where the partition is defined, on another instance of Analysis Services, or in the data source that is used to supply the partition's source data. Partitions allow the source data and aggregate data of a cube to be distributed across multiple hard drives and among multiple server computers. For a cube of moderate to large size, partitions can greatly improve query performance, load performance, and ease of cube maintenance. **For more information about remote partitions, see [Analysis Services Multidimensional Database Logical Architecture-Delete](#).**

The storage mode of each partition can be configured independently of other partitions in the measure group. Partitions can be stored by using any combination of options for source data location, storage mode, proactive caching, and aggregation design. Options for real-time OLAP and proactive caching let you balance query speed against latency when you design a partition. Storage options can also be applied to related dimensions and to facts in a measure group. This flexibility lets you design cube storage strategies appropriate to your needs. For more information, see [Partition Storage](#), [Aggregations](#) and [Proactive Caching](#).

Partition Structure

The structure of a partition must match the structure of its measure group, which means that the measures that define the measure group must also be defined in the partition, along with all related dimensions. Therefore, when a partition is created, it automatically inherits the same set of measures and related dimensions that were defined for the measure group.

However, each partition in a measure group can have a different fact table, and these fact tables can be from different data sources. When different partitions in a measure group have different

fact tables, the tables must be sufficiently similar to maintain the structure of the measure group, which means that the processing query returns the same columns and same data types for all fact tables for all partitions.

When fact tables for different partitions are from different data sources, the source tables for any related dimensions, and also any intermediate fact tables, must also be present in all data sources and must have the same structure in all the databases. Also, all dimension table columns that are used to define attributes for cube dimensions related to the measure group must be present in all of the data sources. There is no need to define all the joins between the source table of a partition and a related dimension table if the partition source table has the identical structure as the source table for the measure group.

Columns that are not used to define measures in the measure group can be present in some fact tables but absent in others. Similarly, columns that are not used to define attributes in related dimension tables can be present in some databases but absent in others. Tables that are not used for either fact tables or related dimension tables can be present in some databases but absent in others.

Data Sources and Partition Storage

A partition is based either on a table or view in a data source, or on a table or named query in a data source view. The location where partition data is stored is defined by the data source binding. Typically, you can partition a measure group horizontally or vertically:

- In a horizontally partitioned measure group, each partition in a measure group is based on a separate table. This kind of partitioning is appropriate when data is separated into multiple tables. For example, some relational databases have a separate table for each month's data.
- In a vertically partitioned measure group, a measure group is based on a single table, and each partition is based on a source system query that filters the data for the partition. For example, if a single table contains several months data, the measure group could still be partitioned by month by applying a Transact-SQL WHERE clause that returns a separate month's data for each partition.

Each partition has storage settings that determine whether the data and aggregations for the partition are stored in the local instance of Analysis Services or in a remote partition using another instance of Analysis Services. The storage settings can also specify the storage mode and whether proactive caching is used to control latency for a partition. For more information, see [Partition Storage Modes \(Analysis Services\)](#), [Proactive Caching](#), and [Remote Partitions](#).

Incremental Updates

When you create and manage partitions in multiple-partition measure groups, you must take special precautions to guarantee that cube data is accurate. Although these precautions do not usually apply to single-partition measure groups, they do apply when you incrementally update partitions. When you incrementally update a partition, a new temporary partition is created that has a structure identical to that of the source partition. The temporary partition is processed and then merged with the source partition. Therefore, you must ensure that the processing query that populates the temporary partition does not duplicate any data already present in an existing partition. For more information, see [Defining and Configuring a Partition](#).

See Also

[Measure Groups \(Analysis Services\)](#)

[Defining and Configuring a Partition](#)

[Cubes \(Analysis Services\)](#)

Partition Storage Modes and Processing

The storage mode of a partition affects the query and processing performance, storage requirements, and storage locations of the partition and its parent measure group and cube. The choice of storage mode also affects processing choices.

A partition can use one of three basic storage modes:

- Multidimensional OLAP (MOLAP)
- Relational OLAP (ROLAP)
- Hybrid OLAP (HOLAP)

Microsoft SQL Server Analysis Services supports all three basic storage modes. It also supports proactive caching, which enables you to combine the characteristics of ROLAP and MOLAP storage for both immediacy of data and query performance. For more information, see [Partitions \(Analysis Services - Multidimensional Data\)](#).

MOLAP

The MOLAP storage mode causes the aggregations of the partition and a copy of its source data to be stored in a multidimensional structure in Analysis Services when the partition is processed. This MOLAP structure is highly optimized to maximize query performance. The storage location can be on the computer where the partition is defined or on another computer running Analysis Services. Because a copy of the source data resides in the multidimensional structure, queries can be resolved without accessing the partition's source data. Query response times can be decreased substantially by using aggregations. The data in the partition's MOLAP structure is only as current as the most recent processing of the partition.

As the source data changes, objects in MOLAP storage must be processed periodically to incorporate those changes and make them available to users. Processing updates the data in the MOLAP structure, either fully or incrementally. The time between one processing and the next creates a latency period during which data in OLAP objects may not match the source data. You can incrementally or fully update objects in MOLAP storage without taking the partition or cube offline. However, there are situations that may require you to take a cube offline to process certain structural changes to OLAP objects. You can minimize the downtime required to update MOLAP storage by updating and processing cubes on a staging server and using database synchronization to copy the processed objects to the production server. You can also use proactive caching to minimize latency and maximize availability while retaining much of the performance advantage of MOLAP storage. For more information, see [Proactive Caching \(SSAS\)](#), [Synchronizing Analysis Services Databases](#), and [Processing Analysis Services Objects](#).

ROLAP

The ROLAP storage mode causes the aggregations of the partition to be stored in indexed views in the relational database that was specified in the partition's data source. Unlike the MOLAP storage mode, ROLAP does not cause a copy of the source data to be stored in the Analysis Services data folders. Instead, when results cannot be derived from the query cache, the indexed views in the data source is accessed to answer queries. Query response is generally slower with ROLAP storage than with the MOLAP or HOLAP storage modes. Processing time is also typically slower with ROLAP. However, ROLAP enables users to view data in real time and can save storage space when you are working with large datasets that are infrequently queried, such as purely historical data.



Note

When using ROLAP, Analysis Services may return incorrect information related to the unknown member if a join is combined with a GROUP BY clause. Analysis Services eliminates relational integrity errors instead of returning the unknown member value.

If a partition uses the ROLAP storage mode and its source data is stored in SQL Server Database Engine, Analysis Services tries to create indexed views to contain aggregations of the partition. If Analysis Services cannot create indexed views, it does not create aggregation tables. Although Analysis Services handles the session requirements for creating indexed views on SQL Server Database Engine, the following conditions must be met by the ROLAP partition and the tables in its schema in order for Analysis Services to create indexed views for aggregations:

- The partition cannot contain measures that use the **Min** or **Max** aggregate functions.
- Each table in the schema of the ROLAP partition must be used only one time. For example, the schema cannot contain [dbo].[address] AS "Customer Address" and [dbo].[address] AS "SalesRep Address".
- Each table must be a table, not a view.
- All table names in the partition's schema must be qualified with the owner name, for example, [dbo].[customer].
- All tables in the partition's schema must have the same owner; for example, you cannot have a FROM clause that references the tables [tk].[customer], [john].[store], and [dave].[sales_fact_2004].
- The source columns of the partition's measures must not be nullable.
- All tables used in the view must have been created with the following options set to ON:
 - ANSI_NULLS
 - QUOTED_IDENTIFIER
- The total size of the index key, in SQL Server Database Engine, cannot exceed 900 bytes. SQL Server Database Engine will assert this condition based on the fixed length key columns when the CREATE INDEX statement is processed. However, if there are variable length columns in the index key, SQL Server Database Engine will also assert this condition for every update to the base tables. Because different aggregations have different view definitions, ROLAP processing using indexed views can succeed or fail depending on the aggregation design.

- The session creating the indexed view must have the following options set to ON: ARITHABORT, CONCAT_NULL_YIELDS_NULL, QUOTED_IDENTIFIER, ANSI_NULLS, ANSI_PADDING, and ANSI_WARNING. This setting can be made in SQL Server Management Studio.
- The session creating the indexed view must have the following option set to OFF: NUMERIC_ROUNDABORT. This setting can be made in SQL Server Management Studio.

HOLAP

The HOLAP storage mode combines attributes of both MOLAP and ROLAP. Like MOLAP, HOLAP causes the aggregations of the partition to be stored in a multidimensional structure in an SQL Server Analysis Services instance. HOLAP does not cause a copy of the source data to be stored. For queries that access only summary data in the aggregations of a partition, HOLAP is the equivalent of MOLAP. Queries that access source data—for example, if you want to drill down to an atomic cube cell for which there is no aggregation data—must retrieve data from the relational database and will not be as fast as they would be if the source data were stored in the MOLAP structure. With HOLAP storage mode, users will typically experience substantial differences in query times depending upon whether the query can be resolved from cache or aggregations versus from the source data itself.

Partitions stored as HOLAP are smaller than the equivalent MOLAP partitions because they do not contain source data and respond faster than ROLAP partitions for queries involving summary data. HOLAP storage mode is generally suited for partitions in cubes that require rapid query response for summaries based on a large amount of source data. However, where users generate queries that must touch leaf level data, such as for calculating median values, MOLAP is generally a better choice.

See Also

[Designing Partition Storage and Aggregations](#)

[Proactive Caching](#)

[Synchronizing Analysis Services Databases](#)

[Partitions](#)

Proactive Caching (Partitions)

Proactive caching provides automatic MOLAP cache creation and management for OLAP objects. The cubes immediately incorporate changes that are made to the data in the database, based upon notifications received from the database. The goal of proactive caching is to provide the performance of traditional MOLAP, while retaining the immediacy and ease of management offered by ROLAP.

A simple **T:Microsoft.AnalysisServices.ProactiveCaching** object is composed of: timing specification, and table notification. The timing specification defines the timeframe for updating the cache after a change notification has been received. The table notification defines the notification schema between the data table and the

T:Microsoft.AnalysisServices.ProactiveCaching object.

Multidimensional OLAP (MOLAP) storage provides the best query response, but with a penalty of some data latency. Real-time relational OLAP (ROLAP) storage lets users immediately browse the most recent changes in a data source, but at the penalty of significantly poorer performance than multidimensional OLAP (MOLAP) storage because of the absence of precalculated summaries of data and because relational storage is not optimized for OLAP-style queries. If you have applications in which your users need to see recent data and you also want the performance advantages of MOLAP storage, SQL Server Analysis Services offers the option of proactive caching to address this scenario, particularly in combination with the use of partitions. Proactive caching is set on a per partition and per dimension basis. Proactive caching options can provide a balance between the enhanced performance of MOLAP storage and the immediacy of ROLAP storage, and provide automatic partition processing when underlying data changes or on a set schedule.

Proactive Caching Configuration Options

SQL Server Analysis Services provides several proactive caching configuration options that enable you to maximize performance, minimize latency, and schedule processing. Proactive caching features simplify the process of managing data obsolescence. The proactive caching settings determine how frequently the multidimensional OLAP structure, also called the MOLAP cache, is rebuilt, whether the outdated MOLAP storage is queried while the cache is rebuilt or the underlying ROLAP data source, and whether the cache is rebuilt on a schedule or based on changes in the database.

Minimizing Latency

With proactive caching set to minimize latency, user queries against an OLAP object are made against either ROLAP storage or MOLAP storage, depending whether recent changes have occurred to the data and how proactive caching is configured. The query engine directs queries against source data in MOLAP storage until changes occur in the data source. To minimize latency, after changes occur in a data source, cached MOLAP objects can be dropped and querying switched to ROLAP storage while the MOLAP objects are rebuilt in cache. After the MOLAP objects are rebuilt and processed, queries are automatically switched to the MOLAP storage. The cache refresh can occur extremely quickly for a small partition, such as the current partition - which can be as small as the current day.

Maximizing Performance

To maximize performance while also reducing latency, caching can also be used without dropping the current MOLAP objects. Queries then continue against the MOLAP objects while data is read into and processed in a new cache. This method provides better performance but may result in queries returning old data while the new cache is being built.

See Also

[Choosing a Standard Storage Setting](#)

[Choosing a Standard Storage Setting](#)

Remote Partitions

The data of a remote partition is stored on a different instance of Microsoft SQL Server Analysis Services than the instance that contains the definitions (metadata) of the partition and its parent cube. A remote partition is administered on the same instance of Analysis Services where the partition and its parent cube are defined.

Note

To store a remote partition, the computer must have an instance of SQL Server Analysis Services installed and be running the same service pack level as the instance where the partition was defined. Remote partitions on instances of an earlier version of Analysis Services are not supported.

When remote partitions are included in a measure group, the memory and CPU utilization of the cube is distributed across all the partitions in the measure group. For example, when a remote partition is processed, either alone or as part of parent cube processing, most of the memory and CPU utilization for that partition occurs on the remote instance of Analysis Services.

Note

A cube that contains remote partitions can contain write-enabled dimensions; however, this may affect performance for the cube. For more information about write-enabled dimensions, see [Processing \(Analysis Services - Multidimensional Data\)](#).

Storage Modes for Remote Partitions

Remote partitions may use any of the storage types used by local partitions: multidimensional OLAP (MOLAP), hybrid OLAP (HOLAP), or relational OLAP (ROLAP). Remote partitions may also use proactive caching. Depending on the storage mode of a remote partition, the following data is stored on the remote instance of Analysis Services.

Storage Type	Data
MOLAP	The partition's aggregations and a copy of the partition's source data
HOLAP	The partitions aggregations
ROLAP	No partition data

If a measure group contains multiple MOLAP or HOLAP partitions stored on multiple instances of Analysis Services, the cube distributes the data in the measure group data among those instances of Analysis Services.

Merging Remote Partitions

Remote partitions can be merged only with other remote partitions that are stored on the same remote instance of Analysis Services. For more information about merging partitions, see [Merging Partitions](#).

Archiving and Restoring Remote Partitions

Data in remote partitions can be archived or restored when the database that stores the remote partition is archived or restored. If you restore a database without restoring a remote partition, you must process the remote partition before you can use the data in the partition. For more information about archiving and restoring databases, see [Backing Up and Restoring an Analysis Services Database](#).

See Also

[Creating and Managing a Remote Partition](#)

[Processing OLAP Objects](#)

Write-Enabled Partitions

The data in a cube is generally read-only. However, for certain scenarios, you may want to write-enable a partition. Write-enabled partitions are used to enable business users to explore scenarios by changing cell values and analyzing the effects of the changes on cube data. When you write-enable a partition, client applications can record changes to the data in the partition. These changes, known as writeback data, are stored in a separate table and do not overwrite any existing data in a measure group. However, they are incorporated into query results as if they are part of the cube data.

You can write-enable an entire cube or only certain partitions in the cube. Write-enabled dimensions are different but complementary. A write-enabled partition lets users update partition cells, whereas a write-enabled dimension lets users update dimension members. You can also use these two features in combination. For example, a write-enabled cube or a write-enabled partition does not have to include any write-enabled dimensions. **Related**

topic: [Defining Write-Enabled Dimensions](#).



Note

If you want to write-enable a cube that has a Microsoft Access database as a data source, do not use Microsoft OLE DB Provider for ODBC Drivers in the data source definitions for the cube, its partitions, or its dimensions. Instead, you can use Microsoft Jet 4.0 OLE DB Provider, or any version of the Jet Service Pack that includes Jet 4.0 OLE. For more information, see the Microsoft Knowledge Base article [How to obtain the latest service pack for the Microsoft Jet 4.0 Database Engine](#).

A cube can be write-enabled only if all its measures use the **Sum** aggregate function. Linked measure groups and local cubes cannot be write-enabled.

Writeback Storage

Any change made by the business user is stored in the writeback table as a difference from the currently displayed value. For example, if an end user changes a cell value from 90 to 100, the value **+10** is stored in the writeback table, together with the time of the change and information about the business user who made it. The net effect of accumulated changes is displayed to client applications. The original value in the cube is preserved, and an audit trail of changes is recorded in the writeback table.

Changes to leaf and nonleaf cells are handled differently. A leaf cell represents an intersection of a measure and a leaf member from every dimension referenced by the measure group. The value of a leaf cell is taken directly from the fact table, and cannot be divided further by drilling down. If a cube or any partition is write-enabled, changes can be made to a leaf cell. Changes can be made to a nonleaf cell only if the client application provides a way of distributing the changes among the leaf cells that make up the nonleaf cell. This process, called allocation, is managed through the UPDATE CUBE statement in Multidimensional Expressions (MDX). Business intelligence developers can use the UPDATE CUBE statement to include allocation functionality. For more information, see [UPDATE CUBE Statement](#).



Important

When updated cells do not overlap, the **Update Isolation Level** connection string property can be used to enhance performance for UPDATE CUBE. For more information, see **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString**.

Regardless of whether a client application distributes changes that were made to nonleaf cells, whenever queries are evaluated, changes in the writeback table are applied to both leaf and nonleaf cells so that business users can view the effects of the changes throughout the cube.

Changes that were made by the business user are kept in a separate writeback table that you can work with as follows:

- Convert to a partition to permanently incorporate changes into the cube. This action makes the measure group read-only. You can specify a filter expression to select the changes you want to convert.
- Discard to return the partition to its original state. This action makes the partition read-only.

Security

A business user is permitted to record changes in a cube's writeback table only if the business user belongs to a role that has read/write permission to the cube's cells. For each role, you can control which cube cells can and cannot be updated. For more information, see [Granting Cube Access](#).

See Also

[Write-Enabled Dimensions \(Analysis Services\)](#)

[Aggregations \(Analysis Services\)](#)

[Partitions \(Analysis Services\)](#)

[Defining Write-Enabled Dimensions](#)

Perspectives

A perspective is a definition that allows users to see a cube in a simpler way. A perspective is a subset of the features of a cube. A perspective enables administrators to create views of a cube, helping users to focus on the most relevant data for them. A perspective contains subsets of all objects from a cube. A perspective cannot include elements that are not defined in the parent cube.

A simple **T:Microsoft.AnalysisServices.Perspective** object is composed of: basic information, dimensions, measure groups, calculations, KPIs, and actions. Basic information includes the name and the default measure of the perspective. The dimensions are a subset of the cube dimensions. The measure groups are a subset of the cube measure groups. The calculations are a subset of the cube calculations. The KPIs are a subset of the cube KPIs. The actions are a subset of the cube actions.

A cube has to be updated and processed before the perspective can be used.

Cubes can be very complex objects for users to explore in Microsoft SQL Server Analysis Services. A single cube can represent the contents of a complete data warehouse, with multiple measure groups in a cube representing multiple fact tables, and multiple dimensions based on multiple dimension tables. Such a cube can be very complex and powerful, but daunting to users who may only need to interact with a small part of the cube in order to satisfy their business intelligence and reporting requirements.

In Microsoft SQL Server Analysis Services, you can use a perspective to reduce the perceived complexity of a cube in Analysis Services. A perspective defines a viewable subset of a cube that provides focused, business-specific or application-specific viewpoints on the cube. The perspective controls the visibility of objects that are contained by a cube. The following objects can be displayed or hidden in a perspective:

- Dimensions
- Attributes
- Hierarchies
- Measure groups
- Measures
- Key Performance Indicators (KPIs)
- Calculations (calculated members, named sets, and script commands)
- Actions

For example, the **Adventure Works** cube in the Adventure Works DW Multidimensional 2012 sample Analysis Services database contains eleven measure groups and twenty-one different cube dimensions, representing sales, sales forecasting, and financial data. A client application can directly reference the complete cube, but this viewpoint may be overwhelming to a user trying to extract basic sales forecasting information. Instead, the same user can use the **Sales Targets** perspective to limit the view of the **Adventure Works** cube to only those objects relevant to sales forecasting.

Objects in a cube that are not visible to the user through a perspective can still be directly referenced and retrieved using XML for Analysis (XMLA), Multidimensional Expressions (MDX), or Data Mining Extensions (DMX) statements. Perspectives do not restrict access to objects in a cube and should not be used as such; instead, perspectives are used to provide a better user experience while accessing a cube.

A perspective is a read-only view of the cube; objects in the cube cannot be renamed or changed by using a perspective. Similarly, the behavior or features of a cube, such as the use of visual totals, cannot be changed by using a perspective.

Security

Perspectives are not meant to be used as a security mechanism, but as a tool for providing a better user experience in business intelligence applications. All security for a particular perspective is inherited from the underlying cube. For example, perspectives cannot provide access to objects in a cube to which a user does not already have access. - Security for the cube must be resolved before access to objects in the cube can be provided through a perspective.

Related topic: [Security and Protection](#)

See Also

[Working with Perspectives in Model Designer](#)

Cube Translations

A translation is a simple mechanism to change the displayed labels and captions from one language to another. Each translation is defined as a pair of values: a string with the translated text, and a number with the language ID. Translations are available for all objects in Analysis Services. Dimensions can also have the attribute values translated. The client application is responsible for finding the language setting that the user has defined, and switch to display all captions and labels to that language. An object can have as many translations as you want.

A simple **T:Microsoft.AnalysisServices.Translation** object is composed of: language ID number, and translated caption. The language ID number is an **Integer** with the language ID. The translated caption is the translated text.

In Microsoft SQL Server Analysis Services, a cube translation is a language-specific representation of the name of a cube object, such as a caption or a display folder. Analysis Services also supports translations of dimension and member names.

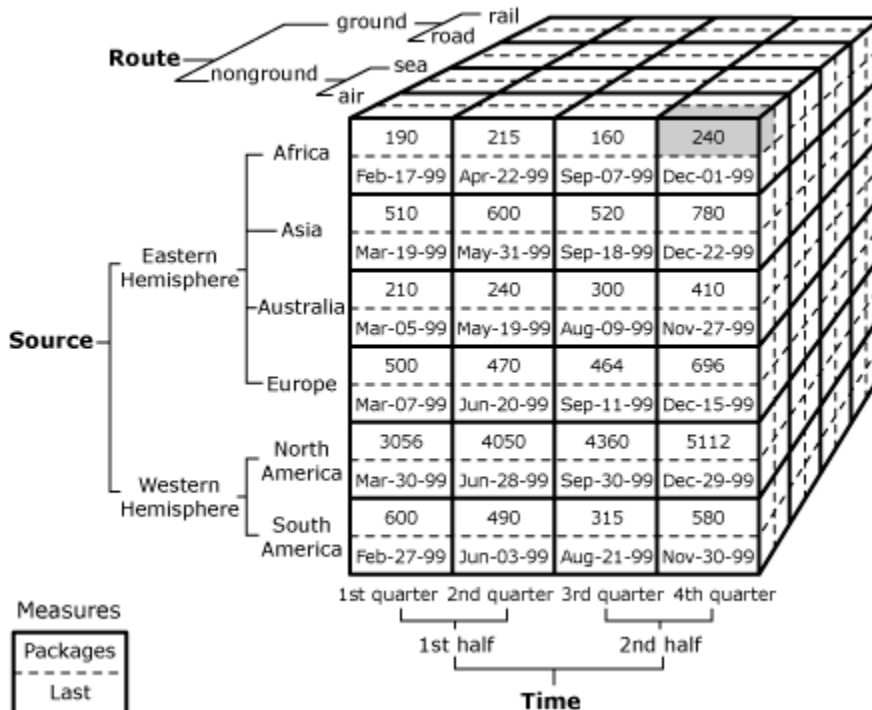
Translations provide server support for client applications that can support multiple languages. Frequently, users from different countries view cube data. It is useful to be able to translate various elements of a cube into a different language so that these users can view and understand the cube's metadata. For example, a business user in France can access a cube from a workstation with a French locale setting, and view the object property values in French. Similarly, a business user in Germany can access the same cube from a workstation with a German locale setting and view the object property values in German.

The collation and language information for the client computer is stored in the form of a locale identifier (LCID). Upon connection, the client passes the LCID to the instance of Analysis Services. The instance uses the LCID to determine which set of translations to use when providing metadata for Analysis Services objects to each business user. If an Analysis Services object does not contain the specified translation, the default language is used to return the content back to the client.

See Also

Cube Cells

A cube is composed of cells, organized by measure groups and dimensions. A cell represents the unique logical intersection in a cube of one member from every dimension in the cube. For example, the cube described by the following diagram contains one measure group that has two measures, organized along three dimensions named Source, Route, and Time.



The single shaded cell in this diagram is the intersection of the following members:

- The air member of the Route dimension.
- The Africa member of the Source dimension.
- The 4th quarter member of the Time dimension.
- The Packages measure.

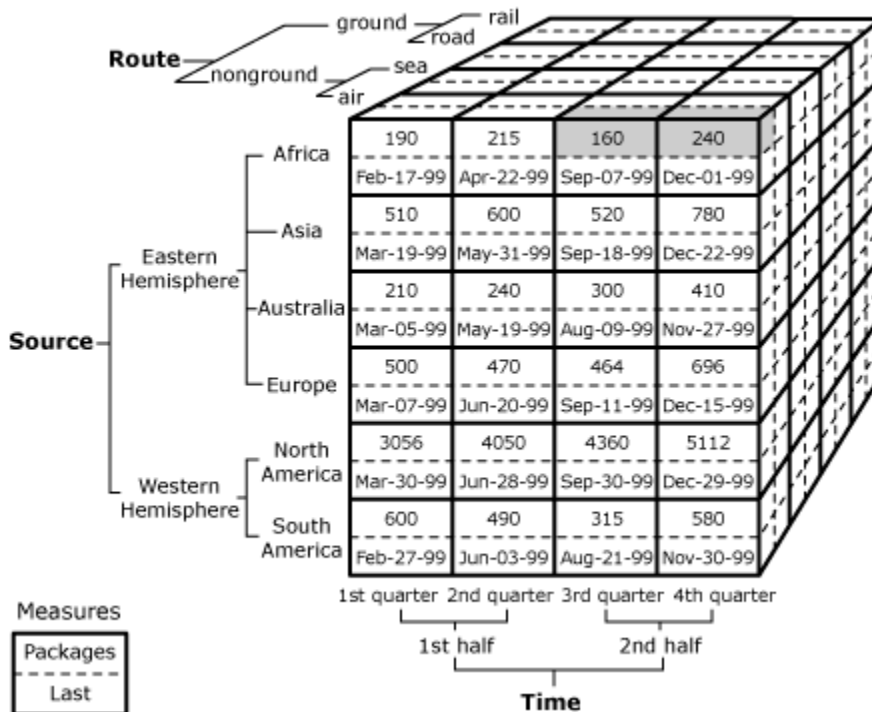
Leaf and Nonleaf Cells

The value for a cell in a cube can be obtained in one of several ways. In the previous example, the value in the cell can be directly retrieved from the fact table of the cube, because all the members used to identify that cell are *leaf members*. A leaf member has no child members, hierarchically speaking, and typically references a single record in a dimension table. This kind of cell is referred to as a *leaf cell*.

However, a cell can also be identified by using *nonleaf members*. A nonleaf member is a member that has one or more child members. In this case, the value of the cell is typically derived from the aggregation of child members associated with the nonleaf member. For example, the intersection of the following members and dimensions refers to a cell whose value is supplied by aggregation:

- The air member of the Route dimension.
- The Africa member of the Source dimension.
- The 2nd half member of the Time dimension.
- The Packages member.

The 2nd half member of the Time dimension is a nonleaf member. Therefore, all of values associated with it must be aggregated values, as shown in the following diagram.



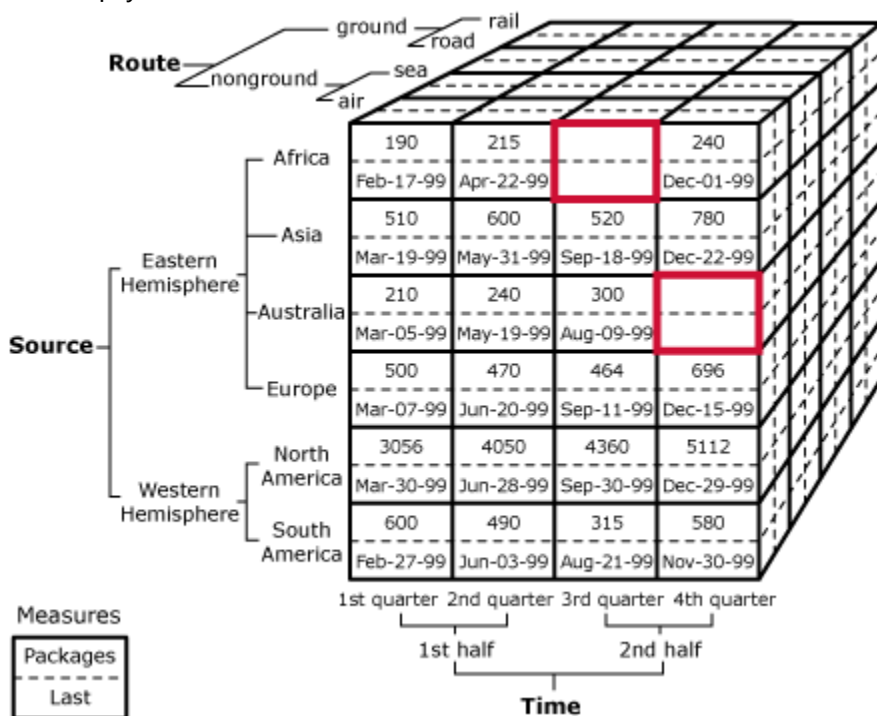
Assuming the aggregations for the 3rd quarter and 4th quarter members are summations, the value of the specified cell is 400, which is the total of all of the leaf cells shaded in the previous diagram. Because the value of the cell is derived from the aggregation of other cells, the specified cell is considered a *nonleaf cell*.

The cell values derived for members that use custom rollups and member groups, in addition to custom members, are handled similarly. However, cell values derived for calculated members are based completely on the Multidimensional Expressions (MDX) expression used to define the calculated member; in some cases, there may be no actual cell data involved. For more information, see [Aggregations and Aggregation Designs](#), [Custom Member Formulas](#), and [Calculations](#).

Empty Cells

It is not required that every cell in a cube contain a value; there can be intersections in a cube that have no data. These intersections, called empty cells, frequently occur in cubes because not every intersection of a dimension attribute with a measure within a cube contains a corresponding record in a fact table. The ratio of empty cells in a cube to the total number of cells in a cube is frequently referred to as the *sparsity* of a cube.

For example, the structure of the cube shown in the following diagram is similar to other examples in this topic. However, in this example, there were no air shipments to Africa for the third quarter or to Australia for the fourth quarter. There is no data in the fact table to support the intersections of those dimensions and measures; therefore the cells at those intersections are empty.



In SQL Server Analysis Services, an empty cell is a cell that has special qualities. Because empty cells can skew the results of crossjoins, counts, and so on, many MDX functions supply the ability to ignore empty cells for the purposes of calculation. For more information, see [Multidimensional Expressions \(MDX\) Reference](#), and [Key Concepts in MDX \(MDX\)](#).

Security

Access to cell data is managed in Analysis Services at the role level, and can be finely controlled by using MDX expressions. For more information, see [Granting Custom Access to Dimension Data](#), and [Granting Custom Access to Cell Data](#).

See Also

[Cube Storage \(SSAS\)](#)

Cube Storage

Storage may include only the cube metadata, or may include all of the source data from the fact table as well as the aggregations defined by dimensions related to the measure group. The amount of data stored depends upon the storage mode selected and the number of aggregations. The amount of data stored directly affects query performance. Microsoft SQL Server Analysis Services uses several techniques for minimizing the space required for storage of cube data and aggregations:

- Storage options enable you to select the storage modes and locations that are most appropriate for cube data.
- A sophisticated algorithm designs efficient summary aggregations to minimize storage without sacrificing speed.
- Storage is not allocated for empty cells.

Storage is defined on a partition-by-partition basis, and at least one partition exists for each measure group in a cube. For more information, see [Aggregations and Aggregation Designs](#), [Partition Storage](#), [Measures and Measure Groups](#), and [Defining and Configuring a Measure Group](#).

Partition Storage

Storage for a measure group can be divided into multiple partitions. Partitions enable you to distribute a measure group into discrete segments on a single server or across multiple servers, and to optimize storage and query performance. Each partition in a measure group can be based on a different data source and stored using different storage settings.

You specify the data source for a partition when you create it. You can also change the data source for any existing partition. A measure group can be partitioned vertically or horizontally. Each partition in a vertically partitioned measure group is based on a filtered view of a single source table. For example, if a measure group is based on a single table that contains several years of data, you could create a separate partition for each year's data. In contrast, each partition in a horizontally partitioned measure group is based on a separate table. You would use horizontal partitions if the data source stores each year's data in a separate table.

Partitions are initially created with the same storage settings as the measure group in which they are created. The storage settings determine whether the detail and aggregation data is stored in multidimensional format on the instance of Analysis Services, in relational format on the source server, or a combination of both. Storage settings also determine whether proactive caching is used to automatically process source data changes to the multidimensional data stored on the Analysis Services.

The partitions of a cube are not visible to the user. However, the choice of storage settings for different partitions may affect the immediacy of data, the amount of disk space that is used, and query performance. Partitions can be stored on multiple instances of Analysis Services. This

provides a clustered approach to cube storage, and distributes workload across Analysis Services servers. For more information, see [Partition Storage](#), [Remote Partitions](#), and [Partitions](#).

Linked Measure Groups

It can require considerable disk space to store multiple copies of a cube on different instances of Analysis Services, but you can greatly reduce the space needed by replacing the copies of the measure group with linked measure groups. A linked measure group is based on a measure group in a cube in another Analysis Services database, on the same or a different instance of Analysis Services. A linked measure group can also be used with linked dimensions from the same source cube. The linked dimensions and measure groups use the aggregations of the source cube and have no data storage requirements of their own. Therefore, by maintaining the source measure groups and dimensions in one database, and creating linked cubes and dimensions in cubes in other databases, you can save disk space that otherwise would be used for storage. For more information, see [Linked Measure Groups](#), and [Linked Dimensions](#).

See Also

[Aggregations and Aggregation Designs](#)

Aggregations and Aggregation Designs

An **T:Microsoft.AnalysisServices.AggregationDesign** object defines a set of aggregation definitions that can be shared across multiple partitions.

An **T:Microsoft.AnalysisServices.Aggregation** object represents the summarization of measure group data at certain granularity of the dimensions.

A simple **T:Microsoft.AnalysisServices.Aggregation** object is composed of: basic information and dimensions. Basic information includes the name of the aggregation, the ID, annotations, and a description. The dimensions are a collection of

T:Microsoft.AnalysisServices.AggregationDimension objects that contain the list of granularity attributes for the dimension.

Aggregations are precalculated summaries of data from leaf cells. Aggregations improve query response time by preparing the answers before the questions are asked. For example, when a data warehouse fact table contains hundreds of thousands of rows, a query requesting the weekly sales totals for a particular product line can take a long time to answer if all the rows in the fact table have to be scanned and summed at query time to compute the answer. However, the response can be almost immediate if the summarization data to answer this query has been precalculated. This precalculation of summary data occurs during processing and is the foundation for the rapid response times of OLAP technology.

Cubes are the way that OLAP technology organizes summary data into multidimensional structures. Dimensions and their hierarchies of attributes reflect the queries that can be asked of the cube. Aggregations are stored in the multidimensional structure in cells at coordinates specified by the dimensions. For example, the question "What were the sales of product X in 1998 for the Northwest region?" involves three dimensions (Product, Time, and Geography) and

one measure (Sales). The value of the cell in the cube at the specified coordinates (product X, 1998, Northwest) is the answer, a single numeric value.

Other questions may return multiple values. For example, "How much were the sales of hardware products by quarter by region for 1998?" Such queries return sets of cells from the coordinates that satisfy the specified conditions. The number of cells returned by the query depends on the number of items in the Hardware level of the Product dimension, the four quarters in 1998, and the number of regions in the Geography dimension. If all summary data has been precalculated into aggregations, the response time of the query will depend only on the time that is required to extract the specified cells. No calculation or reading of data from the fact table is required.

Although precalculation of all possible aggregations in a cube might provide the fastest possible response time for all queries, Analysis Services can easily calculate some aggregated values from other precalculated aggregations. Additionally, calculating all possible aggregations requires significant processing time and storage. Therefore, there is a tradeoff between storage requirements and the percentage of possible aggregations that are precalculated. If no aggregations are precalculated (0%), the amount of required processing time and storage space for a cube is minimized, but query response time may be slow because the data required to answer each query must be retrieved from the leaf cells and then aggregated at query time to answer each query. For example, returning a single number that answers the question asked earlier ("What were the sales of product X in 1998 for the Northwest region") might require reading thousands of rows of data, extracting the value of the column used to provide the Sales measure from each row, and then calculating the sum. Moreover, the length of time required to retrieve that data will vary depending on the storage mode chosen for the data—MOLAP, HOLAP, or ROLAP. **Related topic:** [Partition Storage Modes and Processing](#).

Designing Aggregations

Microsoft SQL Server Analysis Services incorporates a sophisticated algorithm to select aggregations for precalculation so that other aggregations can be quickly computed from the precalculated values. For example, if the aggregations are precalculated for the Month level of a Time hierarchy, the calculation for a Quarter level requires only the summarization of three numbers, which can be quickly computed on demand. This technique saves processing time and reduces storage requirements, with minimal effect on query response time.

The Aggregation Design Wizard provides options for you to specify storage and percentage constraints on the algorithm to achieve a satisfactory tradeoff between query response time and storage requirements. However, the Aggregation Design Wizard's algorithm assumes that all possible queries are equally likely. The Usage-Based Optimization Wizard lets you adjust the aggregation design for a measure group by analyzing the queries that have been submitted by client applications. By using the wizard to tune a cube's aggregation you can increase responsiveness to frequent queries and decrease responsiveness to infrequent queries without significantly affecting the storage needed for the cube.

Aggregations are designed by using the wizards but are not actually calculated until the partition for which the aggregations are designed is processed. After the aggregation has been created, if the structure of a cube ever changes, or if data is added to or changed in a cube's

source tables, it is usually necessary to review the cube's aggregations and process the cube again. **Related topic:** [Designing Partition Storage and Aggregations](#).

See Also

[Partition Storage](#)

Physical Architecture

In This Section

The following topics provide more information about the architecture of an Analysis Services solution.

Topic	Description
Server Architecture	Describes the components of an Analysis Services server.
Local Cubes	Describes how stand-alone cubes are implemented and the scope of such implementation under an Analysis Services solution.
Clients	Describes the client architecture to access data and metadata from an Analysis Services solution.

OLAP Engine Server Components

The server component of Microsoft SQL Server Analysis Services is the **msmdsrv.exe** application, which ordinarily runs as a Windows service. This application consists of security components, an XML for Analysis (XMLA) listener component, a query processor component and numerous other internal components that perform the following functions:

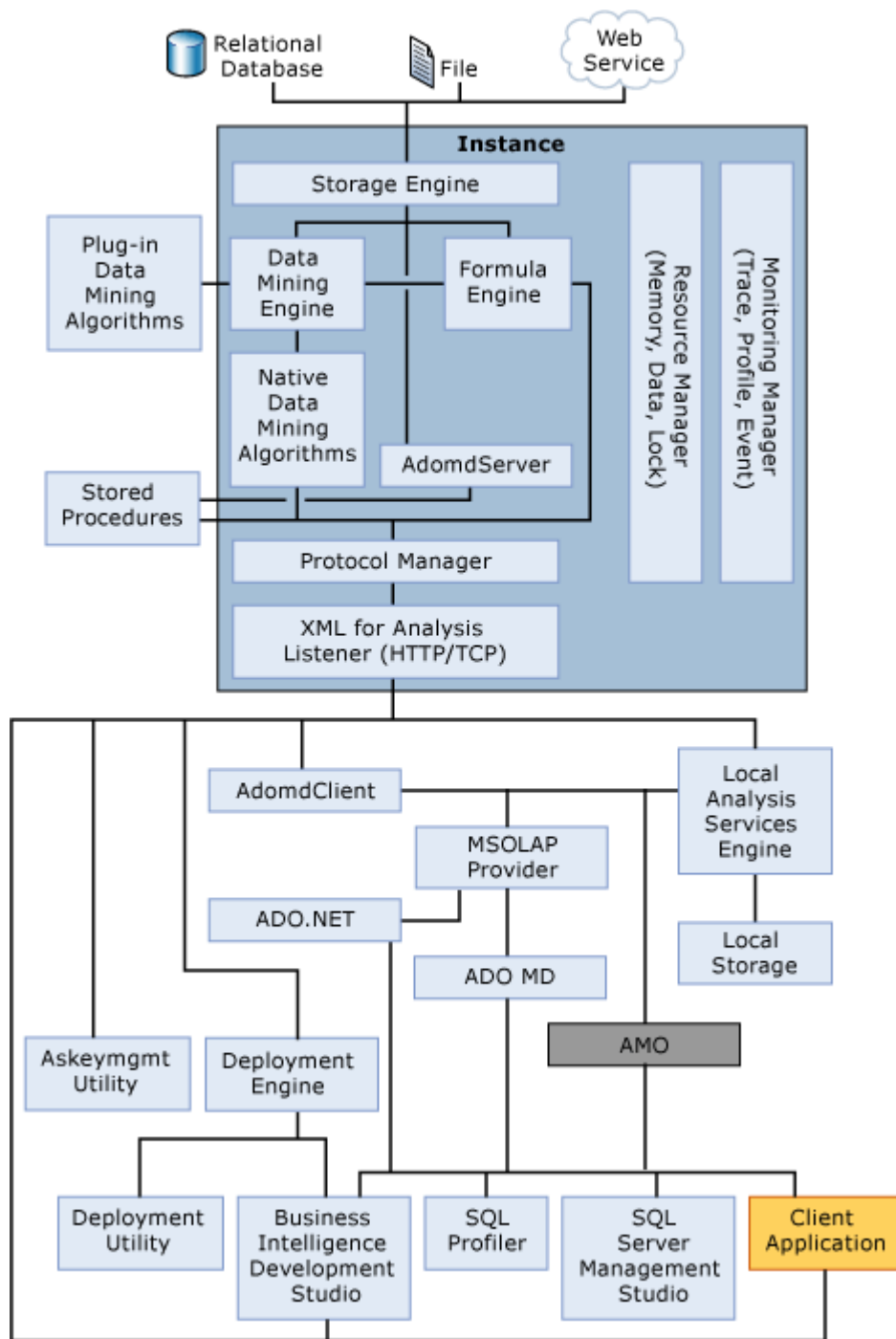
- Parsing statements received from clients
- Managing metadata
- Handling transactions
- Processing calculations
- Storing dimension and cell data
- Creating aggregations
- Scheduling queries
- Caching objects

- Managing server resources

Architectural Diagram

An Analysis Services instance runs as a stand-alone service and communication with the service occurs through XML for Analysis (XMLA), by using either HTTP or TCP. AMO is a layer between the user application and the Analysis Services instance. This layer provides access to Analysis Services administrative objects. AMO is a class library that takes commands from a client application and converts those commands into XMLA messages for the Analysis Services instance. AMO presents Analysis Services instance objects as classes to the end user application, with method members that run commands and property members that hold the data for the Analysis Services objects.

The following illustration shows the Analysis Services components architecture, including all major elements running within the Analysis Services instance and all user components that interact with the instance. The illustration also shows that the only way to access the instance is by using the XML for Analysis (XMLA) Listener, either by using HTTP or TCP.



Note

For more information, see the [SQL Server 2005 Analysis Services Performance Guide](#).

XMLA Listener

The XMLA listener component handles all XMLA communications between Analysis Services and its clients. The Analysis Services **Port** configuration setting in the msmdsrv.ini file can be used to specify a port on which an Analysis Services instance listens. A value of 0 in this file indicates that Analysis Services listen on the default port. Unless otherwise specified, Analysis Services uses the following default TCP ports:

Port	Description
2383	Default instance of SQL Server Analysis Services.
2382	Redirector for other instances of SQL Server Analysis Services.
Dynamically assigned at server startup	Named instance of SQL Server Analysis Services.

In This Section

- [Object Naming \(Analysis Services - Multidimensional Data\)](#)
- [Maximum Capacity Specifications \(Analysis Services - Multidimensional Data\)](#)

See Also

[Specifying and Restricting Ports](#)

[Physical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

Server Process

In This Section

The following topics provide more information about how Analysis Services instances are implemented.

Topic	Description
Analysis Services Configuration Properties	Describes configuration properties and settings for each instance of Analysis Services.

Object Naming

Object Names, IDs, and References

In general, every major object contains the following properties:

- **Name** Normally, the names of sibling objects are unique within the scope of the parent collection. For example, two different **Database** objects may have the same **Dimension** names. However, within each **Database**, the **Dimension** names are unique. Exceptions to this rule are noted later. For example, **Measure** names need to be unique at the **Cube** level, but they do not need to be unique at the level of the parent **MeasureGroup**.
- **ID** The uniqueness restrictions that apply to **Name** properties also apply to **ID** properties. In addition, **ID** properties cannot be changed. **ID** properties are assigned upon creation by the user, by the application, or automatically by the server (if the properties are not specified). If the server assigns **ID** properties, the server sets the **ID** properties to the initial name of the object.
- **Description**

Object References

Objects are referenced by their **ID** properties. The naming convention for **ID** properties starts with a name that is based on the target type, adds the suffix "ID", and possibly adds a prefix that provides extra information on the role that the object plays. For example, the object is the default measure used in a **Cube**. The target type of the object is **Measure**. To this target type name, you add the suffix "ID" and the prefix "Default." The resulting **ID** property name becomes **DefaultMeasureID**, as shown in the following code:

```
<Cube>
  <DefaultMeasureID>Amount</DefaultMeasureID>
  ...
</Cube>
```

When a qualified name that consists of multiple IDs is required (for example, **CubeID** and **MeasureID** both refer to a measure in another **Cube**), then an outer element is introduced to contain the set of **ID** elements.

Reference Exceptions

A **DataSource** reference that consists of a single dot (.) refers to the current database of the current OLAP server. For example, the following XML fragment points to the current **Database** of the current OLAP server instance:

```
<DataSourceID>.</DataSourceID>
```

References to **DataSourceView** (DSV) elements are exceptions to the referencing rules. A **DataSet** object in the Microsoft .NET Framework describes the schema for a DSV. In the schema, a DSV object has a **Name**, but not an immutable **ID**. Therefore, references to a DSV object needs to use the **Name** as the identification mechanism. For example, the reference, **ColumnBinding.TableID**, does not contain the **ID**. Instead, the reference contains the **Name** of the appropriate table.

Naming Guidelines

ASSL applies the same rules for case and whitespace to **Names** and **ID** properties as use in DSO 8.0:

- The uniqueness check for **Name** and **ID** is not case sensitive. Therefore, it is not possible to have a **Cube** named "sales" and another named "Sales" in the same database.
- While a **Name** or **ID** property can contain embedded spaces, the property cannot contain leading or trailing spaces. Leading and trailing spaces are implicitly trimmed. This rule applies both to the **Name** and **ID** of an object, as well as to the values of elements that reference that **Name** and **ID**.

The following rules also apply to **Name** and **ID** properties. These rules are similar to rules in DSO 8.0.

- The maximum number of characters is 100.
- There is no special requirement for the first character of an identifier. The first character may be any valid character

The following reserved names must not be used:

- AUX
- CLOCK\$
- COM1 through COM9 (COM1, COM2, COM3, and so on)
- CON
- LPT1 through LPT9 (LPT1, LPT2, LPT3, and so on)
- NUL
- PRN
- NULL is not allowed as a character in any string within the XML

The following table lists invalid characters for specific objects.

Object	Invalid characters
Server	The name must follow the rules for computer names. (IP addresses are not valid.)
DataSource	: / \ * ? " () [] {} < >
Level or Attribute	. , ; ' ` : / \ * ? " & % \$! + = () [] {} < >
Dimension or Hierarchy	.,;'\`:/* ?' "&%\$!+=() [] {} <,>
All other objects	. , ; ' ` : / \ * ? " & % \$! + = () [] {} < >

Localized Names

Captions for objects that are visible to clients (for example, **Dimension**, Hierarchy, and **Level**) can be localized into different languages. Captions for objects that are defined by way of commands (for example, calculated measures and named sets) are provided as part of the MDX definition of the objects.

The bindings for attributes also allow a different source (for example, a different source column) for the attribute name to be provided for different languages.

It is not possible to localize the Names of objects.

Maximum Capacity Specifications

The following tables specify the maximum sizes and numbers of various objects defined in Analysis Services components under different server deployment modes.

This topic contains the following sections:

[Multidimensional and Data Mining \(DeploymentMode=0\)](#)

[SharePoint \(DeploymentMode=1\)](#)

[Tabular \(DeploymentMode=2\)](#)

Multidimensional and Data Mining (DeploymentMode=0)

MOLAP storage mode, which stores both data and metadata, has additional physical limits on file sizes. String store files are a maximum size of 4 GB by default. If you require larger files for string stores, you can specify a different string storage architecture. For more information, see [Configure Scalable String Storage for Dimensions and Measures](#).

Object	Maximum sizes/numbers
Databases in an instance	$2^{31}-1 = 2,147,483,647$
Dimensions in a database	$2^{31}-1 = 2,147,483,647$
Attributes in a dimension	$2^{31}-1 = 2,147,483,647$
Members in a dimension attribute	$2^{31}-1 = 2,147,483,647$
User-defined hierarchies in a dimension	$2^{31}-1 = 2,147,483,647$
Levels in a user-defined hierarchy	$2^{31}-1 = 2,147,483,647$
Cubes in a database	$2^{31}-1 = 2,147,483,647$
Measure groups in a cube	$2^{31}-1 = 2,147,483,647$
Measures in a measure group	$2^{31}-1 = 2,147,483,647$
Calculations in a cube	$2^{31}-1 = 2,147,483,647$
KPIs in a cube	$2^{31}-1 = 2,147,483,647$

Object	Maximum sizes/numbers
Actions in a cube	$2^{31}-1 = 2,147,483,647$
Partitions in a cube	$2^{31}-1 = 2,147,483,647$
Translations in a cube	$2^{31}-1 = 2,147,483,647$
Aggregations in a partition	$2^{31}-1 = 2,147,483,647$
Cells returned by a query	$2^{31}-1 = 2,147,483,647$
Record size of the source query	64K
Length of object names	100 characters
Maximum number of distinct states in a data mining model attribute column	$2^{31}-1 = 2,147,483,647$
Maximum number of attributes considered (feature selection)	$2^{31}-1 = 2,147,483,647$

For more information about object naming guidelines, see [Objects and Object Characteristics](#).

For more information about data source limitations for online analytical processing (OLAP) and data mining, see [Working with Data Sources \(SSAS\)](#), [Working with Data Sources \(SSAS\)](#), and [Objects and Object Characteristics](#).

SharePoint (DeploymentMode=1)

Object	Maximum sizes/numbers
Databases in an instance	$2^{31}-1 = 2,147,483,647$
Tables in a database	$2^{31}-1 = 2,147,483,647$
Columns in a table	$2^{31}-1 = 2,147,483,647$  Warning - Total number of columns in a table depends on the total number of Measures and Calculated Columns associated to the same table. - The maximum number of 'Columns + Measures + Calculated Columns' for a table is $2^{31}-1 = 2,147,483,647$
Rows in a table	Unlimited

Object	Maximum sizes/numbers
	 Warning With the restriction that no single column may contain more than 1,999,999,997 distinct values.
Hierarchies in a table	$2^{31-1} = 2,147,483,647$
Levels in a hierarchy	$2^{31-1} = 2,147,483,647$
Relationships	$2^{31-1} = 2,147,483,647$
Key Columns in a table	$2^{31-1} = 2,147,483,647$
Measures in a table	$2^{31-1} = 2,147,483,647$  Warning - Total number of Measures in a table depends on the total number of Columns and Calculated Columns associated to the same table. - The maximum number of 'Columns + Measures + Calculated Columns' for a table is $2^{31-1} = 2,147,483,647$
Calculated Columns in a table	$2^{31-1} = 2,147,483,647$  Warning - Total number of Calculated Columns in a table depends on the total number of Columns and Measures associated to the same table. - The maximum number of 'Columns + Measures + Calculated Columns' for a table is $2^{31-1} = 2,147,483,647$
Cells returned by a query	$2^{31-1} = 2,147,483,647$
Record size of the source query	64K
Length of object names	100 characters

Tabular (DeploymentMode=2)

Object	Maximum sizes/numbers
Databases in an instance	$2^{31}-1 = 2,147,483,647$
Tables in a database	$2^{31}-1 = 2,147,483,647$
Columns in a table	$2^{31}-1 = 2,147,483,647$  Warning - Total number of columns in a table depends on the total number of Measures and Calculated Columns associated to the same table. - The maximum number of 'Columns + Measures + Calculated Columns' for a table is $2^{31}-1 = 2,147,483,647$
Rows in a table	Unlimited  Warning With the restriction that no single column in the table can have more than 1,999,999,997 distinct values.
Hierarchies in a table	$2^{31}-1 = 2,147,483,647$
Levels in a hierarchy	$2^{31}-1 = 2,147,483,647$
Relationships	$2^{31}-1 = 2,147,483,647$
Key Columns in a table	$2^{31}-1 = 2,147,483,647$
Measures in a table	$2^{31}-1 = 2,147,483,647$  Warning - Total number of Measures in a table depends on the total number of Columns and Calculated Columns associated to the same table. - The maximum number of 'Columns + Measures + Calculated Columns' for a table is $2^{31}-1 = 2,147,483,647$
Calculated Columns in a table	$2^{31}-1 = 2,147,483,647$

Object	Maximum sizes/numbers
	 Warning - Total number of Calculated Columns in a table depends on the total number of Columns and Measures associated to the same table. - The maximum number of 'Columns + Measures + Calculated Columns' for a table is $2^{31}-1 = 2,147,483,647$
Cells returned by a query	$2^{31}-1 = 2,147,483,647$
Record size of the source query	64K
Length of object names	100 characters

See Also

[Determine Server Mode \(Analysis Services\)](#)

[General Properties](#)

Data Types in Analysis Services

For all **T:Microsoft.AnalysisServices.DataItem** objects, Analysis Services supports the following subset of **System.Data.OleDb.OleDbType**. To set or read the data type, use [DataItem DataType \(ASSL\)](#).

Supported Data Types

BigInt	A 64-bit signed integer. The BigInt value type represents integers with values ranging from negative 9,223,372,036,854,775,808 to positive 9,223,372,036,854,775,807.
Binary	A stream of binary data of Byte type. Byte is a value type that represents unsigned integers with values that range from 0 to 255.
Boolean	Instances of this type have values of either true or false .

Currency	A currency value ranging from - 922,337,203,685,477.5808 to +922,337,203,685,477.5807 with accuracy to a ten-thousandth of a currency unit (four decimal places).
Date	Date and time data, stored as a double. The whole portion is the number of days since December 30, 1899, and the fractional portion is a fraction of a day or time of the day.
Double	A floating-point number within the range of -1.79769313486232E +308 to 1.79769313486232E +308. A Double value stores number information up to 15 decimal digits of precision.
Integer	A 32-bit signed integer that represents signed integers with values that range from negative 2,147,483,648 through positive 2,147,483,647.
Single	A floating-point number within the range of - 3.4028235E +38 through 3.4028235E +38. A Single value stores number information up to 7 decimal digits of precision.
Smallint	A 16-bit signed integer. The Smallint value type represents signed integers with values ranging from negative 32768 to positive 32767.
Tinyint	An 8-bit signed integer. The Tinyint value type represents integers with values ranging from negative 128 to positive 127.
UnsignedBigInt	A 64-bit unsigned integer. The UnsignedBigInt value type represents unsigned integers with values ranging from 0 to 18,446,744,073,709,551,615.
UnsignedInt	A 32-bit unsigned integer. The UnsignedInt value type represents unsigned integers with values ranging from 0 to

	4,294,967,295.
UnsignedSmallInt	A 16-bit unsigned integer. The UnsignedSmallInt value type represents unsigned integers with values ranging from 0 to 65535.
UnsignedTinyInt	An 8-bit unsigned integer. The UnsignedTinyInt value type represents unsigned integers with values that range from 0 to 255
WChar	A null-terminated stream of Unicode characters. A WChar is a sequential collection of Unicode characters that is used to represent text.

AMO Validations on Data Types

The following table lists the extra validations that Analysis Management Objects (AMO) does for certain bindings:

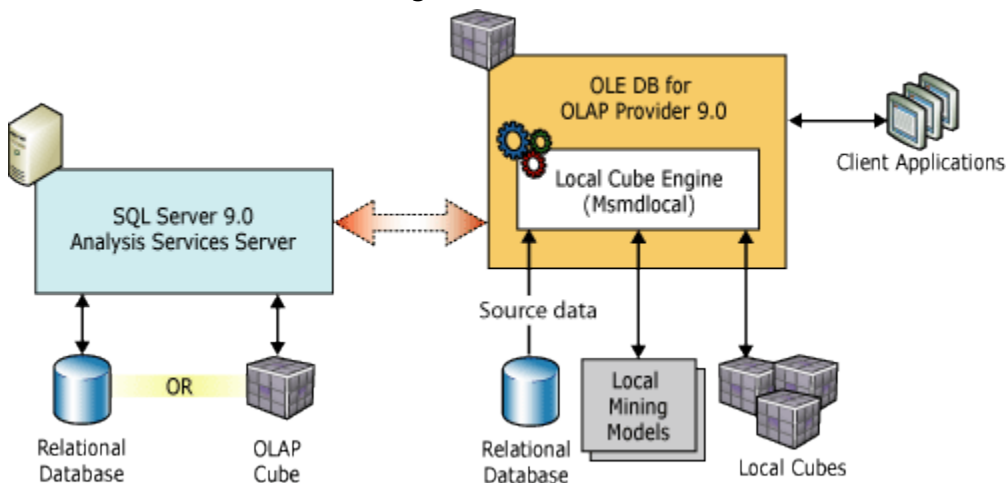
Object	Binding	Allowed Data Types
DimensionAttribute	KeyColumns	All but Binary
	NameColumn	Only WChar
	SkippedLevelsColumn	Only integer types: BigInt, Integer, SmallInt, TinyInt, UnsignedBigInt, UnsignedInt, UnsignedSmallInt, UnsignedTinyInt
	CustomRollupColumn	Only WChar
	CustomRollupPropertiesColumn	Only WChar
	UnaryOperatorColumn	Only WChar
	ValueColumn	All
AttributeTranslation	CaptionColumn	Only WChar
ScalarMiningStructureColumn	KeyColumns	All but Binary
	NameColumn	Only WChar

Object	Binding	Allowed Data Types
TableMiningStructureColumn	ForeignKeyColumns	All but Binary
MeasureGroupAttribute	KeyColumns	All but Binary
Distinct Count Measure	Source	BigInt, Currency, Double, Integer, Single, SmallInt, TinyInt, UnsignedBigInt, UnsignedInt, UnsignedSmallInt, UnsignedTinyInt

Local Cubes

To create, update or delete local cubes, you must write and execute either an ASSL script or an AMO program.

Local cubes and local mining models allow analysis on a client workstation while it is disconnected from the network. For example, a client application might call the OLE DB for OLAP 9.0 Provider (MSOLAP.3), which loads the local cube engine to create and query local cubes, as shown in the following illustration:



ADMOD.NET and Analysis Management Objects (AMO) also load the local cube engine when interacting with local cubes. Only a single process can access a local cube file, because the local cube engine exclusively locks a local cube file when it establishes a connection to the local cube. With a process, up to five simultaneous connections are permitted.

A .cub file may contain more than one cube or data mining model. Queries to the local cubes and data mining models are handled by the local cube engine and do not require a connection to an Analysis Services instance.



Note

The use of SQL Server Management Studio and SQL Server Data Tools (SSDT) to manage local cubes is not supported.

Local Cubes

A local cube can be created and populated from either an existing cube in an Analysis Services instance or from a relational data source.

Source for data for local cube	Creation method
Server-based cube	You can use either the CREATE GLOBAL CUBE statement or an Analysis Services Scripting Language (ASSL) script to create and populate a cube from a server-based cube. For more information, see CREATE GLOBAL CUBE Statement (MDX) or Analysis Services Scripting Language Reference .
Relational data source	You use an ASSL script to create and populate a cube from an OLE DB relational database. To create a local cube using ASSL, you simply connect to a local cube file (*.cub) and execute the ASSL script in the same manner as executing an ASSL script against an Analysis Services instance to create a server cube. For more information, see Analysis Services Scripting Language Reference .

Use the REFRESH CUBE statement to rebuild a local cube and update its data. For more information, see [REFRESH CUBE Statement \(MDX\)](#).

Local Cubes Created from Server-based Cubes

When creating local cubes created from server-based cubes, the following considerations apply:

- Distinct count measures are not supported.
- When you add a measure, you must also include at least one dimension that is related to the measure being added. For more information about dimension relationships to measure groups, see [Dimension Relationships](#).

- When you add a parent-child hierarchy, levels and filters on a parent-child hierarchy are ignored and the entire parent-child hierarchy is included.
- Member properties are not created.
- When you include a semi-additive measure, no slices are permitted on either the Account or the Time dimension.
- Reference dimensions are always materialized.
- When you include a many-to-many dimension, the following rules apply:
 - You cannot slice the many-to-many dimension.
 - You must add a measure from the intermediary measure group.
 - You cannot slice any of the dimensions common to the two measure groups involved in the many-to-many relationship.
- Only those calculated members, named sets, and assignments that rely upon measures and dimensions added to the local cube will appear in the local cube. Invalid calculated members, named sets, and assignments will be automatically excluded.

Security

In order for a user to create a local cube from a server cube, the user must be granted **Drillthrough and Local Cube** permissions on the server cube. For more information, see [Granting Cube Access](#).

Local cubes are not secured using roles like server cubes. Anyone with file-level access to a local cube file can query cubes in it. You can use the **Encryption Password** connection property on a local cube file to set a password on the local cube file. Setting a password on a local cube file requires all future connections to the local cube file to use this password in order to query the file.

See Also

[CREATE GLOBAL CUBE Statement \(MDX\)](#)

[ASSL](#)

[REFRESH CUBE Statement \(MDX\)](#)

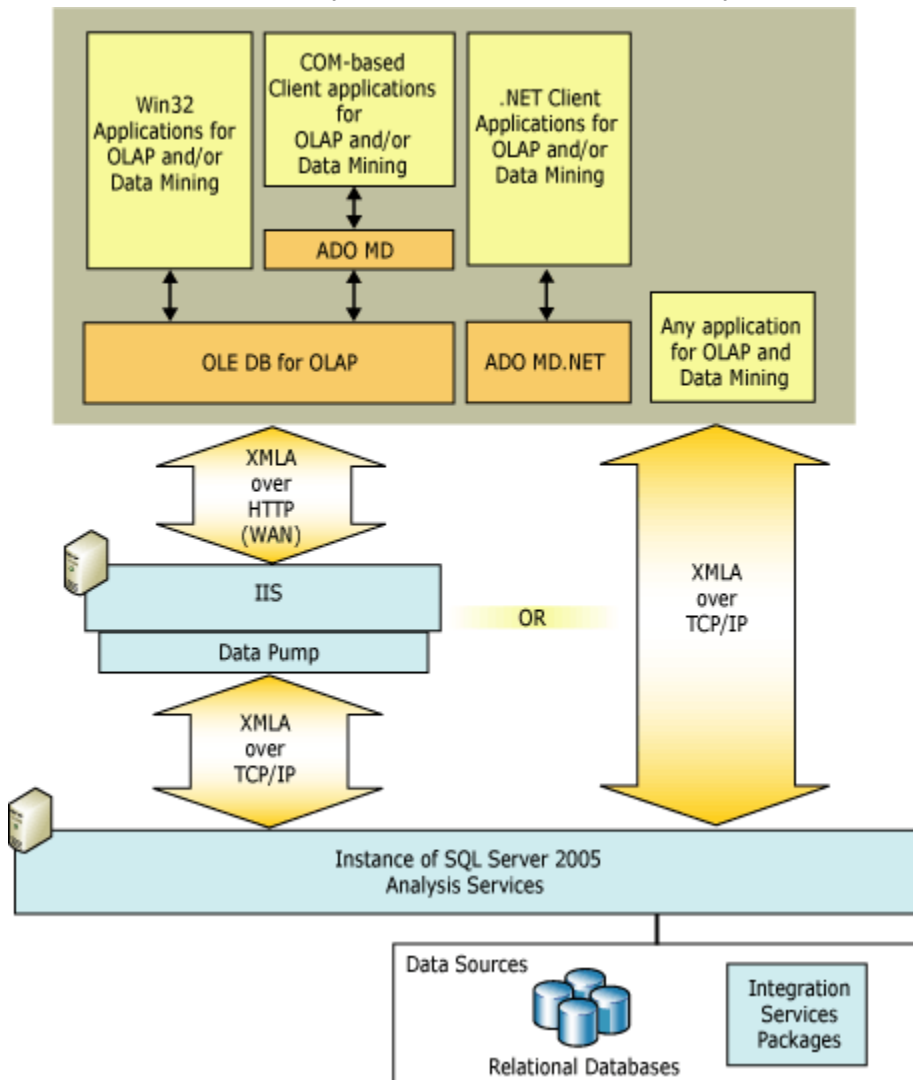
Clients

Microsoft SQL Server Analysis Services supports a thin-client architecture. The Analysis Services calculation engine is entirely server-based, so all queries are resolved on the server. As a result, only a single round trip between the client and the server is required for each query, resulting in scalable performance as queries increase in complexity.

The native protocol for Analysis Services is XML for Analysis (XML/A). Analysis Services provides several data access interfaces for client applications, but all of these components communicate with an instance of Analysis Services using XML for Analysis.

Several different providers are provided with Analysis Services to support different programming languages. A provider communicates with an Analysis Services server by sending and receiving

XML for Analysis in SOAP packets over TCP/IP or over HTTP through Internet Information Services (IIS). An HTTP connection uses a COM object instantiated by IIS, called a data pump, which acts as a conduit for Analysis Services data. The data pump does not examine the underlying data contained in the HTTP stream in any way, nor are any of the underlying data structures available to any of the code in the data library itself.

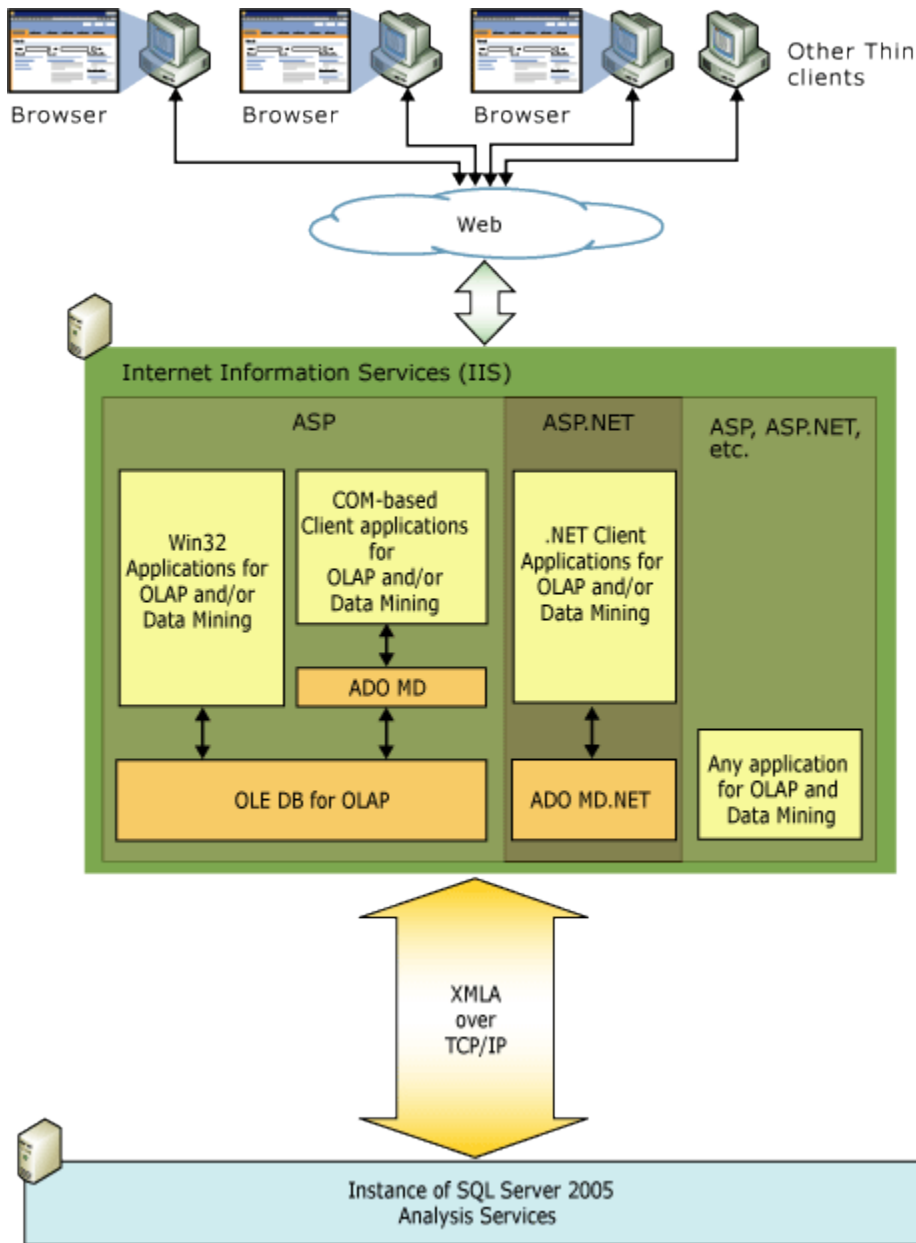


Win32 client applications can connect to an Analysis Services server using OLE DB for OLAP interfaces or the Microsoft® ActiveX® Data Objects (ADO) object model for Component Object Model (COM) automation languages, such as Microsoft Visual Basic®. Applications coded with .NET languages can connect to an Analysis Services server using ADOMD.NET.

Existing applications can communicate with Analysis Services without modification simply by using one of the Analysis Services providers.

Programming Language	Data Access Interface
C++	OLE DB for OLAP
Visual Basic 6	ADO MD
.NET languages	ADO MD.NET
Any language that supports SOAP	XML for Analysis

Analysis Services has a Web architecture with a fully scalable middle tier for deployment by both small and large organizations. Analysis Services provides broad middle tier support for Web services. ASP applications are supported by OLE DB for OLAP and ADO MD, ASP.NET applications are supported by ADOMD.NET. The middle tier, illustrated in the following figure, is scalable to many concurrent users.



Both client and middle tier applications can communicate directly with Analysis Services without using a provider. Client and middle tier applications may send XML for Analysis in SOAP packets over TCP/IP, HTTP, or HTTPS. The client may be coded using any language that supports SOAP. Communication in this case is most easily managed by Internet Information Services (IIS) using HTTP, although a direct connection to the server using TCP/IP may also be coded. This is the thinnest possible client solution for Analysis Services.

Analysis Services in Tabular or SharePoint Mode

In SQL Server 2012, the server can be started in xVelocity in-memory analytics engine (VertiPaq) mode for tabular databases and for PowerPivot workbooks that have been published to a SharePoint site.

PowerPivot for Excel and SQL Server Data Tools (SSDT) are the only client environments that are supported for creating and querying in-memory databases that use SharePoint or Tabular mode, respectively. The embedded PowerPivot database that you create by using the Excel and PowerPivot tools is contained within the Excel workbook, and is saved as part of the Excel .xlsx file.

However, a PowerPivot workbook can use data that is stored in a traditional cube if you import the cube data into the workbook. You can also import data from another PowerPivot workbook if it has been published to a SharePoint site.



Note

When you use a cube as a data source for a PowerPivot workbook, the data that you get from the cube is defined as an MDX query; however, the data is imported as a flattened snapshot. You cannot interactively work with the data or refresh the data from the cube.

For more information about using an SSAS cube as a data source, see the [PowerPivot for Excel](#).

Interfaces for PowerPivot Client

PowerPivot interacts with the xVelocity in-memory analytics engine (VertiPaq) storage engine within the workbook by using the established interfaces and languages for Analysis Services: AMO and ADOMD.NET, and MDX and XMLA. Within the add-in, measures are defined by using a formula language similar to Excel, Data Analysis Expressions (DAX). DAX expressions are embedded within the XMLA messages that are sent to the in-process server. For more information, see [MDX and DAX](#).

Providers

Communications between PowerPivot and Excel use the MSOLAP OLEDB provider (version 11.0). Within the MSOLAP provider, there are four different modules, or transports, that can be used for sending messages between the client and server.

TCP/IP Used for normal client-server connections.

HTTP Used for HTTP connections via the SSAS data pump service, or by a call to the SharePoint PowerPivot Web Service (WS) component.

INPROC Used for connections to the in-process engine.

CHANNEL Reserved for communications with the PowerPivot System Service in the SharePoint farm. For more information about the components needed to work with PowerPivot in a SharePoint installation, see [Planning and Architecture \(PowerPivot for SharePoint\)](#).

See Also

[Server Cubes \(Analysis Services - Multidimensional Data\)](#)

International Considerations

Microsoft SQL Server Analysis Services supports the storage and manipulation of multilingual data and metadata. Features include national language and collation support for sorts and comparisons, translations for multilingual metadata support, and currency conversion functionality. Using the Analysis Services multilingual and multicultural features, you can do the following:

Action	Description
Define the language and collation support for Analysis Services instances and databases.	You can define the default language and Windows collation for an Analysis Services instance, as well as defining translations for the properties of Analysis Services databases. To define the language and collation support for Analysis Services instances and databases: • You can use SQL Server Management Studio to define the default language and collation for an Analysis Services instance. For more information, see Business Intelligence Wizard F1 Help. • You can use the SQL Server Data Tools (SSDT) to define the translations for the caption, description, and account types for an Analysis Services database. For more information, see Working with Translations (SSAS).
Define language and collation support for objects in an Analysis Services database.	Translations for data and metadata can be defined on an Analysis Services instance so that client applications can receive Analysis Services objects in a specified language and collation. To define and retrieve translated data and metadata: • You can define translations on Analysis Services objects to support multiple languages. For more information, see Working with Translations (SSAS).

Action	Description
	You can retrieve data and metadata from Analysis Services objects on which translations have been defined automatically by providing a locale identifier when connecting to an Analysis Services instance. For more information, see Working with Client Applications (SSAS).
Provide support for multiple currencies in an Analysis Services database.	Analysis Services provides currency conversion support by using specialized Multidimensional Expressions (MDX) scripts to convert measures containing currency data. To provide support for multiple currencies in an Analysis Services database: You can use the Business Intelligence Wizard to generate an MDX script that uses a combination of data and metadata from dimensions, attributes, and measure groups to convert measures containing currency data. For more information, see Working with Currency Conversions (SSAS).

In This Section

Topic	Description
Working with Languages and Collations (SSAS)	Describes how to specify default language and Windows collation for an Analysis Services instance, as well as how your choices affect data and metadata managed by Analysis Services.
Working with Translations (SSAS)	Describes how to define translations for an Analysis Services database and objects contained by the database, as well as how Analysis Services resolves requests for translated data and metadata from client

Topic	Description
	applications.
Working with Currency Conversions (SSAS)	Describes how to define a currency conversion using the Business Intelligence Wizard.
Working with Client Applications (SSAS)	Describes the common issues encountered by client applications when working with multilingual and multicultural data and metadata in Analysis Services.

See Also

[Translations \(SSAS\)](#)

[Database Properties Dialog Box \(SSAS\)](#)

[Database Designer \(SSAS\)](#)

[Business Intelligence Wizard F1 Help \(SSAS\)](#)

Languages and Collations

Microsoft SQL Server Analysis Services supports all languages that are supported by Microsoft Windows operating systems. For more information about language support in SQL Server Analysis Services, see [International Considerations \(Analysis Services - Multidimensional Data\)](#). Besides specifying the default language and collation used by an Analysis Services instance, you can also provide multilanguage support for individual Analysis Services objects, including cubes, measure groups, dimensions, hierarchies, and attributes, by defining a translation associated with an Analysis Services object. The default language and collation settings for an Analysis Services instance specify the settings used for data and metadata if a translation for a specific language identifier is not provided for an Analysis Services object, or if a client application does not specify a language identifier when connecting to an Analysis Services instance.

Language Identifiers

Analysis Services uses Windows language identifiers to specify the selected language for Analysis Services instances and objects. A Windows language identifier corresponds to a combination of Windows primary language and sublanguage identifiers. For example, if you select **English (United States)** in the **Language** drop-down list on the **Collation Settings** page of the **Microsoft SQL Server Installation Wizard**, the corresponding Windows language identifier, 0x0409 (or 1033), is specified in the Language element of the configuration settings file for the Analysis Services instance. For more information about available Windows language identifiers, see "Table of Language Identifiers" in the MSDN documentation.

Collations

Analysis Services uses Windows collations to specify the selected collation for Analysis Services instances and objects. A Windows collation identifier corresponds to a combination of code page and sort order information. For example, if you select **Latin1_General** in the **Windows collations** drop-down list on the **Collation Settings** page of the **Microsoft SQL Server Installation Wizard**, and select the **Binary** sort order option, the corresponding Windows collation identifier, `Latin1_General_BIN`, is specified in the Collation element of the configuration settings file for the Analysis Services instance.

Windows Collations

Windows collations define rules for storing character data based on an associated Windows locale. The base Windows collation rules specify which alphabet or language is used when dictionary sorting is applied, and also the code page that is used to store non-Unicode character data. Binary collations sort data based on the sequence of coded values that are defined by the locale and data type. A binary collation in Analysis Services defines the language locale and the ANSI code page to be used, enforcing a binary sort order. Because of their relative simplicity, binary collations are useful in achieving improved application performance. For non-Unicode data types, data comparisons are based on the code points defined in the ANSI code page. For Unicode data types, data comparisons are based on the Unicode code points. For binary collations on Unicode data types, the locale is not considered in data sorts. For example, `Latin1_General_BIN` and `Japanese_BIN` yield identical sorting results when used on Unicode data. For more information about Windows collations in SQL Server, see [Working with Collations](#).

By specifying a Windows collation for Analysis Services, the Analysis Services instance uses the same code pages and sorting and comparison rules as an application that is running on a computer for which you have specified the associated Windows locale. For example, the French Windows collation for Analysis Services matches the collation attributes of the French locale for Windows.

There are more Windows locales than there are Windows collations defined for Analysis Services. The names of Windows locales are based on a language identifier, such as English, and a sublanguage identifier, such as United States or Australia. However, many languages share common alphabets and rules for sorting and comparing characters. For example, 33 Windows locales, including all the Portuguese and English Windows locales, use the Latin1 code page (1252) and follow a common set of rules for sorting and comparing characters. The SQL Server Windows collation `Latin1_General`, based on this code page and associated sorting rules, supports all 33 of these Windows locales. Also, Windows locales specify attributes that are not covered by Analysis Services Windows collations, such as currency, date, and time formats. Because countries and regions such as Australia and the United States have different currency, date, and time formats, they require different Windows collations. They do not require different Analysis Services Windows collations, however, because they have the same alphabet and rules for sorting and comparing characters.



Note

While multiple language identifiers can be specified for Analysis Services objects, the same Analysis Services Windows collation is used for all Analysis Services objects, with a

single exception, regardless of language identifier. The single exception to this functionality is the **CaptionColumn** property of an attribute in a database dimension, for which you can specify an Analysis Services Windows collation to collate the members of the specified attribute. For more information about defining attribute translations, see [Working with Translations \(SSAS\)](#). If the same language is used by all of the users for your Analysis Services instance, select the collation that supports the specified default language for your instance. If multiple languages are used, choose a collation that best supports the requirements of the various languages. For example, if the users of your instance generally speak western European languages, select the Latin1_General collation.

Sort Order Options

Several sort order options can be applied to the specified Analysis Services Windows collation to additionally define sorting and comparison rules based on case, accent, kana, and width sensitivity. The following table describes Windows collation sort order options and associated suffixes for Analysis Services.

Sort order (suffix)	Sort order description
Binary (_BIN) ¹	Sorts and compares data in Analysis Services based on the bit patterns defined for each character. Binary sort order is case sensitive and accent sensitive. Binary is also the fastest sorting order. For more information, see Using Binary Collations. If this option is not selected, Analysis Services follows sorting and comparison rules as defined in dictionaries for the associated language or alphabet. This option corresponds to the Binary option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
BIN2 (_BIN2) ¹	Sorts and compares data in Analysis Services based on Unicode code points for Unicode data. For non-Unicode data, BIN2 will use comparisons identical to binary sorts. The advantage of using a BIN2 sort order is that no data resorting is required in

Sort order (suffix)	Sort order description
	applications that compare sorted data. As a result, BIN2 provides simpler application development and possible performance increases. For more information, see Using Binary Collations. This option corresponds to the Binary 2 option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
Case-sensitive (_CS)	Distinguishes between uppercase and lowercase letters. If selected, lowercase letters sort ahead of their uppercase versions. This option is set by selecting the Case-sensitive option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
Case-insensitive (_CI)	Does not distinguish between uppercase and lowercase letters. Analysis Services considers the uppercase and lowercase letters to be identical for sorting purposes. This option is set by clearing the Case-sensitive option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
Accent-sensitive (_AS)	Distinguishes between accented and unaccented characters. For example, 'a' is not equal to 'á'. If this option is not selected, Analysis Services considers the accented and unaccented versions of letters to be

Sort order (suffix)	Sort order description
	identical for sorting purposes. This option corresponds to the Accent-sensitive option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
Accent-insensitive (_AI)	Does not distinguish between accented and unaccented characters. Analysis Services considers the accented and unaccented versions of letters to be identical for sorting purposes. This option is set by clearing the Accent-sensitive option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
Kana-sensitive (_KS)	Distinguishes between the two types of Japanese kana characters: hiragana and katakana. If this option is not selected, Analysis Services considers hiragana and katakana characters to be equal for sorting purposes.  Note There is no sort order suffix for kana-insensitive sorting. This option corresponds to the Kana-sensitive option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.
Width-sensitive (_WS)	Distinguishes between a single-byte character and the same character when

Sort order (suffix)	Sort order description
	represented as a double-byte character. If this option is not selected, Analysis Services considers the single-byte and double-byte representation of the same character to be identical for sorting purposes.  Note There is no sort order suffix for width-insensitive sorting. This option corresponds to the Width-sensitive option on the Collation Settings page of the Microsoft SQL Server Installation Wizard or the Language/Collation page of the Analysis Server Properties dialog box in SQL Server Management Studio.

¹ If BIN2 is selected, the case-sensitive, case-insensitive, accent-sensitive, accent-insensitive, kana-sensitive, and width-sensitive options are not available.

Each Windows collation is combined with sort order suffixes to define case, accent, width, or kana sensitivity. For example, the default value of the **Collation** configuration property for Analysis Services is Latin1_General_AS_CS, specifying that the Latin1_General collation is used, with an accent-sensitive, case-sensitive sort order.

Specifying the Default Language and Collation

You can specify the default language and collation settings for an Analysis Services instance during installation, in the **Collation Settings** page of the **Microsoft SQL Server Installation Wizard**.

After installation, you can change the default language and collation settings for an Analysis Services instance in SQL Server Management Studio using the **Language/Collation** page of the **Analysis Server Properties** dialog box. For more information about how to use the **Analysis Server Properties** dialog box to change language and collation settings, see [Language/Collation \(Analysis Server Properties Dialog Box\) \(SSAS\)](#).

Using EnableFast1033Locale

If you use the English (United States) language identifier (0x0409, or 1033) as the default language for the Analysis Services instance, you can get additional performance benefits by setting the **EnableFast1033Locale** configuration property, an advanced configuration property available only for that language identifier. Setting the value of this property to **true** enables Analysis Services to use a faster algorithm for string hashing and comparison. For more

information about setting configuration properties, see [Analysis Services Configuration Properties](#).

See Also

[International Considerations for Analysis Services \(SSAS\)](#)

Translations

Multilanguage support in Microsoft SQL Server Analysis Services is accomplished by using translations. A translation contains a language identifier and bindings for properties of Analysis Services objects which can be presented in multiple languages. For example, you can define a translation for an Analysis Services database to present the caption and description of that database in a specified language. For more information about translations, see [Languages and Collations \(Analysis Services - Multidimensional Data\)](#).

Defining Translations

You can define translations in SQL Server Data Tools (SSDT) by using the appropriate designer for the Analysis Services object to be translated. Defining a translation creates a **Translation** object associated with the appropriate Analysis Services object that has the specified explicit literal values, in the specified language, for the properties of the associated Analysis Services object.

The following objects and properties in Analysis Services can have translations associated with them:

Object	Properties	Designer
Database	Caption, Description	Database Designer
Cube	Caption, Description	Cube Designer
Measure group	Caption	Cube Designer
Measure	Caption, DisplayFolder	Cube Designer
Cube dimension	Caption	Cube Designer
Perspective	Caption	Cube Designer
Key performance indicator (KPI)	Caption, Description, DisplayFolder	Cube Designer
Action	Caption	Cube Designer
Named set	Caption	Cube Designer
Calculated member	Caption	Cube Designer
Database dimension	Caption, AttributeAllMember	Dimension Designer

Object	Properties	Designer
Attribute	Caption, CaptionColumn¹, AttributeHierarchyDisplayFolder, NamingTemplate, MembersWithDataCaption	Dimension Designer
Hierarchy	Caption, AllMemberName	Dimension Designer
Level	Caption	Dimension Designer

¹ The **CaptionColumn** property of an attribute can be bound to a column in a data source view and can use a Windows collation other than that specified for the instance, unlike other translations.

Defining Attribute Translations

Translations associated with attributes in database dimensions are handled differently than other translations in the following ways:

- A column binding, instead of an explicit literal value, can be associated with the **CaptionColumn** property so that the member names of members for that attribute can be translated.
- A Windows collation other than the collation specified for the instance can be used so that members in the attribute can be appropriately sorted for the language specified in the translation.

You can use the **Attribute Data Translation** dialog box in SQL Server Data Tools (SSDT) to define translations for attributes in database dimensions. For more information about the **Attribute Data Translation** dialog box, see [Attribute Data Translation Dialog Box \(SSAS\)](#).

Resolving Translations

If a client application requests information in a specified language identifier, the Analysis Services instance attempts to resolve data and metadata for Analysis Services objects to the closest possible language identifier. If the client application does not specify a default language, or specifies the neutral locale identifier (0) or process default language identifier (1024), then Analysis Services uses the default language for the instance to return data and metadata for Analysis Services objects.

If the client application specifies a language identifier other than the default language identifier, the instance iterates through all available translations for all available objects. If the specified language identifier matches the language identifier of a translation, Analysis Services returns that translation. If a match cannot be found, Analysis Services attempts to use one of the following methods to return translations with a language identifier closest to the specified language identifier:

- For the following language identifiers, Analysis Services attempts to use an alternate language identifier if a translation for the specified language identifier is not defined:

Specified language identifier	Alternate language identifier
3076 - Chinese (Hong Kong SAR, PRC)	1028 - Chinese (Taiwan)
5124 - Chinese (Macao SAR)	1028 - Chinese (Taiwan)
1028 - Chinese (Taiwan)	Default language
4100 - Chinese (Singapore)	2052 - Chinese (PRC)
2074 - Croatian	Default language
3098 - Croatian (Cyrillic)	Default language

- For all other specified language identifiers, Analysis Services extracts the primary language of the specified language identifier and retrieves the language identifier indicated by Windows as the best match for the primary language. If a translation for the best match language identifier cannot be found, or if the specified language identifier is the best match for the primary language, then the default language is used.

See Also

[International Considerations for Analysis Services \(SSAS\)](#)

[Working with Languages and Collations \(SSAS\)](#)

Currency Conversions

Microsoft SQL Server Analysis Services uses a combination of features, guided by Multidimensional Expressions (MDX) scripts, to provide currency conversion support in cubes supporting multiple currencies.

Currency Conversion Terminology

The following terminology is used in Analysis Services to describe currency conversion functionality:

Pivot currency

The currency against which exchange rates are entered in the rate measure group.

Local currency

The currency used to store transactions on which measures to be converted are based.

The local currency can be identified by either:

- A currency identifier in the fact table stored with the transaction, as is commonly the case with banking applications where the transaction itself identifies the currency used for that transaction.
- A currency identifier associated with an attribute in a dimension table that is then associated with a transaction in the fact table, as is commonly the case in financial

applications where a location or other identifier, such as a subsidiary, identifies the currency used for an associated transaction.

Reporting currency

The currency to which transactions are converted from the pivot currency.



Note

For many-to-one currency conversions, the pivot currency and reporting currency are the same.

Currency dimension

A database dimension defined with the following settings:

- The **Type** property of the dimension is set to Currency.
- The **Type** property of one attribute for the dimension is set to CurrencyName.



Important

The values of this attribute must be used in all columns that should contain a currency identifier.

Rate measure group

A measure group in a cube, defined with the following settings:

- A regular dimension relationship exists between a currency dimension and the rate measure group.
- A regular dimension relationship exists between a time dimension and the rate measure group.
- Optionally, the **Type** property is set to ExchangeRate. While the Business Intelligence Wizard uses the relationships with the currency and time dimensions to identify likely rate measure groups, setting the **Type** property to ExchangeRate allows client applications to more easily identify rate measure groups.
- One or more measures, representing the exchange rates contained by the rate measure group.

Reporting currency dimension

The dimension, defined by the Business Intelligence Wizard after a currency conversion is defined, that contains the reporting currencies for that currency conversion. The reporting currency dimension is based on a named query, defined in the data source view on which the currency dimension associated with the rate measure group is based, from the dimension main table of the currency dimension. The dimension is defined with the following settings:

- The **Type** property of the dimension is set to Currency.
- The **Type** property of the key attribute for the dimension is set to CurrencyName.
- The **Type** property of one attribute within the dimension is set to CurrencyDestination, and the column bound to the attribute contains the currency identifiers that represent the reporting currencies for the currency conversion.

Defining Currency Conversions

You can use the Business Intelligence Wizard to define currency conversion functionality for a cube, or you can manually define currency conversions using MDX scripts.

Prerequisites

Before you can define a currency conversion in a cube using the Business Intelligence Wizard, you must first define at least one currency dimension, at least one time dimension, and at least one rate measure group. From these objects, the Business Intelligence Wizard can retrieve the data and metadata used to construct the reporting currency dimension and MDX script needed to provide currency conversion functionality.

Decisions

You need to make the following decisions before the Business Intelligence Wizard can construct the reporting currency dimension and MDX script needed to provide currency conversion functionality:

- Exchange rate direction
- Converted members
- Conversion type
- Local currencies
- Reporting currencies

Exchange Rate Directions

The rate measure group contains measures representing exchange rates between local currencies and the pivot currency (commonly referred to as the corporate currency). The combination of exchange rate direction and conversion type determines the operation performed on measures to be converted by the MDX script generated using the Business Intelligence Wizard. The following table describes the operations performed depending on the exchange rate direction and conversion type, based on the exchange rate direction options and conversion directions available in the Business Intelligence Wizard.

Exchange rate direction	Many-to-one	One-to-many	Many-to-many
n pivot currency to 1 sample currency	Multiply the measure to be converted by the exchange rate measure for the local currency in order to convert the measure into the pivot currency.	Divide the measure to be converted by the exchange rate measure for the reporting currency in order to convert the measure into the reporting currency.	Multiply the measure to be converted by the exchange rate measure for the local currency in order to convert the measure into the pivot currency, then divide the converted measure by the exchange rate

			measure for the reporting currency in order to convert the measure into the reporting currency.
n sample currency to 1 pivot currency	Divide the measure to be converted by the exchange rate measure for the local currency in order to convert the measure into the pivot currency.	Multiply the measure to be converted by the exchange rate measure for the reporting currency in order to convert the measure into the reporting currency.	Divide the measure to be converted by the exchange rate measure for the local currency in order to convert the measure into the pivot currency, then multiply the converted measure by the exchange rate measure for the reporting currency in order to convert the measure into the reporting currency.

You choose the exchange rate direction on the **Set currency conversion options** page of the Business Intelligence Wizard. For more information about setting conversion direction, see [International Considerations \(Analysis Services - Multidimensional Data\)](#).

Converted Members

You can use the Business Intelligence Wizard to specify which measures from the rate measure group are used to convert values for:

- Measures in other measure groups.
- Members of an attribute hierarchy for an account attribute in a database dimension.
- Account types, used by members of an attribute hierarchy for an account attribute in a database dimension.

The Business Intelligence Wizard uses this information within the MDX script generated by the wizard to determine the scope of the currency conversion calculation. For more information about specifying members for currency conversion, see [Select Members \(Business Intelligence Wizard\)](#).

Conversion Types

The Business Intelligence Wizard supports three different types of currency conversion:

- **One-to-many**

Transactions are stored in the fact table in the pivot currency, and then converted to one or more other reporting currencies.

For example, the pivot currency can be set to United States dollars (USD), and the fact table stores transactions in USD. This conversion type converts these transactions from the pivot currency to the specified reporting currencies. The result is that transactions can be stored in the specified pivot currency and viewed either in the specified pivot currency or in any of the reporting currencies specified in the reporting currency dimension defined for the currency conversion.

- **Many-to-one**

Transactions are stored in the fact table in local currencies, and then converted into the pivot currency. The pivot currency serves as the only specified reporting currency in the reporting currency dimension.

For example, the pivot currency can be set to United States dollars (USD), and the fact table stores transactions in euros (EUR), Australian dollars (AUD), and Mexican pesos (MXN). This conversion type converts these transactions from their specified local currencies to the pivot currency. The result is that transactions can be stored in the specified local currencies and viewed in the pivot currency, which is specified in the reporting currency dimension defined for the currency conversion.

- **Many-to-many**

Transactions are stored in the fact table in local currencies. The currency conversion functionality converts such transactions into the pivot currency, and then to one or more other reporting currencies.

For example, the pivot currency can be set to United States dollars (USD), and the fact table stores transactions in euros (EUR), Australian dollars (AUD), and Mexican pesos (MXN). This conversion type converts these transactions from their specified local currencies to the pivot currency, and then the converted transactions are converted again from the pivot currency to the specified reporting currencies. The result is that transactions can be stored in the specified local currencies and viewed either in the specified pivot currency or in any of the reporting currencies that are specified in the reporting currency dimension defined for the currency conversion.

Specifying the conversion type allows the Business Intelligence Wizard to define the named query and dimension structure of the reporting currency dimension, as well as the structure of the MDX script defined for the currency conversion.

Local Currencies

If you choose a many-to-many or many-to-one conversion type for your currency conversion, you need to specify how to identify the local currencies from which the MDX script generated by the Business Intelligence Wizard performs the currency conversion calculations. The local currency for a transaction in a fact table can be identified in one of two ways:

- The measure group contains a regular dimension relationship to the currency dimension. For example, in the Adventure Works DW Multidimensional 2012 sample Analysis Services

database, the Internet Sales measure group has a regular dimension relationship to the Currency dimension. The fact table for that measure group contains a foreign key column that references the currency identifiers in the dimension table for that dimension. In this case, you can select the attribute from the currency dimension that is referenced by the measure group to identify the local currency for transactions in the fact table for that measure group. This situation most often occurs in banking applications, where the transaction itself determines the currency used within the transaction.

- The measure group contains a referenced dimension relationship to the currency dimension, through another dimension that directly references the currency dimension. For example, in the Adventure Works DW Multidimensional 2012 sample Analysis Services database, the Financial Reporting measure group has a referenced dimension relationship to the Currency dimension through the Organization dimension. The fact table for that measure group contains a foreign key column that references members in the dimension table for the Organization dimension. The dimension table for the Organization dimension, in turn, contains a foreign key column that references the currency identifiers in the dimension table for the Currency dimension. This situation most often occurs in financial reporting applications, where the location or subsidiary for a transaction determines the currency for the transaction. In this case, you can select the attribute that references the currency dimension from the dimension for the business entity.

Reporting Currencies

If you choose a many-to-many or one-to-many conversion type for your currency conversion, you need to specify the reporting currencies for which the MDX script generated by the Business Intelligence Wizard performs the currency conversion calculations. You can either specify all the members of the currency dimension related to the rate measure group, or select individual members from the dimension.

The Business Intelligence Wizard creates a reporting currency dimension, based on a named query constructed from the dimension table for the currency dimension using the selected reporting currencies.



Note

If you select the one-to-many conversion type, a reporting currency dimension is also created. The dimension contains only one member representing the pivot currency, because the pivot currency is also used as the reporting currency for a one-to-many currency conversion.

A separate reporting currency dimension is defined for each currency conversion defined in a cube. You can change the name of the reporting currency dimensions after creation, but if you do so you must also update the MDX script generated for that currency conversion to ensure that the correct name is used by the script command when referencing the reporting currency dimension.

Defining Multiple Currency Conversions

Using the Business Intelligence Wizard, you can define as many currency conversions as needed for your business intelligence solution. You can either overwrite an existing currency conversion

or append a new currency conversion to the MDX script for a cube. Multiple currency conversions defined in a single cube provide flexibility in business intelligence applications that have complex reporting requirements, such as financial reporting applications that support multiple, separate conversion requirements for international reporting.

Identifying Currency Conversions

The Business Intelligence Wizard identifies each currency conversion by framing the script commands for the currency conversion in the following comments:

```
//<Currency conversion>
...
[MDX statements for the currency conversion]
...
//</Currency conversion>
```

If you change or remove these comments, the Business Intelligence Wizard is unable to detect the currency conversion, so you should not change these comments.

The wizard also stores metadata in comments within these comments, including the creation date and time, the user, and the conversion type. These comments should also not be changed because the Business Intelligence Wizard uses this metadata when displaying existing currency conversions.

You can change the script commands contained in a currency conversion as needed. If you overwrite the currency conversion, however, your changes will be lost.

See Also

[International Considerations for Analysis Services \(SSAS\)](#)

Client Applications

When working with client applications in multiple languages for Microsoft SQL Server Analysis Services, the following general guidelines allow you to increase the portability of your business intelligence solution.

Handling Translations

Translations provide display information for the names of Analysis Services objects, but the identifiers for the same objects are not translated. Whenever possible, use the identifiers and keys for Analysis Services objects instead of the translated captions and names. For example, use member keys instead of member names for Multidimensional Expressions (MDX) statements and scripts to ensure portability across multiple languages.

Handling Date and Time Values

When you perform month and day-of-week comparisons and operations, use the numeric date and time parts instead of date and time part strings. Date and time part strings are determined in part by the language identifier specified for the instance, and the current translation provided by the instance for the members of the time dimension. Take advantage of date and time functions in MDX for negotiating time dimensions, as well as the Visual Basic for Applications

(VBA) date and time functions for returning numeric date and time parts instead of the name strings. Use the literal date and time part strings when returning results to be displayed to a user, because the strings are frequently more meaningful than a numeric representation. However, do not code any logic that depends on the displayed names being in a specific language.

See Also

[International Considerations \(Analysis Services - Multidimensional Data\)](#)

Developing with ADOMD.NET

ADOMD.NET is a Microsoft .NET Framework data provider that is designed to communicate with Microsoft SQL Server Analysis Services. ADOMD.NET uses the XML for Analysis protocol to communicate with analytical data sources by using either TCP/IP or HTTP connections to transmit and receive SOAP requests and responses that are compliant with the XML for Analysis specification. Commands can be sent in Multidimensional Expressions (MDX), Data Mining Extensions (DMX), Analysis Services Scripting Language (ASSL), or even a limited syntax of SQL, and may not return a result. Analytical data, key performance indicators (KPIs), and mining models can be queried and manipulated by using the ADOMD.NET object model. By using ADOMD.NET, you can also view and work with metadata either by retrieving OLE DB-compliant schema rowsets or by using the ADOMD.NET object model.

The ADOMD.NET data provider is represented by the **Microsoft.AnalysisServices.AdomdClient** namespace.

In This Section

Topic	Description
Analysis Services Data Access Interfaces (Analysis Services - Multidimensional Data)	Describes how to use ADOMD.NET client objects to retrieve data and metadata from analytical data sources.
ADOMD.NET Server Programming	Describes how to use ADOMD.NET server objects to create stored procedures and user-defined functions.
Redistributing ADOMD.NET	Describes the process of redistributing ADOMD.NET.
N:Microsoft.AnalysisServices.AdomdClient	Details the objects that are contained in the Microsoft.AnalysisServices.AdomdClient namespace.

See Also

[Multidimensional Expressions \(MDX\) Reference](#)

[Data Mining Extensions \(DMX\) Reference](#)

[Schema Rowsets](#)

[ASSL](#)

[Analysis Services Data Access Interfaces \(SSAS\)](#)

ADOMD.NET Client Programming

The ADOMD.NET client components reside within the **Microsoft.AnalysisServices.AdomdClient** namespace (in `microsoft.analysiservices.adomdclient.dll`). These client components provide the functionality for client and middle-tier applications to easily query data and metadata from an analytical data store, such as Microsoft SQL Server Analysis Services.

Using the ADOMD.NET Client Objects

In querying an analytical data source, there are a set of common tasks that need to be performed. The following table represents the common tasks in which you use the ADOMD.NET client objects to perform such a query.

Task	Description
Establishing Connections	In ADOMD.NET, you use an T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object to establish connections with analytical data sources, such as Analysis Services databases. You can use the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object to run commands, retrieve data, and retrieve metadata from the analytical data source.
Retrieving Metadata	After a connection has been established, you can use a wide variety of objects to retrieve information about the underlying data source. This functionality allows applications to adapt to the data source to which they have connected.
Executing Commands	The T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand object provides the interfaces necessary for running commands against the underlying analytical data source.

Task	Description
Retrieving Data	After a command runs, data could be retrieved and parsed using either the T:Microsoft.AnalysisServices.AdomdClient.CellSet , T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader , or System.XmlReader objects.
Performing Transactions	All of the actions listed in the previous rows of this table can take place within a read-committed transaction, in which shared locks are held while the data is being read to avoid dirty reads. The data can still be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. The T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction object provides the transaction functionality in ADOMD.NET.

Interaction with the ADOMD.NET object hierarchy typically starts with one or more of the objects in the topmost layer, as described in the following table.

To	Use this object
Connect to an analytical data source	T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection The T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object represents both a connection to a data source and the data source metadata. For example, you can connect to a Microsoft SQL Server Analysis Services local cube (.cub) file, and then examine the P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Cubes property to obtain metadata about the cubes present on the analytical data source. This object also represents the implementation of the IDbConnection interface, an interface that is required by all .NET Framework data providers.
Discover the data mining capabilities of the data source	T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection The T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object exposes several mining collections: The T:Microsoft.AnalysisServices.AdomdClient.Mining

To	Use this object
	ModelCollection contains a list of every mining model in the data source. - The T:Microsoft.AnalysisServices.AdomdClient.MiningServiceCollection provides information about the available mining algorithms. - The T:Microsoft.AnalysisServices.AdomdClient.MiningStructureCollection exposes information about the mining structures on the server.
Query the data source	T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand The T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand object represents the statement or query that will be sent to the server. Once a connection is established to a data source, you use a T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand object to run statements in the supported language, such as Multidimensional Expressions (MDX) or Data Mining Data Mining Extensions (DMX). You can also use a T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand object to return results in the form of T:Microsoft.AnalysisServices.AdomdClient.CellSet or T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader objects.
Retrieve data in a fast, efficient way	T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader The T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader can be created with a call to the M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.Execute or M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteReader method of an T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand object. This object implements the IDbDataReader interface from the System.Data namespace of the .NET Framework class library.
Retrieve analytical data with the highest	T:Microsoft.AnalysisServices.AdomdClient.CellSet The T:Microsoft.AnalysisServices.AdomdClient.CellSet

To	Use this object
amount of metadata	can be created with a call to the M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.Execute or M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteCellSet method of an T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand. Once an T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand has returned a T:Microsoft.AnalysisServices.AdomdClient.CellSet, you can then examine the analytical data contained by the T:Microsoft.AnalysisServices.AdomdClient.CellSet.
Retrieve metadata about cubes, such as available dimensions, measures, named sets, and so on	T:Microsoft.AnalysisServices.AdomdClient.CubeDef The T:Microsoft.AnalysisServices.AdomdClient.CubeDef represents metadata about a cube. You reference the T:Microsoft.AnalysisServices.AdomdClient.CubeDef from the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.
Retrieve data using the System.Data.IDbDataAdapter interface	T:Microsoft.AnalysisServices.AdomdClient.AdomdDataAdapter The T:Microsoft.AnalysisServices.AdomdClient.AdomdDataAdapter provides read-only support for existing .NET Framework client applications.

See Also

[ADOMD.NET Server Programming](#)

[ADOMD.NET](#)

ADOMD.NET Client Functionality

ADOMD.NET, as with other Microsoft .NET Framework data providers, serves as a bridge between an application and a data source. However, ADOMD.NET is unlike other .NET Framework data providers in that ADOMD.NET works with analytical data. To work with analytical data, ADOMD.NET supports functionality that is very different than other .NET Framework data providers. ADOMD.NET not only allows you to retrieve data, but also to retrieve metadata and change the structure of the analytical data store:

Retrieving Metadata

Applications can learn more about the data that can be retrieved from the data source through metadata retrieval, using either schema rowsets or the object model. Information such as the types of each key performance indicator (KPI) that are available, the dimensions in a cube, and the parameters needed by the mining models are all discoverable. Metadata is most important to *dynamic* applications that require user input to determine the type, depth, and scope of data to be retrieved. Examples include Query Analyzer, Microsoft Excel, and other querying tools. Metadata is less critical to *static* applications that perform a predefined set of actions.

For more information: [ADOMD.NET Client Programming](#).

Retrieving Data

Data retrieval is the actual retrieval of the information stored in the data source. Data retrieval is the primary function of "static" applications, which know the structure of the data source. Data retrieval is also the end result of "dynamic" applications. The value of the KPI at a given time of day, the number of bicycles sold within the last hour for each store, and the factors governing the annual performance of employees are all examples of data that can be retrieved. Retrieving data is vital for any querying application.

For more information: [Retrieving Data](#).

Changing the Structure of Analytical Data

ADOMD.NET can also be used to actually change the structure of the analytical data store. Though this is usually done through the Analysis Management Objects (AMO) object model, you can use ADOMD.NET to send Analysis Services Scripting Language (ASSL) commands to create, alter, or delete objects on the server.

For more information: [Executing Commands, Analysis Management Objects \(AMO\), Analysis Services Scripting Language \(ASSL\)](#)

Retrieving metadata, retrieving data, and changing data structure each occur at a specific point in the workflow of a typical ADOMD.NET application.

Typical Process Flow

Traditional ADOMD.NET applications usually follow the same workflow when working with an analytical database:

1. First, a connection is made to the database, using the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object. When you open the connection, the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object exposes metadata about the server to which you have connected. In a dynamic application, some of this information is typically shown to the user so that the user can make a selection, such as which cube to query. The connection created during this step can be reused multiple times by the application, reducing overhead.

For more information: [Establishing Connections in ADOMD.NET](#)

2. Once a connection has been made, a dynamic application would then query the server for more specific metadata. For a static application, the programmer knows in advance which objects the application will be querying, and thus will not need to retrieve this metadata. Metadata that is retrieved can be used by the application and the user for the next step.
For more information: [Retrieving Metadata from an Analytical Data Source](#)
3. The application then runs a command against the server. This command can be for the purpose of retrieving additional metadata, retrieving data, or modifying the database structure. For any of these tasks, the application could use a previously-determined query, or make use of newly retrieved metadata to create additional queries.
For more information: [Retrieving Metadata from an Analytical Data Source](#), [Retrieving Data from an Analytical Data Source](#), [Executing Commands Against an Analytical Data Source](#)
4. Once the command has been sent to the server, the server begins to return the metadata or data to the client. This information can be viewed by using a **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object, an **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object, or a **System.XmlReader** object.

To illustrate this traditional workflow, the following example contains a method that opens a connection to the database, executes a command against a known cube, and retrieves the results into a cellset. The cellset then returns a tab-delimited string containing column headers, row headers, and cell data.

Adomd.NetClient#ReturnCommandUsingCellSet

See Also

[ADOMD.NET Client Programming](#)

Migrating From ADO MD To ADOMD.NET

The ADOMD.NET library is similar to the ActiveX Data Objects Multidimensional (ADO MD) library, an extension of the ActiveX Data Objects (ADO) library that is used to access multidimensional data in Component Object Model (COM)–based client applications. ADO MD provides easy access to multidimensional data from unmanaged languages such as C++ and Microsoft Visual Basic. ADOMD.NET provides easy access to analytical (both multidimensional and data mining) data from managed languages such as Microsoft C# and Microsoft Visual Basic .NET. Additionally, ADOMD.NET provides an enhanced metadata object model.

Migrating existing client applications from ADO MD to ADOMD.NET is easy, but there are several important differences regarding migration:

To provide connectivity and data access to client applications

ADO MD	ADOMD.NET
Requires references to both Adodb.dll and	Requires a single reference to

Adomd.dll.	Microsoft.AnalysisServices.AdomdClient.dll.

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** class provides connectivity support, in addition to access to metadata.

To retrieve metadata for multidimensional objects

ADO MD	ADOMD.NET
Use the Catalog class.	Use the P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Cubes property of the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection .

To run queries and return cellset objects

ADO MD	ADOMD.NET
Use the CellSet class.	Use the T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand class.

To access the metadata that is used to display a cellset

ADO MD	ADOMD.NET
Use the Position class.	Use the T:Microsoft.AnalysisServices.AdomdClient.Set and T:Microsoft.AnalysisServices.AdomdClient.Tuple objects.

nNote

The **T:Microsoft.AnalysisServices.AdomdClient.Position** class is supported for backward compatibility.

To retrieve mining model metadata

ADO MD	ADOMD.NET
No class available.	Use one of the data mining collections: The

	T:Microsoft.AnalysisServices.AdomdClient.MiningModelCollection contains a list of every mining model in the data source. • The T:Microsoft.AnalysisServices.AdomdClient.MiningServiceCollection provides information about the available mining algorithms. • The T:Microsoft.AnalysisServices.AdomdClient.MiningStructureCollection exposes information about the mining structures on the server.

To highlight these differences the following migration example compares an existing ADO MD application to an equivalent ADOMD.NET application.

Looking at a Migration Example

Both the existing ADO MD and equivalent ADOMD.NET code examples shown in this section perform the same set of actions: creating a connection, running a Multidimensional Expressions (MDX) statement, and retrieving metadata and data. However, these two sets of code do not use the same objects to perform those tasks.

Existing ADO MD Code

The following code example, drawn from ADO MD 2.8 documentation, is written in Microsoft Visual Basic® 6.0 and uses ADO MD to demonstrate how to connect to and query a Microsoft SQL Server data source. This ADO MD example uses the following objects:

- Creates a connection from a **Catalog** object.
- Runs the Multidimensional Expressions (MDX) statement using the **Cellset** object.
- Retrieves the metadata and data from the **Position** object, retrieved from the **Cellset** object.

```
Private Sub cmdCellSettoDebugWindow_Click()

Dim cat As New ADOMD.Catalog

Dim cst As New ADOMD.Cellset

Dim i As Integer

Dim j As Integer

Dim k As Integer

Dim strServer As String

Dim strSource As String

Dim strColumnHeader As String

Dim strRowText As String


On Error GoTo Error_cmdCellSettoDebugWindow_Click

Screen.MousePointer = vbHourglass
```

```

' *-----
' * Set server to local host.
' *-----

    strServer = "LOCALHOST"

' *-----
' * Set MDX query string source.
' *-----

    strSource = strSource & "SELECT "
    strSource = strSource & "{[Measures].members} ON COLUMNS,"
    strSource = strSource & _
        "NON EMPTY [Store].[Store City].members ON ROWS"
    strSource = strSource & " FROM Sales"

' *-----
' * Set active connection.
' *-----

    cat.ActiveConnection = "Data Source=" & strServer & _
        ";Provider=msolap;"

' *-----
' * Set cellset source to MDX query string.
' *-----

    cst.Source = strSource

' *-----
' * Set cellset active connection to current connection
' *-----

    Set cst.ActiveConnection = cat.ActiveConnection

' *-----
' * Open cellset.
' *-----

```

```

cst.Open

'*-----
'* Allow space for row header text.
'*-----

strColumnHeader = vbTab & vbTab & vbTab & vbTab & vbTab & vbTab

'*-----
'* Loop through column headers.
'*-----

    For i = 0 To cst.Axes(0).Positions.Count - 1
        strColumnHeader = strColumnHeader & _
            cst.Axes(0).Positions(i).Members(0).Caption & vbTab & _
                vbTab & vbTab & vbTab
    Next

    Debug.Print vbTab & strColumnHeader & vbCrLf

'*-----
'* Loop through row headers and provide data for each row.
'*-----

    strRowText = ""
    For j = 0 To cst.Axes(1).Positions.Count - 1
        strRowText = strRowText & _
            cst.Axes(1).Positions(j).Members(0).Caption & vbTab & _
                vbTab & vbTab & vbTab
        For k = 0 To cst.Axes(0).Positions.Count - 1
            strRowText = strRowText & cst(k, j).FormattedValue & _
                vbTab & vbTab & vbTab & vbTab
        Next
        Debug.Print strRowText & vbCrLf
        strRowText = ""
    Next

```

```

    Screen.MousePointer = vbDefault
Exit Sub

Error_cmdCellSettoDebugWindow_Click:

    Beep

    Screen.MousePointer = vbDefault

    MsgBox "The following error has occurred:" & vbCrLf & _
        Err.Description, vbCritical, " Error!"

    Exit Sub

End Sub

```

Equivalent ADOMD.NET Code

The following example, written in Visual Basic .NET and using ADOMD.NET, demonstrates how to perform the same actions as the previous Visual Basic 6.0 example. The major difference between the following example and the ADO MD example shown earlier is the objects that are used to perform the actions. The ADOMD.NET example uses the following objects:

- Creates a connection with an **AdomdConnection** object.
- Runs the MDX statement using an **AdomdCommand** object.
- Retrieves the metadata and data from the **Set** object, retrieved from the **Cellset** object.

```

Private Sub DisplayCellSetInOutputWindow()

    Dim conn As AdomdConnection

    Dim cmd As AdomdCommand

    Dim cst As CellSet

    Dim i As Integer

    Dim j As Integer

    Dim k As Integer

    Dim strServer As String = "LOCALHOST"

    Dim strSource As String = "SELECT [Measures].members ON COLUMNS, " & _
        "NON EMPTY [Store].[Store City].members ON ROWS FROM SALES"

    Dim strOutput As New System.IO.StringWriter

    '*-----
    '* Open connection.
    '*-----

    Try

```

```

' Create a new AdomdConnection object, providing the connection
' string.

conn = New AdomdConnection("Data Source=" & strServer & _
";Provider=msolap;")

' Open the connection.

conn.Open()

Catch ex As Exception

    Throw New ApplicationException( _
        "An error occurred while connecting.")

End Try

Try

'*-----

'* Open cellset.

'*-----

    ' Create a new AdomdCommand object, providing the MDX query string.

    cmd = New AdomdCommand(strSource, conn)

    ' Run the command and return a CellSet object.

    cst = cmd.ExecuteCellSet()

'*-----

'* Concatenate output.

'*-----

' Include spacing to account for row headers.

strOutput.Write(vbTab, 6)

' Iterate through the first axis of the CellSet object and
' retrieve column headers.

For i = 0 To cst.Axes(0).Set.Tuples.Count - 1

    strOutput.Write(cst.Axes(0).Set.Tuples(i).Members(0).Caption)

    strOutput.Write(vbTab, 4)

Next

```



```

strOutput.WriteLine()

' Iterate through the second axis of the CellSet object and
' retrieve row headers and cell data.
For j = 0 To cst.Axes(1).Set.Tuples.Count - 1
    ' Append the row header.
    strOutput.Write(cst.Axes(1).Set.Tuples(j).Members(0).Caption)
    strOutput.Write(vbTab, 4)

    ' Append the cell data for that row.
    For k = 0 To cst.Axes(0).Set.Tuples.Count - 1
        strOutput.Write(cst.Cells(k, j).FormattedValue)
        strOutput.Write(vbTab, 4)
    Next
    strOutput.WriteLine()
Next

' Display the output.
Debug.WriteLine(strOutput.ToString)

'*-----
'* Release resources.
'*-----

    conn.Close()
Catch ex As Exception
    ' Ignore or handle errors.
Finally
    cst = Nothing
    cmd = Nothing
    conn = Nothing
End Try
End Sub

```

Establishing Connections in ADOMD.NET

In ADOMD.NET, you use the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object to open connections with analytical data sources, such as Microsoft SQL Server Analysis Services databases. When the connection is no longer needed, you should explicitly close the connection.

Opening a Connection

To open a connection in ADOMD.NET, you must first specify a connection string to a valid analytical data source and database. Then, you must explicitly open the connection to that data source.

Specifying a Multidimensional Data Source

To specify an analytical data source and database, you set the

P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString property of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object. The connection string specified for the **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString** property is an OLE DB–compliant string. ADOMD.NET uses the specified connection string to determine how to connect to the server.

The **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString** property can be set on either an existing **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object or during the creation an instance of an

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object. The following code demonstrates how to set the

P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString property when you create an **:Microsoft.AnalysisServices.AdomdClient.AdomdConnection**:

```
Dim advwrksConnection As New AdomdConnection("Data Source=localhost;Catalog=AdventureWorksAS")
System.Diagnostics.Debug.WriteLine(advwrksConnection.ConnectionString)

AdomdConnection advwrksConnection = new AdomdConnection("Data
Source=localhost;Catalog=AdventureWorksAS");

System.Diagnostics.Debug.WriteLine(advwrksConnection.ConnectionString);
```

Opening a Connection to the Data Source

After you have specified the connection string, you must use the

M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Open method to open the connection. When you open a **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object, you can set various levels of security for the connection. The security level that is used for the connection depends on the value of the **ProtectionLevel** connection string setting. For more information about opening secure connections in ADOMD.NET, see [ADOMD.NET Client Programming](#).

Working with a Connection

Each open connection exists in a session, which provides support for stateful operations. A session can be shared by more than one open connection. Sharing a session enables more than

one client to share the same context. For more information, see [Connections and Sessions in ADOMD.NET](#).

You can use an open connection to retrieve metadata, data, and run commands. For more information, see [Retrieving Metadata](#), [Retrieving Data](#), and [Executing Commands](#).

When the connection is open, you can retrieve data, retrieve metadata, and run commands from within a read-committed transaction, in which shared locks are held while the data is being read to avoid dirty reads. The data can still be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. For more information, see [Performing Transactions](#).

Closing a Connection

We recommended that you explicitly close an

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object as soon as you no longer need the connection. To explicitly close the connection, you use the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Close** and **Overload:System.ComponentModel.Component.Dispose** methods of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object.

A connection that is not explicitly closed, but is allowed to fall out of scope, may not release server resources quickly enough to enable high-concurrency Analysis Services client applications to efficiently open new connections. Depending on how you created the connection, the session used by the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object can remain active if the connection is not explicitly closed.

For more information about sessions, see [Connections and Sessions in ADOMD.NET](#).

Important

In the **Finalize** method of any implemented class, do not call the **Close** or **Dispose** methods of an **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object, **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object, or any other managed object. In a finalizer, only release unmanaged resources that are directly owned by the implemented class. If the implemented class does not own any unmanaged resources, do not include a **Finalize** method in the class definition.

See Also

[ADOMD.NET Programming](#)

Establishing Secure Connections in ADOMD.NET

When you use a connection in ADOMD.NET, the security method that is used for the connection depends on the value of the **ProtectionLevel** property of the connection string used when you call the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Open** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection**.

The **ProtectionLevel** property offers four levels of security: unauthenticated, authenticated, signed, and encrypted. The following table describes these various security levels.

Note

If you choose to use database connection pooling, the database will not be able to manage security. This is because database connection pooling requires that the connection string be identical to pool connections. Therefore, you must manage security elsewhere.

Security Level	ProtectionLevel Value
unauthenticated connection An unauthenticated connection does no form of authentication. This kind of connection represents the most widely supported, but least secure, form of connection.	None
authenticated connection An authenticated connection authenticates the user who is making the connection, but does not secure additional communications. This kind of connection is useful in that you can establish the identity of the user or application that is connecting to an analytical data source.	Connect
signed connection A signed connection authenticates the user who is requesting the connection, and then makes sure that transmissions are not modified. This kind of connection is useful when the authenticity of the transferred data must be verified. However, a signed connection only prevents the content of the data packet from being modified. The content can still be viewed in transit.  Note A signed connection is only supported by the XML for Analysis provider supplied by Microsoft SQL Server Analysis Services.	Pkt Integrity or PktIntegrity
encrypted connection An encrypted connection is the	Pkt Privacy or PktPrivacy

Security Level	ProtectionLevel Value
default connection type used by ADOMD.NET. This kind of connection authenticates the user who is requesting the connection, and then also encrypts the data that is transmitted. An encrypted connection is the securest form of connection that can be created by ADOMD.NET. The content of the data packet cannot be viewed or modified, thereby protecting data during transit.  Note An encrypted connection is only supported by the XML for Analysis provider supplied by SQL Server Analysis Services.	

However, not all levels of security are available for all kinds of connections:

- A TCP connection can use any one of the four levels of security. In fact, a TCP connection, when you use it with Windows Integrated Security, offers the securest method of connecting to an analytical data source.
- An HTTP connection can only be an authenticated connection. Therefore, the **ProtectionLevel** property must be set to **Connect**.
- An HTTPS connection can only be an encrypted connection. Therefore, the **ProtectionLevel** property must be set to **Pkt Privacy** or **PktPrivacy**.

Securing TCP Connections

For a TCP connection, the **ProtectionLevel** property supports all four levels of security, as shown in the following table.

ProtectionLevel Value	Use with TCP Connection?	Results
None	Yes	Specifies an unauthenticated connection. A TCP stream is requested from the provider, but there is no form of authentication performed on the user who is

ProtectionLevel Value	Use with TCP Connection?	Results
		requesting the stream.
Connect	Yes	Specifies an authenticated connection. A TCP stream is requested from the provider, and then the security context of the user who is requesting the stream is authenticated against the server: • If authentication succeeds, no other action is taken. • If authentication fails, the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object disconnects from the multidimensional data source and an exception is thrown. After authentication succeeds or fails, the security context that is used to authenticate the connection is disposed.
Pkt Integrity or PktIntegrity	Yes	Specifies a signed connection. A TCP stream is requested from the provider, and then the security context of the user who is requesting the stream is authenticated against the server: • If authentication succeeds, the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object closes the existing TCP stream and opens a signed TCP stream to handle all requests. Each request for data or metadata is authenticated by using the security context that was used to open the connection. Additionally, each packet is digitally signed to make sure that the payload of the TCP packet has not been changed in any way. • If authentication fails, the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object disconnects from the multidimensional data source and an exception is thrown.
Pkt Privacy or PktPrivacy	Yes	Specifies an encrypted connection.  Note You can also specify an encrypted connection by not setting the ProtectionLevel property in the connection string. A TCP stream is requested from the provider, and then the security context of the user requesting the stream is

ProtectionLevel Value	Use with TCP Connection?	Results
		authenticated against the server: - If authentication succeeds, the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object closes the existing TCP stream and opens up an encrypted TCP stream to handle all requests. Each request for data or metadata is authenticated by using the security context that was used to open the connection. Additionally, the payload of each TCP packet is encrypted by using the highest encryption method supported by both the provider and the multidimensional data source. - If authentication fails, the T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object disconnects from the multidimensional data source and an exception is thrown.

Using Windows Integrated Security for the Connection

Windows Integrated Security is the securest way of establishing and securing a connection to an instance of Analysis Services. Windows Integrated Security does not reveal security credentials, such as a user name or password, during the authentication process, but instead uses the security identifier of the currently running process to establish identity. For most client applications, this security identifier represents the identity of the currently logged-on user.

To use Windows Integrated Security, the connection string requires the following settings:

- For the **Integrated Security** property, either do not set this property or set this property to **SSPI**.



Note

Windows Integrated Security is only available for TCP connections because HTTP connections must use the **Basic** setting for the **Integrated Security** property.

- For the **ProtectionLevel** property, set this property to **Connect**, **Pkt Integrity**, or **Pkt Privacy**.

Securing HTTP Connections

HTTPS and Secure Sockets Layer (SSL) can be used to externally secure HTTP communications with an analytical data source.

Because an XMLA provider only uses secure HTTP, an HTTP connection in ADOMD.NET must be a signed connection, as shown in the following table.

ProtectionLevel Value	Use with HTTP or HTTPS
None	No
Connect	HTTP
Pkt Integrity or PktIntegrity	No
Pkt Privacy or PktPrivacy	HTTPS

Opening a Secure HTTP Connection

The following example demonstrates how to use ADOMD.NET to open an HTTP connection for the **AdventureWorksAS** sample Analysis Services database:

```
Public Function GetAWEncryptedConnection( _
    Optional ByVal serverName As String = "https://localhost/isapy/msmdpump.dll") _
    As AdomdConnection

    Dim strConnectionString As String = ""
    Dim objConnection As New AdomdConnection

    Try
        ' To establish an encrypted connection, set the
        ' ProtectionLevel setting to PktPrivacy.
        strConnectionString = "DataSource=" & serverName & ";" & _
            "Catalog=AdventureWorksAS;" & _
            "ProtectionLevel=PktPrivacy;"

        ' Note that username and password are not supplied here.
        ' The current security context is used for authentication
        ' purposes.

        objConnection.ConnectionString = strConnectionString
        objConnection.Open()
    Catch ex As Exception
        objConnection = Nothing
        Throw ex
    Finally
```

```

        ' Return the encrypted connection.

        GetAWEncryptedConnection = objConnection

    End Try

End Function

```

See Also

[Establishing Connections in ADOMD.NET](#)

Working with Connections and Sessions in ADOMD.NET

In XML for Analysis (XMLA), sessions provide support for stateful operations during analytical data access. Sessions frame the scope and context of commands and transactions for an analytical data source. The XMLA elements used to manage sessions are [BeginSession](#), [Session](#), and [EndSession](#).

ADOMD.NET uses these three XMLA session elements when you start a session, perform queries or retrieve data during the session, and close a session.

Starting a Session

The **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.SessionID** property of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object contains the identifier of the active session associated with the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object. By using this property correctly, you can effectively control both client and server statefulness in your application:

- If the **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.SessionID** property is not set to a valid session ID when the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Open** method is called, the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object requests a new session ID from the provider. ADOMD.NET initiates a session by sending an XMLA **BeginSession** header to the provider. If ADOMD.NET is successful in starting a session, ADOMD.NET sets the value of the **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.SessionID** property to the session ID of the newly created session.
- If the **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.SessionID** property is set to a valid session ID when the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Open** method is called, the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object tries to connect to the specified session.

If the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object cannot connect to the specified session, or if the provider does not support sessions, an exception is thrown.

Note

After you have had ADOMD.NET create a session, you can connect multiple **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** objects to that single active

session, or you can disconnect a single

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object from that session and reconnect that object to another session.

Working in a Session

After ADOMD.NET connects the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object to a valid session, ADOMD.NET will send an XMLA **Session** header to the provider with every request for data or metadata made by an application. Every request will have the session ID set to the value of the **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.SessionID** property.

A session ID does not guarantee that a session remains valid. If the session expires (for example, if the session times out or the connection is lost), the provider can choose to end and roll back the actions of that session. If this occurs, all subsequent method calls from the

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object will throw an exception.

Because exceptions are thrown only when the next request is sent to the provider, not when the session expires, your application must be able to handle these exceptions any time that your application retrieves data or metadata from the provider.

Closing a Session

If the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Close** method is called without specifying the value of the `endSession` parameter, or if the `endSession` parameter is set to `True`, both the connection to the session and the session associated with the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object are closed. To close a session, ADOMD.NET sends an XMLA **EndSession** header to the provider, with the session ID set to the value of the **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.SessionID** property.

If the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Close** method is called with the `endSession` parameter set to `False`, the session associated with the

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object remains active but the connection to the session is closed.

Example of Managing a Session

The following example demonstrates how to open a connection, create a session, and close the connection while keeping the session open in ADOMD.NET:

```
Public Function CreateSession(ByVal connectionString As String) As String

    Dim strSessionID As String = ""

    Dim objConnection As New AdomdConnection

    Try

        ' First, try to connect to the specified data source.

        ' If the connection string is not valid, or if the specified

        ' provider does not support sessions, an exception is thrown.
```

```

objConnection.ConnectionString = connectionString
objConnection.Open()

' Now that the connection is open, retrieve the new
' active session ID.
strSessionID = objConnection.SessionID

' Close the connection, but leave the session open.
objConnection.Close(False)

Return strSessionID

Finally
    objConnection = Nothing
End Try
End Function

static string CreateSession(string connectionString)
{
    string strSessionID = "";
    AdomdConnection objConnection = new AdomdConnection();
    try
    {
        /*First, try to connect to the specified data source.
        If the connection string is not valid, or if the specified
        provider does not support sessions, an exception is thrown. */
        objConnection.ConnectionString = connectionString;
        objConnection.Open();

        // Now that the connection is open, retrieve the new
        // active session ID.
        strSessionID = objConnection.SessionID;
        // Close the connection, but leave the session open.
        objConnection.Close(false);
        return strSessionID;
    }
}

```

```

finally
{
    objConnection = null;
}
}

```

See Also

[Establishing Connections in ADOMD.NET](#)

Performing Transactions in ADOMD.NET

In ADOMD.NET, you use the **T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction** object to manage transaction context for a given

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object. This functionality allows you to run several commands within the same context. Each command will read the same data without the read data changing between each command execution.



Note

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction** class is the implementation of the **System.Data.IDbTransaction** interface, part of the Microsoft .NET Framework Class Library and implemented by all .NET Framework data providers that support transactions.

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction** object only supports read-committed transactions, in which shared locks are held while the data is being read to avoid dirty reads.

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** is used to start the transaction. To use the transaction, commands are then run against the connection that started the transaction. When you are finished with the transaction, you can either be roll back or commit the transaction.

Starting a Transaction

You create an instance of an **T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction** object by calling the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.BeginTransaction** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object. The following example shows how to create an instance of the

T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction object:

```

Dim objTransaction As AdomdTransaction = objConnection.BeginTransaction()

AdomdTransaction objTransaction = objConnection.BeginTransaction();

```

Rolling Back a Transaction

To roll back an existing, incomplete transaction, you call the

M:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction.Rollback method of the

T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction object. If you call this method on an existing, complete transaction, an exception is thrown.

Committing a Transaction

After you call the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.BeginTransaction** method to start a transaction, you can complete the transaction by calling the **M:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction.Commit** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdTransaction** object. If this method is called on an existing, complete transaction, an exception is thrown.

See Also

[ADOMD.NET Client Programming](#)

[ADOMD.NET Programming](#)

Retrieving Metadata from an Analytical Data Source

Metadata is important to applications that retrieve and work with analytical data. When retrieving data from a relational data source, the dimensionality of such data is predictable, even with nested datasets. Result sets from a relational database are typically two-dimensional or scalar in structure. However, data retrieved from analytical data sources can be of variable dimensionality, organized along potentially deep hierarchies.

To handle the complexity of metadata retrieval from analytical data sources, ADOMD.NET provides two forms of metadata retrieval:

The Object Model

The ADOMD.NET object model is generally easier to use than schema rowsets. For most scenarios, you can access the metadata of various database objects just by using the object model. ADOMD.NET exposes the object model through the

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.

For more information: [Schema Rowsets](#)

Schema Rowsets

A complete, but more difficult approach to retrieving metadata is through using schema rowsets.

A schema rowset is an OLE DB rowset that encapsulates the description for all objects of a particular type in the database. Schema information in an analytical data source includes databases or catalogs available from the data source, cubes and mining models in a database, roles that exist for cubes at the data source, and so on. This metadata can be retrieved by using the

Overload:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet method, passing in either a **GUID** or an XML for Analysis (XMLA) name.

For more information: [Working With Schema Rowsets](#)

Each of these metadata retrieval methods access different types of metadata. The following table describes the different metadata available for each method, and the methods used to access it.

GUID (used in Schema Rowsets)	XMLA Name (used in Schema Rowsets)	ADOMD.NET Object Model
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Actions	MDSHEMA_ACTIONS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Catalogs	DBSCHEMA_CATALOGS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Columns	DBSCHEMA_COLUMNS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Connections	DISCOVER_CONNECTIONS	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Cubes	MDSHEMA_CUBES Rowset	AdomdConnection.Cubes
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.DataSources	DISCOVER_DATASOURCES Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.DBConnections	DISCOVER_DB_CONNECTIONS	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Dimensions	MDSHEMA_DIMENSIONS Rowset	AdomdConnection.Cubes[].Dimensions
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.DimensionStat	DISCOVER_DIMENSION_STAT	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Enumerators	DISCOVER_ENUMERATORS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Functions	MDSHEMA_FUNCTIONS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Hierarchies	MDSHEMA_HIERARCHIES Rowset	AdomdConnection.Cubes[].Dimensions[].Hierarchies
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.InputDataSources	MDSHEMA_INPUT_DATASOURCES Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Instances	DISCOVER_INSTANCE Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Jobs	DISCOVER_JOBS	

GUID (used in Schema Rowsets)	XMLA Name (used in Schema Rowsets)	ADOMD.NET Object Model
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Keywords	DISCOVER_KEYWORDS Rowset (OLE DB for OLAP)	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Kpis	MDSHEMA_KPIS Rowset	AdomdConnection.Cubes[].KPIs
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Levels	MDSHEMA_LEVELS Rowset	AdomdConnection.Cubes[].Dimensions[].Hierarchies[].Levels
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Literals	DISCOVER_LITERAL Rowset	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Locations	DISCOVER_LOCATIONS	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Locks	DISCOVER_LOCKS	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MasterKey	DISCOVER_MASTER_KEY	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MeasureGroup Dimensions	MDSHEMA_MEASUREGROUP_DIMENSIONS	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MeasureGroups	MDSHEMA_MEASUREGROUPS Rowset	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Measures	MDSHEMA_MEASURES Rowset	AdomdConnection.Cubes[].Measures
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MemberProperties	MDSHEMA_PROPERTIES Rowset	PropertyCollection available from most major ADOMD.NET objects.
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Members	MDSHEMA_MEMBERS Rowset	AdomdConnection.Cubes[].Dimensions[].Hierarchies[].Levels[].GetMembers()
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MemoryGrant	DISCOVER_MEMORYGRANT	
F:\Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MemoryUsage	DISCOVER_MEMORYUSAGE	

GUID (used in Schema Rowsets)	XMLA Name (used in Schema Rowsets)	ADOMD.NET Object Model
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningColumns	DMSHEMA_MINING_COLUMNS Rowset	AdomdConnection.MiningModels[].MiningModelColumns
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningFunctions	DMSHEMA_MINING_FUNCTIONS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningModelContent	DMSHEMA_MINING_MODEL_CONTENT Rowset	AdomdConnection.MiningModels[].MiningContentNodes
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningModelContentPmml	DMSHEMA_MINING_MODEL_CONTENT_P_MML Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningModels	DMSHEMA_MINING_MODELS Rowset	AdomdConnection.MiningModels
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningModelXml	DMSHEMA_MINING_MODEL_XML Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningServiceParameters	DMSHEMA_MINING_SERVICE_PARAMETERS Rowset	AdomdConnection.MiningServices[].MiningServiceParameters
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningServices	DMSHEMA_MINING_SERVICES Rowset	AdomdConnection.MiningServices
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningStructureColumns	DMSHEMA_MINING_STRUCTURE_COLUMNS Rowset	AdomdConnection.MiningStructures[].MiningStructureColumns
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.MiningStructures	DMSHEMA_MINING_STRUCTURES Rowset	AdomdConnection.MiningStructures
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.PartitionDimensionStat	DISCOVER_PARTITION_DIMENSION_STAT	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.PartitionStat	DISCOVER_PARTITION_STAT	

GUID (used in Schema Rowsets)	XMLA Name (used in Schema Rowsets)	ADOMD.NET Object Model
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.PerformanceCounters	DISCOVER_PERFORMANCE_COUNTERS	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.ProviderTypes	DBSCHEMA_PROVIDER_TYPES Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.SchemaRowsets	DISCOVER_SCHEMA_ROWSETS Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Sessions	DISCOVER_SESSIONS	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Sets	MDSHEMA_SETS Rowset	AdomdConnection.Cubes[].NamedSets
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Tables	DBSCHEMA_TABLES Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.TablesInfo	DBSCHEMA_TABLES_INFO	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.TraceColumns	DISCOVER_TRACE_COLUMNS	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.TraceDefinitionProviderInfo	DISCOVER_TRACE_DEFINITION_PROVIDERINFO	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.TraceEventCategories	DISCOVER_TRACE_EVENT_CATEGORIES	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Traces	DISCOVER_TRACES	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.Transactions	DISCOVER_TRANSACTIONS	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.XmlaProperties	DISCOVER_PROPERTIES Rowset	
F:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid.XmlMetadata	DISCOVER_XML_METADATA rowset	

See Also

[ADOMD.NET Client Programming](#)

[ADOMD.NET Programming](#)

[Schema Rowsets](#)

Working with the ADOMD.NET Object Model

ADOMD.NET provides an object model for viewing the cubes and subordinate objects contained by an analytical data source. However, not all metadata for a given analytical data source is available through the object model. The object model provides access to only the information that is most useful for a client application to display in order to allow a user to interactively construct commands. Because of the reduced complexity of the metadata to present, the ADOMD.NET object model is easier to use.

In the ADOMD.NET object model, the

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection object provides access to information on the online analytical processing (OLAP) cubes and mining models defined on an analytical data source, and related objects such as dimensions, named sets, and mining algorithms.

Retrieving OLAP Metadata

Each **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object has a collection of **T:Microsoft.AnalysisServices.AdomdClient.CubeDef** objects that represent the cubes available to the user or application. The **T:Microsoft.AnalysisServices.AdomdClient.CubeDef** object exposes information about the cube, as well as various objects related to the cube, such as dimensions, key performance indicators, measures, named sets, and so on.

Whenever possible, you should use the **T:Microsoft.AnalysisServices.AdomdClient.CubeDef** object to represent metadata in client applications designed to support multiple OLAP servers, or for general metadata display and access purposes.

Note

For provider specific metadata, or for detailed metadata display and access, use schema rowsets to retrieve metadata. For more information, see [Retrieving Metadata from an Analytical Data Source](#).

The following example uses the **T:Microsoft.AnalysisServices.AdomdClient.CubeDef** object to retrieve the visible cubes and their dimensions from the local server:

```
Adomd.NetClient#RetrieveCubesAndDimensions
```

Retrieving Data Mining Metadata

Each **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object has several collections that provide information about the data mining capabilities of the data source:

- The **T:Microsoft.AnalysisServices.AdomdClient.MiningModelCollection** contains a list of every mining model in the data source.

- The **T:Microsoft.AnalysisServices.AdomdClient.MiningServiceCollection** provides information about the available mining algorithms.
- The **T:Microsoft.AnalysisServices.AdomdClient.MiningStructureCollection** exposes information about the mining structures on the server.

To determine how to query against a mining model on the server, iterate through the **P:Microsoft.AnalysisServices.AdomdServer.MiningModel.Columns** collection. Each **T:Microsoft.AnalysisServices.AdomdClient.MiningModelColumn** object exposes the following characteristics:

- Whether the object is an input column
(**P:Microsoft.AnalysisServices.AdomdClient.MiningModelColumn.IsInput**).
- Whether the object is a prediction column
(**P:Microsoft.AnalysisServices.AdomdClient.MiningModelColumn.IsPredictable**).
- The values associated with a discrete column
(**P:Microsoft.AnalysisServices.AdomdClient.MiningModelColumn.Values**)
- The type of data in the column
(**P:Microsoft.AnalysisServices.AdomdClient.MiningModelColumn.Type**).

See Also

[Retrieving Metadata](#)

Working with Schema Rowsets in ADOMD.NET

When you need more metadata than is available in the ADOMD.NET object model, ADOMD.NET provides the capability to retrieve the full range of XML for Analysis (XMLA), OLE DB, OLE DB for OLAP, and OLE DB for Data Mining schema rowsets:

XML for Analysis metadata

The XML for Analysis schema rowsets provide a method for retrieving low-level information about the server. Information available includes the data sources available on the server, the keywords reserved by the provider, the literals supported by the provider, and more. You can even use an XML for Analysis schema rowset to discover all schema rowsets supported by the provider.

For more information: [Retrieving Metadata from an Analytical Data Source](#)

OLE DB metadata

The OLE DB schema rowsets provide an industry-standard method for retrieving information from a variety of providers.

For more information: [OLE DB Schema Rowsets](#)

OLAP metadata

Schema information provided for an analytical data source includes databases or catalogs available from the analytical data source, cubes and mining models in a database, roles that exist for cubes at the data source, and more.

For more information: [OLE DB for OLAP Schema Rowsets](#)

Data Mining metadata

In addition to OLAP metadata, data mining metadata can be retrieved using schema rowsets. The available rowsets expose information on the available data mining models in the database, the available mining algorithms, the parameters that the algorithm require, mining structures, and more.

For more information: [Data Mining Schema Rowsets](#)

For each of these various schema rowsets, you retrieve metadata from the rowset by passing either a GUID or XMLA name with the

Overload:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object.

Retrieving Metadata by Passing GUIDS

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid** class contains a list of fields that represent the schema rowsets most commonly supported by providers and analytical data sources. To retrieve both general and provider-specific metadata from a provider or analytical data source, you use the GUIDs contained within the

T:Microsoft.AnalysisServices.AdomdClient.AdomdSchemaGuid object with the either of the following methods:

- **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet(System.Guid,System.Object[])**
- **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet(System.Guid,System.Object[],System.Boolean)**



Note

The ADOMD.NET data provider exposes schema information through functionality made available by your specific provider and analytical data source. Each provider and data source may provide different metadata.

Retrieving Metadata by Passing XMLA Names

The following methods take as arguments the XMLA schema name that identifies which schema information to return, and an array of restrictions on those returned columns:

- **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet(System.String,Microsoft.AnalysisServices.AdomdClient.AdomdRestrictionCollection)**
- **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet(System.String,Microsoft.AnalysisServices.AdomdClient.AdomdRestrictionCollection,System.Boolean)**

- **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet(System.String,System.String,Microsoft.AnalysisServices.AdomdClient.AdomdRestrictionCollection)**
- **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.GetSchemaDataSet(System.String,System.String,Microsoft.AnalysisServices.AdomdClient.AdomdRestrictionCollection,System.Boolean)**

Each of these methods returns an instance of a **DataSet** object that is populated with the schema information. The **DataSet** object is from the **System.Data** namespace of the Microsoft .NET Framework Class Library.

Example

Description

In the following example, the `GetActions` function takes a connection, the cube name, a coordinate, and a coordinate type, retrieves an [MDSHEMA ACTIONS Rowset](#), and returns the actions available on the selected coordinate.

`Adomd.NetClient#GetActions`

See Also

[Retrieving Metadata](#)

Executing Commands Against an Analytical Data Source

After establishing a connection to an analytical data source, you can use an **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object to run commands against and return results from that data source. These commands can retrieve data by using Multidimensional Expressions (MDX), Data Mining Extensions (DMX), or even a limited syntax of SQL. Additionally, you can use Analysis Services Scripting Language (ASSL) commands to modify the underlying database.

Creating a Command

Before running a command, you must create it. You can create a command using one of two methods:

- The first method uses the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** constructor, which can take a command to run at the data source, and an **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object on which to run the command.
- The second method uses the **M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.CreateCommand** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object.

The text of the command to be run can be queried and modified using the **P:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.CommandText** property. The commands that you create do not have to return data after they run.

Running a Command

After you have created an **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object, there are several **M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.Execute** methods that your command can use to perform various actions. The following table lists some of these actions.

To	Use this method
Return results as a stream of data	M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteReader(System.Data.CommandBehavior) to return an T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader object
Return a T:Microsoft.AnalysisServices.AdomdClient.CellSet object	M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteCellSet
Run commands that do not return rows	M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteNonQuery
Return an XMLReader object that contains the data in an XML for Analysis (XMLA) compliant format	M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteXmlReader

Example of Running a Command

This example uses the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** to run an XMLA command that will process the **Adventure Works DW** cube on the local server, without returning data.

```
Adomd.NetClient#ExecuteXMLAProcessCommand
```

Retrieving Data from an Analytical Data Source

Once you make a connection and create the query, you can retrieve any data. In ADOMD.NET, you can retrieve data using three different objects

(**T:Microsoft.AnalysisServices.AdomdClient.CellSet**, **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader**, and **T:System.Xml.XmlReader**) by calling one of the **Execute** methods of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object.

Each of these three objects balances interactivity and overhead:

- *Interactivity* refers to the ease-of-use and amount of information available through the object model.

- *Overhead* refers to the amount of traffic that an object model generates over the network connection to the server, the amount of memory needed for the object model, and the speed with which the object model retrieves data.

To help you select the data retrieval object that best suits the needs of your application, the following table highlights the differences between interactivity and overhead for each object.

Object	Interactivity	Overhead	Retains dimensionality	Usage Information
T:Microsoft.AnalysisServices.AdomdClient.CellSet	Highest	Moderately high, which results in slowest retrieval of data	Yes	ADOMD.NET Client Programming
T:Microsoft.AnalysisServices.AdomdClient.AdomdDataAdapter	Moderate	Moderate	No	Populating a DataSet from a DataAdapter
T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader	Moderate	Moderate	No	Retrieving Data Using the AdomdDataReader
T:System.Xml.XmlReader	Lowest	Lowest, which results in fastest data retrieval	Yes	Retrieving Data Using the XmlReader

See Also

[ADOMD.NET Programming](#)

Retrieving Data Using the CellSet

When retrieving analytical data, the **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object provides the most interactivity and flexibility. The

T:Microsoft.AnalysisServices.AdomdClient.CellSet object is an in-memory cache of hierarchical data and metadata that retains the original dimensionality of the data. The **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object can also be traversed in either a connected or disconnected state. Because of this disconnected ability, the **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object can be used to view data and metadata in any order and provides the most comprehensive object model for data retrieval. This disconnected capability also causes the **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object to have the most overhead, and to be the slowest ADOMD.NET data retrieval object model to populate.

Retrieving Data in a Connected State

To use the **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object to retrieve data, you follow these steps:

1. **Create a new instance of the object.**

To create a new instance of the **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object, you call the **M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.Execute** or **M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteCellSet** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object.

2. **Identify metadata.**

Besides retrieving data, ADOMD.NET also retrieves metadata for the cellset. As soon as the command has run the query and returned a **T:Microsoft.AnalysisServices.AdomdClient.CellSet**, you can retrieve the metadata through various objects. This metadata is needed for client applications to display and interact with cellset data. For example, many client applications provide functionality for drilling down on, or hierarchically displaying the child positions of, a specified position in a cellset.

In ADOMD.NET, the **P:Microsoft.AnalysisServices.AdomdClient.CellSet.Axes** and **P:Microsoft.AnalysisServices.AdomdClient.CellSet.FilterAxis** properties of the **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object represent the metadata of the query and slicer axes, respectively, in the returned cellset. Both properties return references to **T:Microsoft.AnalysisServices.AdomdClient.Axis** objects, which in turn contain the positions represented on each axis.

Each **T:Microsoft.AnalysisServices.AdomdClient.Axis** object contains a collection of **T:Microsoft.AnalysisServices.AdomdClient.Position** objects that represent the set of tuples available for that axis. Each **T:Microsoft.AnalysisServices.AdomdClient.Position** object represents a single tuple that contains one or more members, represented by a collection of **T:Microsoft.AnalysisServices.AdomdClient.Member** objects.

3. **Retrieve data from the cellset collection.**

Besides retrieving metadata, ADOMD.NET also retrieves data for the cellset. As soon as the command has run the query and returned a **T:Microsoft.AnalysisServices.AdomdClient.CellSet**, you can retrieve the data by using the **P:Microsoft.AnalysisServices.AdomdClient.CellSet.Cells** collection of the **T:Microsoft.AnalysisServices.AdomdClient.CellSet**. This collection contains the values that are calculated for the intersection of all axes in the query. Therefore, there are

several indexers for accessing each intersection, or cell. For a list of indexers, see

Overload:Microsoft.AnalysisServices.AdomdClient.CellCollection.Item.

Example of Retrieving Data in a Connected State

The following example makes a connection to the local server, and then runs a command on the connection. The example parses the results by using the **CellSet** object model: the captions (metadata) for the columns are retrieved from the first axis, and the captions (metadata) for each row are retrieved from the second axis, and the intersecting data is retrieved by using the

P:Microsoft.AnalysisServices.AdomdClient.CellSet.Cells collection.

Adomd.NetClient#ReturnCommandUsingCellSet

Retrieving Data in a Disconnected State

By loading XML returned from a previous query, you can use the

T:Microsoft.AnalysisServices.AdomdClient.CellSet object to provide a comprehensive method of browsing analytical data without requiring an active connection.



Note

Not all properties of the objects that are available from the

T:Microsoft.AnalysisServices.AdomdClient.CellSet object are available while in a disconnected state. For more information, see

M:Microsoft.AnalysisServices.AdomdClient.CellSet.LoadXml(System.Xml.XmlReader).

Example of Retrieving Data in a Disconnected State

The following example is similar to the metadata and data example shown earlier in this topic.

However, the command in the following example runs with a call to

M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteXmlReader, and the result is returned as a **System.Xml.XmlReader**. The example then populates the

T:Microsoft.AnalysisServices.AdomdClient.CellSet object by using this **System.Xml.XmlReader**

with the **M:Microsoft.AnalysisServices.AdomdClient.CellSet.LoadXml(System.Xml.XmlReader)**

method. Although this example loads the **System.Xml.XmlReader** immediately, you could cache the XML that is contained by the reader to a hard disk or transport that data to a different application through any means before loading the data into a cellset.

Adomd.NetClient#DemonstrateDisconnectedCellset

See Also

[Retrieving Data Using the XmlReader](#)

[Retrieving Data Using the AdomdDataReader](#)

[Retrieving Data Using the XmlReader](#)

Retrieving Data Using the AdomdDataReader

When retrieving analytical data, the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object provides a good balance between overhead and interactivity. The

T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader object retrieves a read-only, forward-only, flattened stream of data from an analytical data source. This unbuffered stream of

data enables procedural logic to efficiently process results from an analytical data source sequentially. This makes the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** a good choice when retrieving large amounts of data for display purposes because the data is not cached in memory.

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** can also increase application performance by retrieving data as soon as it is available, instead of waiting for the complete results of the query to be returned. The

T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader also reduces system overhead because, by default, this reader stores only one row at a time in memory.

The tradeoff for optimized performance is that the

T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader object provides less information about retrieved data than other data retrieval methods. The

T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader object does not support a large object model for representing data or metadata, nor does this object model allow for more complex analytical features like cell writeback. However, the

T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader object does provide a set of strongly typed methods for retrieving cellset data and a method for retrieving cellset metadata in a tabular format. Additionally, **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** implements the **IDbDataReader** interface to support data binding and for retrieving data using the **SelectCommand** method, from the **System.Data** namespace of the Microsoft .NET Framework Class Library.

Retrieving Data from the AdomdDataReader

To use the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object to retrieve data, you follow these steps:

1. Create a new instance of the object.

To create a new instance of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** class, you call the **M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.Execute** or **M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteReader** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object.

2. Retrieve data.

As the command runs the query, ADOMD.NET returns the results in the **Resultset** format, a tabular format as described in the XML for Analysis specification, to flatten the data for the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object. A tabular format is unusual when querying analytical data considering the variable dimensionality in such data. ADOMD.NET stores these tabular results in the network buffer on the client until you request them by using one of the following methods:

- Call the **M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.Read** method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object.

The **M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.Read** method obtains a row from the query results. You can then pass the name, or the ordinal reference, of the

column to the **Overload:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.Item** property to access each column of the returned row. For example, the first column in the current row is named, ColumnName. Then, either `reader[0].ToString()` or `reader["ColumnName"].ToString()` will return the contents of the first column in the current row.

- Call one of the typed accessor methods.

The **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** provides a series of typed accessor methods—methods that let you access column values in their native data types. When you know the underlying data type of a column value, a typed accessor method reduces the amount of type conversion required when retrieving the column value, and thereby, provides the highest performance.

Some of the typed accessor methods that are available include

M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.GetDateTime(System.Int32),
M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.GetDouble(System.Int32),
and **M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.GetInt32(System.Int32)**.

For a complete list of typed accessor methods, see

Methods.T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.

3. Close the reader.

You should always call the

M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.Close method when you have finished using the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object. While an instance of an **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object is open, the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** is being used exclusively by that **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader**. You will not be able to run any commands on the instance of the

T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection, including creating another **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** or **System.Xml.XmlReader**, until you close the original **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader**.

Example of Retrieving Data from the AdomdDataReader

The following code example iterates through a

T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader object, and returns the first two values, as strings, from each row.

```
If Reader.HasRows Then
    Do While objReader.Read()
        Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", _
            objReader.GetString(0), objReader.GetString(1))
    Loop
Else
    Console.WriteLine("No rows returned.")
```

```

End If

objReader.Close()

if (objReader.HasRows)

    while (objReader.Read())

        Console.WriteLine("\t{0}\t{1}", _

            objReader.GetString(0), objReader.GetString(1));

    else

        Console.WriteLine("No rows returned.");

objReader.Close();

```

Retrieving Metadata from the AdomdDataReader

While an instance of an **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object is open, you can retrieve schema information, or metadata, about the current recordset using the **M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.GetSchemaTable** method.

M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.GetSchemaTable returns a **DataTable** object that is populated with the schema information for the current recordset. The **DataTable** will contain one row for each column of the recordset. Each column of the schema table row maps to a property of the column returned in the cellset, where **ColumnName** is the name of the property and the value of the column is the value of the property.

Example of Retrieving Metadata from the AdomdDataReader

The following code example writes out the schema information for an **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** object.

```

Dim schemaTable As DataTable = objReader.GetSchemaTable()

Dim objRow As DataRow

Dim objColumn As DataColumn

For Each objRow In schemaTable.Rows

    For Each objColumn In schemaTable.Columns

        Console.WriteLine(objColumn.ColumnName & " = " & objRow(objColumn).ToString())

    Next

    Console.WriteLine()

Next

```

```

DataTable schemaTable = objReader.GetSchemaTable();

foreach (DataRow objRow in schemaTable.Rows)
{
    foreach (DataColumn objColumn in schemaTable.Columns)
    {
        Console.WriteLine(objColumn.ColumnName + " = " + objRow[objColumn]);
    }
    Console.WriteLine();
}

```

Retrieving Multiple Result Sets

Data mining supports the concept of nested tables, which ADOMD.NET exposes as nested rowsets. To retrieve the nested rowset associated with each row, you call the **M:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader.GetDataReader(System.Int32)** method.

See Also

[Retrieving Data Using the XmlReader](#)

[Retrieving Data Using the CellSet](#)

[Retrieving Data Using the XmlReader](#)

Retrieving Data Using the XmlReader

The **XmlReader** class, part of the **System.Xml** namespace for the Microsoft .NET Framework Class Library, is similar to the **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader** class in that the **XmlReader** class also provides fast, non-cached, forward-only access to data. If there is no need for an in-memory, analytical view of the data using the

T:Microsoft.AnalysisServices.AdomdClient.CellSet object, the **XmlReader** object is perfect for retrieving XML data, especially for large quantities of data. Because **XmlReader** streams data, **XmlReader** does not have to retrieve and cache all the data before exposing the data to the caller, as would be the case if a **T:Microsoft.AnalysisServices.AdomdClient.CellSet** object were used to convert the XML for Analysis response into an analytical object model representation.

The **XmlReader** class provides direct access to the XML for Analysis response received by ADOMD.NET when the

M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteXmlReader method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object is called. Because the retrieved data is raw XML, you must parse the data and metadata manually. As soon as the data has been retrieved, the **XmlReader** object should be closed.

Retrieving Data and Metadata

To use the **XmlReader** class to retrieve data, you follow these steps:

1. Create a new instance of the object.

To create a new instance of the **XmlReader** class, you call the

M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.Execute or

M:Microsoft.AnalysisServices.AdomdClient.AdomdCommand.ExecuteXmlReader method of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdCommand** object.

2. Retrieve data.

After the command runs the query and returns an **XmlReader**, you must parse the data and metadata. The XML data and metadata is presented in the native format used by the XML for Analysis provider. For most XML for Analysis providers, the native format is the **MDDDataSet** format. The **MDDDataSet** format provides both data and metadata for cellsets in a well-structured format. For more information about the **MDDDataSet** format, see the XML for Analysis specification.

3. Close the reader.

You should always call the

M:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.Close method when you have finished using the **XmlReader** object. While an **XmlReader** is open, that **XmlReader** has exclusive use of the **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection** object that was used to run the command. You will not be able to run any commands using that **T:Microsoft.AnalysisServices.AdomdClient.AdomdConnection**, including creating another **XmlReader** or **T:Microsoft.AnalysisServices.AdomdClient.AdomdDataReader**, until you close the original **XmlReader**.

Example of Retrieving Data from the XmlReader

The following example runs a command and retrieves the data as an **XmlReader**, outputting the contents of the file to the console.

```
Adomd.NetClient#OutputDataWithXML
```

See Also

[Retrieving Data Using the AdomdDataReader](#)

[Retrieving Data Using the CellSet](#)

[Retrieving Data Using the AdomdDataReader](#)

ADOMD.NET Server Programming

The ADOMD.NET server components of ADOMD.NET reside within the **Microsoft.AnalysisServices.AdomdServer** namespace (in `msmgdsrv.dll`). You use these server components to create custom Multidimensional Expressions (MDX) functions and stored procedures that are run on an instance of Microsoft SQL Server Analysis Services. The server objects provide the capabilities for querying cubes and mining models, and for evaluating expressions in a given context. The benefits for creating custom functions and stored procedures include fast execution, centralized deployment, and improved maintainability. The topics in the following table will help you develop ADOMD.NET server applications.

Topic	Description
ADOMD.NET	Describes the uses for ADOMD.NET server objects.
ADOMD.NET Server Object Architecture	Describes the object architecture for ADOMD.NET server objects.
User Defined Functions and Stored Procedures	Walks you through the process of creating a user defined function or stored Procedure.

See Also

[ADOMD.NET Client Programming](#)

[ADOMD.NET](#)

ADOMD.NET Server Functionality

All ADOMD.NET server objects provide read-only access to the data and metadata on the server. To retrieve data and metadata, you use the ADOMD.NET server object model as the server object model does not support schema rowsets.

With ADOMD.NET server objects, you can create a user defined function (UDF) or a stored procedure for Microsoft SQL Server Analysis Services. These in-process methods are called through query statements created in languages such as Multidimensional Expressions (MDX), Data Mining Extensions (DMX), or SQL. These in-process methods also provide added functionality without the latencies associated with network communications.



Note

The **T:Microsoft.AnalysisServices.AdomdServer.AdomdCommand** object only supports DMX.

What is a UDF?

A *UDF* is a method that has the following characteristics:

- You can call the UDF in the context of a query.
- The UDF can take any number of parameters.
- The UDF can return various types of data.

The following example uses the fictional UDF, `FinalSalesNumber`:

```
SELECT SalesPerson.Name ON ROWS,
       FinalSalesNumber() ON COLUMNS
FROM SalesModel
```

What is a Stored Procedure?

A *stored procedure* is a method that has the following characteristics:

- You call a stored procedure on its own with the MDX [CALL](#) statement.
- A stored procedure can take any number of parameters.
- A stored procedure can return a dataset, an **IDataReader**, or an empty result.

The following example uses the fictional stored procedure, `FinalSalesNumbers`:

```
CALL FinalSalesNumbers ()
```

See Also

[ADOMD.NET Server Programming](#)

ADOMD.NET Server Object Architecture

The ADOMD.NET server objects are helper objects that can be used to create user defined functions (UDFs) or stored procedures in Microsoft SQL Server Analysis Services.



Note

To use the **Microsoft.AnalysisServices.AdomdServer** namespace (and these objects), a reference to the `msmgdsrv.dll` must be added to UDF project or stored procedure.



Interaction with the ADOMD.NET object hierarchy typically starts with one or more of the objects in the topmost layer, as described in the following table.

To	Use this object
Evaluate Multidimensional Expressions (MDX) expressions	T:Microsoft.AnalysisServices.AdomdServer.Expression The T:Microsoft.AnalysisServices.AdomdServer.Expression object provides a way to run an MDX expression and evaluate that expression under a specified tuple.
Provide support for executing MDX functions without constructing the full MDX statement	T:Microsoft.AnalysisServices.AdomdServer.MDX The T:Microsoft.AnalysisServices.AdomdServer.MDX object is convenient for calling predefined MDX functions without using the T:Microsoft.AnalysisServices.AdomdServer.Expression object. Additional functions for the T:Microsoft.AnalysisServices.AdomdServer.MDX object should be available in future releases.
Represent the current execution context for the UDF	T:Microsoft.AnalysisServices.AdomdServer.Context The T:Microsoft.AnalysisServices.AdomdServer.Context object exposes information such as the current cube or mining model and various metadata collections. One key use of the T:Microsoft.AnalysisServices.AdomdServer.Context object is the P:Microsoft.AnalysisServices.AdomdServer.Hierarchy.CurrentMember property of the T:Microsoft.AnalysisServices.AdomdServer.Hierarchy object. This key usage enables the author of the UDF or stored procedure to make decisions based on what member from a certain dimension the query is on.
Create sets and tuples	T:Microsoft.AnalysisServices.AdomdServer.SetBuilder , T:Microsoft.AnalysisServices.AdomdServer.TupleBuilder The T:Microsoft.AnalysisServices.AdomdServer.SetBuilder provides a way to create immutable sets, while the T:Microsoft.AnalysisServices.AdomdServer.TupleBuilder provides a way to create immutable tuples.
Support implicit conversion and casting among the six basic types of the MDX language	T:Microsoft.AnalysisServices.AdomdServer.MDXValue The T:Microsoft.AnalysisServices.AdomdServer.MDXValue object provides implicit conversion and casting among the following types: • T:Microsoft.AnalysisServices.AdomdServer.Hierarchy • T:Microsoft.AnalysisServices.AdomdServer.Level

To	Use this object
	• T:Microsoft.AnalysisServices.AdomdServer.Member • T:Microsoft.AnalysisServices.AdomdServer.Tuple • T:Microsoft.AnalysisServices.AdomdServer.Set • Scalar, or value types

See Also

[ADOMD.NET Server Programming](#)

User Defined Functions and Stored Procedures

With ADOMD.NET server objects, you can create user defined function (UDF) or stored procedures for Microsoft SQL Server Analysis Services that interact with metadata and data from the server. These in-process methods are called through Multidimensional Expressions (MDX) or Data Mining Extensions (DMX) statements to provide added functionality without the latencies associated with network communications.

UDF Examples

A UDF is a method that can be called in the context of an MDX or DMX statement, can take any number of parameters, and can return any type of data.

A UDF created using MDX is similar to one created for DMX. The main difference is that certain properties of the **T:Microsoft.AnalysisServices.AdomdServer.Context** object, such as the **P:Microsoft.AnalysisServices.AdomdServer.Context.CurrentCube** and **P:Microsoft.AnalysisServices.AdomdServer.Context.CurrentMiningModel** properties, are available only for one scripting language or the other.

The following examples show how to use a UDF to return a node description, filter tuples, and apply a filter to a tuple.

Returning a Node Description

The following example creates a UDF that returns the node description for a specified node. The UDF uses the current context in which it is being run, and uses a DMX FROM clause to retrieve the node from the current mining model.

```
public string GetNodeDescription(string nodeUniqueName)
{
    return Context.CurrentMiningModel.GetNodeFromUniqueName(nodeUniqueName).Description;
}
```

Once deployed, the previous UDF example can be called by the following DMX expression, which retrieves the most-likely prediction node. The description contains information that describes the conditions that make up the prediction node.

```
select Cluster(), SampleAssembly.GetNodeDescription( PredictNodeId(Cluster()) ) FROM
[Customer Clusters]
```

Returning Tuples

The following example takes a set and a return count, and randomly retrieves tuples from the set, returning a final subset:

```
public Set RandomSample(Set set, int returnCount)
{
    //Return the original set if there are fewer tuples
    //in the set than the number requested.
    if (set.Tuples.Count <= returnCount)
        return set;

    System.Random r = new System.Random();
    SetBuilder returnSet = new SetBuilder();

    //Retrieve random tuples until the return set is filled.
    int i = set.Tuples.Count;
    foreach (Tuple t in set.Tuples)
    {
        if (r.Next(i) < returnCount)
        {
            returnCount--;
            returnSet.Add(t);
        }
        i--;
        //Stop the loop if we have enough tuples.
        if (returnCount == 0)
            break;
    }

    return returnSet.ToSet();
}
```

The previous example is called in the following MDX example. In this MDX example, five random states or provinces are retrieved from the **Adventure Works** database.

```
SELECT SampleAssembly.RandomSample ([Geography].[State-Province].Members, 5) on
ROWS,
[Date].[Calendar].[Calendar Year] on COLUMNS
FROM [Adventure Works]
WHERE [Measures].[Reseller Freight Cost]
```

Applying a Filter to a Tuple

In the following example, a UDF is defined that takes a set, and applies a filter to each tuple in the set, using the Expression object. Any tuples that conform to the filter will be added to a set that is returned.

Adomd.NetServer#FilterSet

The previous example is called in the following MDX example, which filters the set to cities with names beginning with 'A'.

```
Select Measures.Members on Rows,
SampleAssembly.FilterSet([Customer].[Customer Geography].[City], "[Customer].[Customer
Geography].[City].CurrentMember.Name < 'B'") on Columns
From [Adventure Works]
```

Stored Procedure Example

In the following example, an MDX-based stored procedure uses AMO to create partitions, if needed, for Internet Sales.

Adomd.NetServer#CreateInternetSalesMeasureGroupPartitions

Redistributing ADOMD.NET

When you write applications that use ADOMD.NET, you must redistribute ADOMD.NET along with your application. To redistribute ADOMD.NET, include the ADOMD.NET setup program, SQLServer2005_ADOMD.msi, in your application's setup program. The ADOMD.NET setup program can be found on the Microsoft Web site. After you include the ADOMD.NET setup program, have your application's setup program launch the ADOMD.NET setup program and install ADOMD.NET.

The ADOMD.NET setup program installs the ADOMD.NET files in <system drive>:\Program Files\Microsoft.NET\ADOMD.NET\90.

For more information, see [Feature Pack for Microsoft SQL Server - November 2005](#).

See Also

[ADOMD.NET Concepts](#)

Developing with Analysis Management Objects (AMO)

Analysis Management Objects (AMO) is the complete library of programmatically accessed objects that enables an application to manage a running instance of Microsoft SQL Server Analysis Services. This section explains AMO concepts, focusing on major objects, how and when to use them, and the way they are interrelated. For more information about specific objects or classes, see the **N:Microsoft.AnalysisServices**.

See Also

[Decision Support Objects \(DSO\)](#)

[ASSL](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

AMO Concepts and Object Model

This topic provides a definition of Analysis Management Objects (AMO), how AMO is related to other tools and libraries provided in the architecture of Microsoft SQL Server Analysis Services, and a conceptual explanation of all major objects in AMO.

AMO is a complete collection of management classes for Analysis Services that can be used programmatically, under the namespace of **N:Microsoft.AnalysisServices**, in a managed environment. The classes are included in the AnalysisServices.dll file, which is usually found where the SQL Server setup installs the files, under the folder \100\SDK\Assemblies\. To use the AMO classes, include a reference to this assembly in your projects.

By using AMO you are able to create, modify, and delete objects such as cubes, dimensions, mining structures, and Analysis Services databases; over all these objects, actions can be performed from your application in the .NET Framework. You can also process and update the information stored in Analysis Services databases.

With AMO you cannot query your data. To query your data, use [ADOMD.NET](#).

This topic contains the following sections:

AMO in the Analysis Services Architecture

AMO Architecture

Using AMO

Automating Administrative Tasks with AMO

AMO in the Analysis Services Architecture

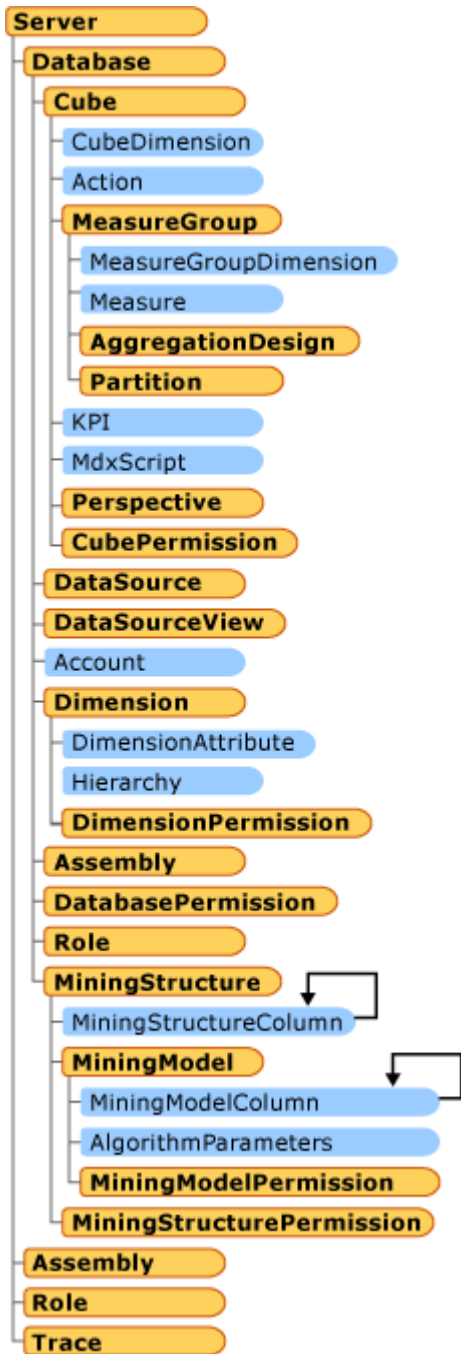
By design, AMO is only intended for object management and not for querying data. If the user needs to query Analysis Services data from a client application, the client application should use [ADOMD.NET](#).

AMO Architecture

AMO is a complete library of classes designed to manage an instance of Analysis Services from a client application in managed code under the .NET Framework version 2.0.

The AMO library of classes is designed as a hierarchy of classes, where certain classes must be instantiated before others in order to use them in your code. There are also auxiliary classes that can be instantiated at any time in your code, but you will probably have instantiated one or more of the hierarchy classes before using any one of the auxiliary classes.

The following illustration is a high-level view of the AMO hierarchy that includes major classes. The illustration shows the placement of the classes among their containers and their peers. A **T:Microsoft.AnalysisServices.Dimension** belongs to a **T:Microsoft.AnalysisServices.Database** and a **T:Microsoft.AnalysisServices.Server**, and can be created at the same time as a **T:Microsoft.AnalysisServices.DataSource** and **T:Microsoft.AnalysisServices.MiningStructure**. Certain peer classes must be instantiated before you can use others. For example, you have to create an instance of **T:Microsoft.AnalysisServices.DataSource** before adding a new **T:Microsoft.AnalysisServices.Dimension** or **T:Microsoft.AnalysisServices.MiningStructure**.



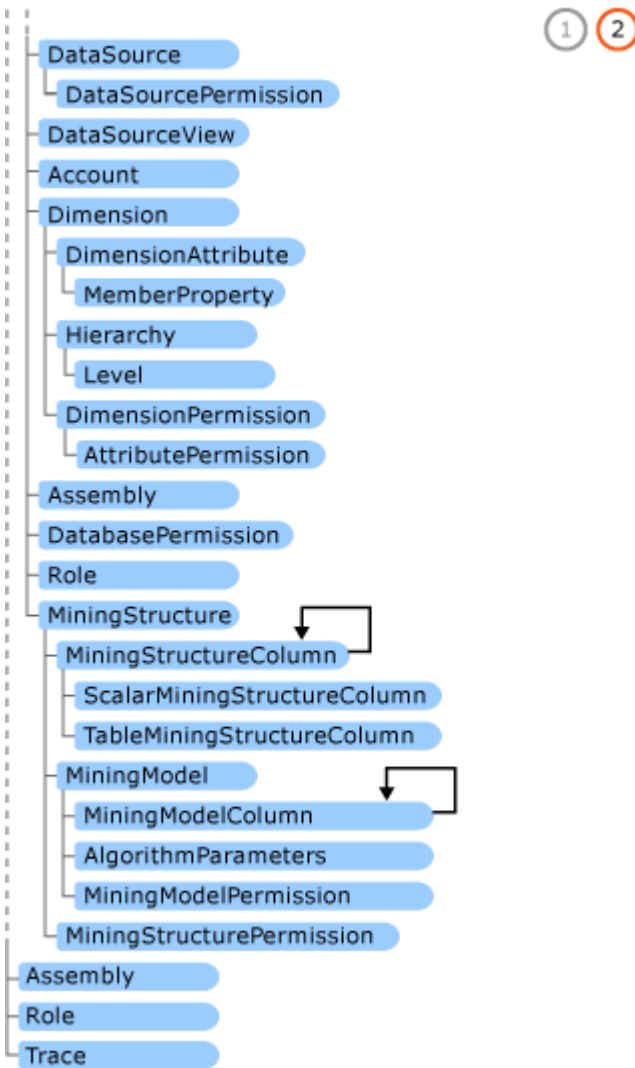
A *major object* is a class that represents a complete object as a whole entity and not as a part of another object. Major objects include **T:Microsoft.AnalysisServices.Server**, **T:Microsoft.AnalysisServices.Cube**, **T:Microsoft.AnalysisServices.Dimension**, and **T:Microsoft.AnalysisServices.MiningStructure**, because these are entities on their own. However, a **T:Microsoft.AnalysisServices.Level** is not a major object, because it is a constituent part of a

T:Microsoft.AnalysisServices.Dimension. Major objects can be created, deleted, modified, or processed independent of other objects. Minor objects are objects that can only be created as part of creating the parent major object. Minor objects are usually created upon a major object creation. Values for minor objects should be defined at creation time because there is no default creation for minor objects.

The following illustration shows the major objects that a **T:Microsoft.AnalysisServices.Server** object contains.



1 2



When programming with AMO, the association between classes and contained classes uses collection type attributes, for example **T:Microsoft.AnalysisServices.Server** and **T:Microsoft.AnalysisServices.Dimension**. To work with one instance of a contained class, you first acquire a reference to a collection object that holds or can hold the contained class. Next, you find the specific object that you are looking for in the collection, and then you can obtain a reference to the object to start working with it.

AMO Classes

AMO is a library of classes designed to manage an instance of Analysis Services from a client application. The AMO library can be thought of as logically-related groups of objects that are used to accomplish a specific task. AMO classes can be categorized in the following way:

Class Set	Purpose
AMO Fundamental Classes	Classes required in order to work with any other set of classes.
OLAP Classes	Classes that let you manage the OLAP objects in Analysis Services.
DataMining Classes	Classes that let you manage the data mining objects in Analysis Services.
Security Classes	Classes that let you control access to other objects and maintain security.
Other classes and methods	Classes and methods that help OLAP or data mining administrators to complete their daily tasks.

Using AMO

AMO is especially useful for automating repetitive tasks, for example creating new partitions in a measure group based on new data in the fact table, or re-training a mining model based on new data. These tasks that create new objects are usually performed on a monthly, weekly, or quarterly basis, and the new objects can easily be named, based in the new data, by the application.

Analysis Services administrators

Analysis Services administrators can use AMO to automate the processing of Analysis Services databases. For designing and deploying Analysis Services databases, you should use SQL Server Data Tools (SSDT).

Developers

Developers can use AMO to develop administrative interfaces for specified sets of users. These interfaces can restrict access to Analysis Services objects and limit users to certain tasks. For example, by using AMO you could create a Backup application that enables a user to see all database objects, select any one of the databases, and backup it to any one of a specified set of devices.

Developers can also embed Analysis Services logic in their applications. For this, developers can create cubes, dimensions, mining structures, and mining models based on user input or other factors.

OLAP advanced users

OLAP advanced users are usually data analysts or other experienced data users who have a strong programming background and who want to enhance their data analysis with a closer

usage of the data objects. For users who are required to work offline, AMO can be very useful to automate creating local cubes before going offline.

Data mining advanced users

For data mining advanced users, AMO is most useful if you have large sets of models that periodically have to be re-trained.

Automating Administrative Tasks with AMO

Most repetitive tasks are best designed, deployed, and maintained if they are developed by using Integration Services than if they are developed as an application in any language of your choice. However, for repetitive tasks that cannot be automated by using Integration Services, you can use AMO. AMO is also useful for when you want to develop a specialized application for business intelligence by using Analysis Services.

Automatic object management

With AMO is very easy to create, update or delete Analysis Services objects (for example **T:Microsoft.AnalysisServices.Database**, **T:Microsoft.AnalysisServices.Dimension**, **T:Microsoft.AnalysisServices.Cube**, mining **T:Microsoft.AnalysisServices.MiningStructure**, and **T:Microsoft.AnalysisServices.MiningModel**, or **T:Microsoft.AnalysisServices.Role**) based on user input or on new acquired data. AMO is ideal for setup applications that have to deploy a developed solution, from an independent software vendor to a final customer. The setup application can verify that an earlier version exists and can update the structure, remove no longer useful objects, and create new ones. If there is no earlier version then can create everything from scratch.

AMO can be powerful to create new partitions based on new data, and can remove old partitions that had gone beyond the scope of the project. For example, for a finance analysis solution that works with the last 36 months of data, as soon as a new month of data is received, the 37th old month could be removed. To optimize performance, new aggregations can be designed based on usage and applied to the last 12 months.

Automatic object processing

Object processing and updated availability can be achieved by using AMO to respond to certain events beyond the ordinary flow data and scheduled tasks that use Integration Services.

Automatic security management

Security management can be automated to include new users to roles and permissions, or to remove other users as soon as their time has expired. New interfaces can be created to simplify security management for security administrators. This can be simpler than using SQL Server Data Tools (SSDT).

Automatic Backup management

Automatic backup management can be done by using Integration Services tasks, or by creating specialized AMO applications that run automatically. By using AMO you can develop Backup interfaces for operators that help them in their daily jobs.

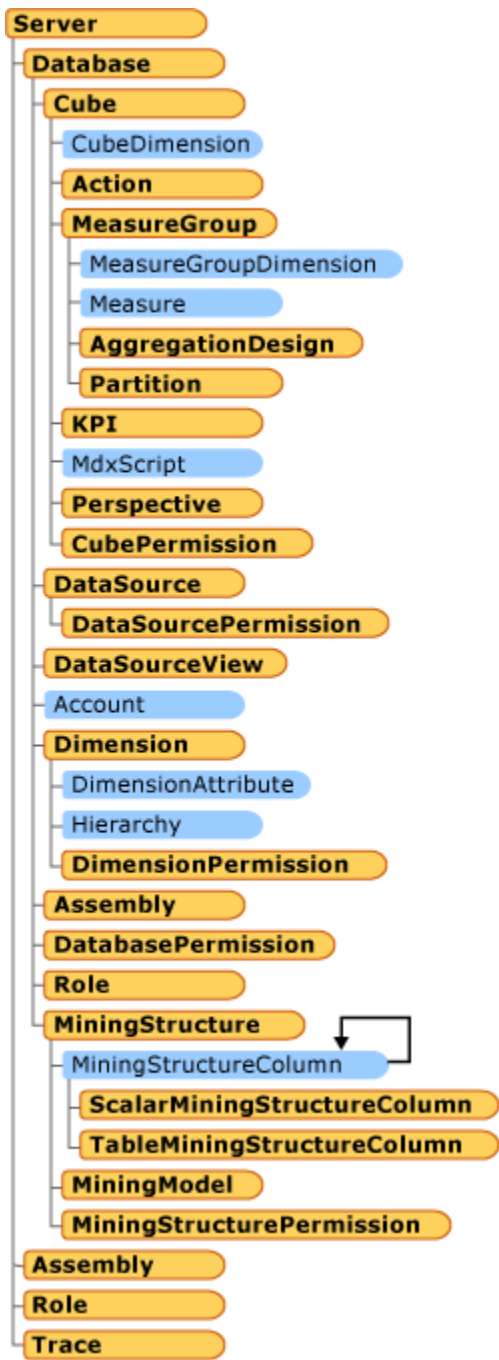
Tasks AMO is not intended for

AMO cannot be used to query the data. To query Analysis Services data, including cubes and mining models, use ADOMD.NET from a user application. For more information, see [ADOMD.NET](#).

Introducing AMO Classes

Analysis Management Objects (AMO) is a library of classes designed to manage an instance of Microsoft SQL Server Analysis Services from a client application. AMO classes are classes that you will use to administer Analysis Services objects such as databases, dimensions, cubes, mining structures and models, roles and permissions, exceptions, and others

The following illustration shows the relationship of the classes that are explained in this topic.



The AMO library can be thought of as logically-related groups of objects that are used to accomplish a specific task. AMO classes can be categorized in the following way. This section includes the following topics:

Topic	Description
Programming Administrative Tasks with AMO (deleted)	Describes classes that are required in order to work with any other set of classes.
OLAP Classes	Describes classes that let you manage the OLAP objects in Analysis Services.
DataMining Classes	Describes classes that let you manage the data mining objects in Analysis Services.
Security Classes	Describes classes that let you control access to other objects and maintain security.
Other classes and methods	Describes classes and methods that help OLAP or Data Mining administrators to complete their daily tasks.

See Also

N:Microsoft.AnalysisServices

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

[Analysis Management Objects \(AMO\)](#)

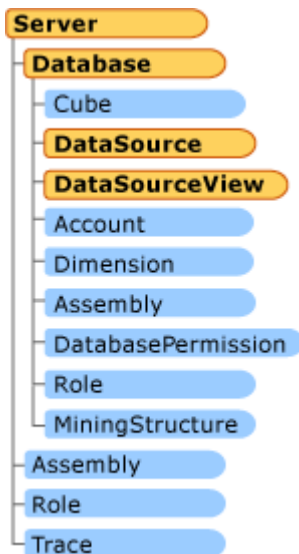
AMO Fundamental Classes

Fundamental classes are the starting point for working with Analysis Management Objects (AMO). Through these classes you establish your environment for the rest of the objects that will be used in your application. Fundamental classes include the following objects:

T:Microsoft.AnalysisServices.Server, **T:Microsoft.AnalysisServices.Database**,

T:Microsoft.AnalysisServices.DataSource, and **T:Microsoft.AnalysisServices.DataSourceView**.

The following illustration shows the relationship of the classes that are explained in this topic.



This topic contains the following sections:

- Server Objects
- Database Objects
- DataSource and DataSourceView Objects

Server Objects

Additionally, you will have access to the following methods:

- Connection management: Connect, Disconnect, Reconnect, and GetConnectionState.
- Transaction management: BeginTransaction, CommitTransaction, and RollbackTransaction.
- Backup and Restore.
- DDL execution: Execute, CancelCommand, SendXmlaRequest, StartXmlaRequest.
- Metadata management: UpdateObjects and Validate.

To connect to a server, you need a standard connection string, as used in ADOMD.NET and OLEDB. For more information, see

P:System.Configuration.ConnectionStringSettings.ConnectionString. The name of the server can be specified as a connection string without having to use a connection string format.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Server in the **N:Microsoft.AnalysisServices**.

Database Objects

To work with a **T:Microsoft.AnalysisServices.Database** object in your application, you must get an instance of the database from the parent server databases collection. To create a database, you add a **T:Microsoft.AnalysisServices.Database** object to a server databases collection and update the new instance to the server. To delete a database, you drop the

T:Microsoft.AnalysisServices.Database object by using its own Drop method.

Databases can be backed up by using the BackUp method (from the **T:Microsoft.AnalysisServices.Database** object or from the **T:Microsoft.AnalysisServices.Server** object), but can only be restored from the **T:Microsoft.AnalysisServices.Server** object with the Restore method.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.Database** in the **N:Microsoft.AnalysisServices**.

DataSource and DataSourceView Objects

Data sources are managed by using the **T:Microsoft.AnalysisServices.DataSourceCollection** from the database class. An instance of **T:Microsoft.AnalysisServices.DataSource** can be created by using the Add method from a **T:Microsoft.AnalysisServices.DataSourceCollection** object. An instance of **T:Microsoft.AnalysisServices.DataSource** can be deleted by using the Remove method from a **T:Microsoft.AnalysisServices.DataSourceCollection** object.

T:Microsoft.AnalysisServices.DataSourceView objects are managed from the **T:Microsoft.AnalysisServices.DataSourceViewCollection** object in the database class.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.DataSource** and **T:Microsoft.AnalysisServices.DataSourceView** in the **N:Microsoft.AnalysisServices**.

See Also

N:Microsoft.AnalysisServices

[Introducing AMO Classes](#)

[Programming AMO Fundamental objects](#)

AMO OLAP Classes

Analysis Management Objects (AMO) OLAP classes help you create, modify, delete, and process cubes, dimensions, and related objects such as Key Performance Indicators (KPIs), actions, and proactive caching.

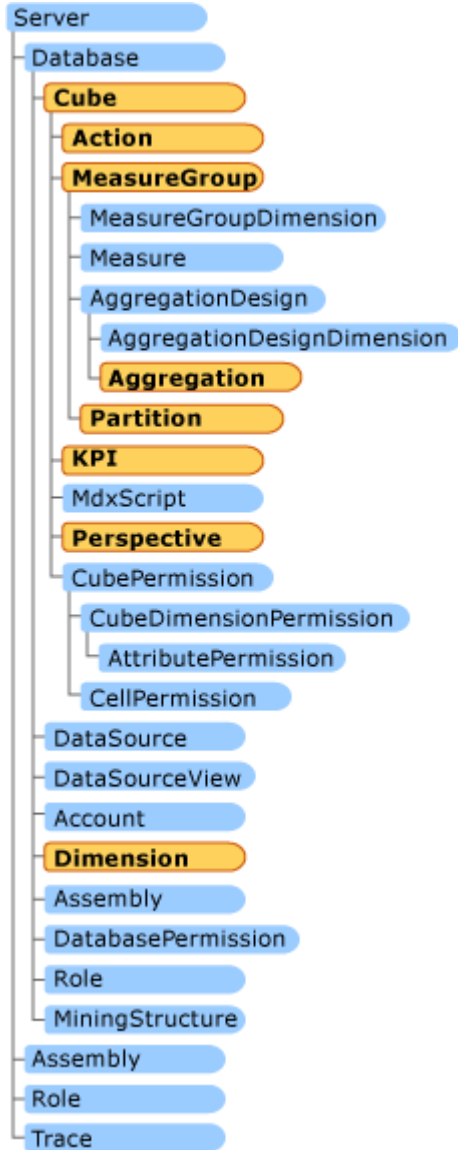
For more information about setting up the AMO programming environment, how to establish a connection with a server, accessing a database or defining data sources and data source views, see [Database Objects \(Analysis Services - Multidimensional Data\)](#).

This topic contains the following sections:

- Dimension Objects
- Cube Objects
- MeasureGroup Objects
- Partition Objects
- AggregationDesign Objects
- Aggregation Objects
- Action Objects
- KPI Objects

- Perspective Objects
- Translation Objects
- ProactiveCaching Objects

The following illustration shows the relationship of the classes that are explained in this topic.



Basic Classes

Dimension Objects

A dimension is created by adding it to the dimensions collection of the parent database, and by updating the **T:Microsoft.AnalysisServices.Dimension** object to the server by using the Update method.

To remove a dimension, it has to be dropped by using the Drop method of the **T:Microsoft.AnalysisServices.Dimension**. Removing a **T:Microsoft.AnalysisServices.Dimension** from the dimensions collection of the database by using the Remove method does not delete it on the server, just in the AMO object model.

A **T:Microsoft.AnalysisServices.Dimension** object can be processed after it has been created. The **T:Microsoft.AnalysisServices.Dimension** can be processed using its own process method, or it can be processed with the parent object's process method when the parent object is processed.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Dimension in the **N:Microsoft.AnalysisServices**.

Cube Objects

A cube is created by adding it to the cubes collection of the database, then updating the **T:Microsoft.AnalysisServices.Cube** object to the server by using the Update method. The Update method of the cube can include the parameter UpdateOptions.ExpandFull, which ensures that all objects in the cube that were modified will be updated to the server in this update action.

To remove a cube, it has to be dropped by using the Drop method of the

T:Microsoft.AnalysisServices.Cube. Removing a cube from the collection does not affect the server.

A **T:Microsoft.AnalysisServices.Cube** object can be processed after it has been created. The **T:Microsoft.AnalysisServices.Cube** can be processed using its own process method, or it can be processed when a parent object processes itself with its own Process method.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Cube in the **N:Microsoft.AnalysisServices**.

MeasureGroup Objects

A measure group is created by adding it to the measure group collection of the cube, then updating the **T:Microsoft.AnalysisServices.MeasureGroup** object to the server by using its own Update method. A **T:Microsoft.AnalysisServices.MeasureGroup** object is removed using its own Drop method.

A **T:Microsoft.AnalysisServices.MeasureGroup** object can be processed after it has been created. The **T:Microsoft.AnalysisServices.MeasureGroup** can be processed by using its own Process method, or it can be processed when a parent object processes itself with its own Process method.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.MeasureGroup in the **N:Microsoft.AnalysisServices**.

Partition Objects

A **T:Microsoft.AnalysisServices.Partition** object is created by adding it to the partitions collection of the parent measure group, then updating the **T:Microsoft.AnalysisServices.Partition** object on the server by using the Update method. A **T:Microsoft.AnalysisServices.Partition** object is removed by using the Drop method.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Partition in the **N:Microsoft.AnalysisServices**.

AggregationDesign Objects

Aggregation designs are constructed using the `AggregationDesign` method from an **T:Microsoft.AnalysisServices.AggregationDesign** object.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.AggregationDesign in the **N:Microsoft.AnalysisServices**.

Aggregation Objects

An **T:Microsoft.AnalysisServices.Aggregation** object is created by adding it to the aggregation designs collection of the parent measure group, then updating the parent measure group object on the server by using the `Update` method. An aggregation is removed from the **T:Microsoft.AnalysisServices.AggregationCollection** by using the `Remove` method or the `RemoveAt` method.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Aggregation in the **N:Microsoft.AnalysisServices**.

Advanced Classes

Advanced classes provide OLAP functionality beyond building and browsing a cube. The following are some of the advanced classes and the benefits they provide:

- Action classes are used to create an active response when browsing certain areas of the cube.
- Key Performance Indicators (KPIs) enable comparison analysis between values of data.
- Perspectives provide selected views of a single cube, so that users can focus on what is important to them.
- Translations allow the cube to be customized to the user locale.
- Proactive caching classes can provide a balance between the enhanced performance of MOLAP storage and the immediacy of ROLAP storage, and provide scheduled partition processing.

AMO is used to set the definitions for this enhanced behavior, but the actual experience is defined by the browsing client that implements all of these enhancements.

Action Objects

An **T:Microsoft.AnalysisServices.Action** object is created by adding it to the actions collection of the cube, then updating the **T:Microsoft.AnalysisServices.Cube** object to the server by using the `Update` method. The update method of the cube can include the parameter `UpdateOptions.ExpandFull`, which ensures that all objects in the cube that were modified will be updated to the server with this update action.

To remove an **T:Microsoft.AnalysisServices.Action** object, it must be removed from the collection and the parent cube must be updated.

A cube must be updated and processed before the action can be used from the client.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Action in the **N:Microsoft.AnalysisServices**.

Kpi Objects

A **T:Microsoft.AnalysisServices.Kpi** object is created by adding it to the KPI collection of the cube, then updating the **T:Microsoft.AnalysisServices.Cube** object to the server by using the Update method. The Update method of the cube can include the parameter `UpdateOptions.ExpandFull`, which ensures that all objects in the cube that were modified will be updated to the server with this update action.

To remove a **T:Microsoft.AnalysisServices.Kpi** object, it must be removed from the collection, then and the parent cube must be updated.

A cube must be updated and processed before the KPI can be used.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.Kpi** in the **N:Microsoft.AnalysisServices**.

Perspective Objects

A **T:Microsoft.AnalysisServices.Perspective** object is created by adding it to the perspective collection of the cube, then updating the **T:Microsoft.AnalysisServices.Cube** object to the server by using the Update method. The Update method of the cube can include the parameter `UpdateOptions.ExpandFull`, which ensures that all objects in the cube that were modified will be updated to the server with this update action.

To remove a **T:Microsoft.AnalysisServices.Perspective** object, it must be removed from the collection, then the parent cube must be updated.

A cube has to be updated and processed before the perspective can be used.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.Perspective** in the **N:Microsoft.AnalysisServices**.

Translation Objects

A **T:Microsoft.AnalysisServices.Translation** object is created by adding it to the translation collection of the desired object, then updating the closest major parent object to the server by using the Update method. The Update method of the closest parent object can include the parameter `UpdateOptions.ExpandFull`, which ensures that all children objects that were modified will be updated to the server with this update action.

To remove a **T:Microsoft.AnalysisServices.Translation** object, it must be removed from the collection, then the closest parent object must be updated.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.Translation** in the **N:Microsoft.AnalysisServices**.

ProactiveCaching Objects

A **T:Microsoft.AnalysisServices.ProactiveCaching** object is created by adding it to the proactive caching object collection of the dimension or partition, then updating the dimension or partition object to the server by using the Update method.

To remove a **T:Microsoft.AnalysisServices.ProactiveCaching** object, it must be removed from the collection, then the parent object must be updated.

A dimension or partition must be updated and processed before proactive caching is enabled and ready to be used.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.ProactiveCaching** in the **N:Microsoft.AnalysisServices**.

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[Programming AMO OLAP basic objects](#)

[Programming AMO OLAP advanced objects](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

AMO Data Mining Classes

Data mining classes help you create, modify, delete, and process data mining objects. Working with data mining objects includes creating data mining structures, creating data mining models, and processing the models.

For more information about how to set up the environment, and about

T:Microsoft.AnalysisServices.Server, **T:Microsoft.AnalysisServices.Database**,

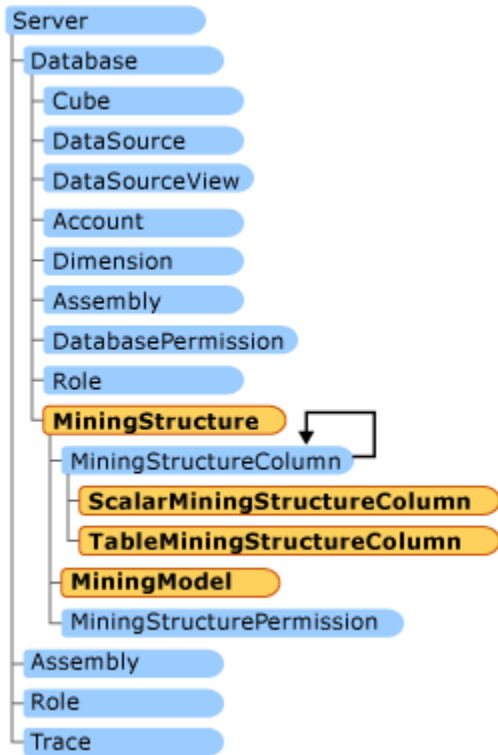
T:Microsoft.AnalysisServices.DataSource, and **T:Microsoft.AnalysisServices.DataSourceView** objects, see [Database Objects \(Analysis Services - Multidimensional Data\)](#).

Defining objects in Analysis Management Objects (AMO) requires setting a number of properties on each object to set up the correct context. Complex objects, such as OLAP and data mining objects, require lengthy and detailed coding.

This topic contains the following sections:

- MiningStructure Objects
- MiningModel Objects

The following illustration shows the relationship of the classes that are explained in this topic.



MiningStructure Objects

A mining structure is the container for mining models. The structure defines all possible columns that the mining models may use. Each mining model defines its own columns from the set of defined columns in the structure.

A simple **T:Microsoft.AnalysisServices.MiningStructure** object is composed of: basic information, a data source view, one or more **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn**, zero or more **T:Microsoft.AnalysisServices.TableMiningStructureColumn**, and a **T:Microsoft.AnalysisServices.MiningModelCollection**.

Basic information includes the name and ID (internal identifier) of the **T:Microsoft.AnalysisServices.MiningStructure** object.

The **T:Microsoft.AnalysisServices.DataSourceView** object holds the underlying data model for the mining structure.

T:Microsoft.AnalysisServices.ScalarMiningStructureColumn are columns or attributes that have single values.

T:Microsoft.AnalysisServices.TableMiningStructureColumn are columns or attributes that have multiple values for each case.

T:Microsoft.AnalysisServices.MiningModelCollection contains all mining models built on the same data.

A **T:Microsoft.AnalysisServices.MiningStructure** object is created by adding it to the **T:Microsoft.AnalysisServices.MiningStructureCollection** of the database and updating the **T:Microsoft.AnalysisServices.MiningStructure** object to the server, by using the Update method. To remove a **T:Microsoft.AnalysisServices.MiningStructure** object, it must be dropped by using the Drop method of the **T:Microsoft.AnalysisServices.MiningStructure** object. Removing a **T:Microsoft.AnalysisServices.MiningStructure** object from the collection does not affect the server.

The **T:Microsoft.AnalysisServices.MiningStructure** can be processed using its own process method, or it can be processed when a parent object processes itself with its own process method.

Columns

Columns hold the data for the model and can be of different types depending on the usage: Key, Input, Predictable, or InputPredictable. Predictable columns are the target of building the mining model.

Single-value columns are known as **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn** in AMO. Multiple-value columns are known as

T:Microsoft.AnalysisServices.TableMiningStructureColumn.

ScalarMiningStructureColumn

A simple **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn** object is composed of basic information, Type, Content, and data binding.

Basic information includes the name and ID (internal identifier) of the

T:Microsoft.AnalysisServices.ScalarMiningStructureColumn.

Type is the data type of the value: LONG, BOOLEAN, TEXT, DOUBLE, DATE.

Content tells the engine how the column can be modeled. Values can be: Discrete, Continuous, Discretized, Ordered, Cyclical, Probability, Variance, StdDev, ProbabilityVariance, ProbabilityStdDev, Support, Key.

Data binding is linking the data mining column with the underlying data model by using a data source view element.

A **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn** is created by adding it to the parent **T:Microsoft.AnalysisServices.MiningStructureCollection**, and updating the parent **T:Microsoft.AnalysisServices.MiningStructure** object to the server by using the Update method.

To remove a **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn**, it must be removed from the collection of the parent **T:Microsoft.AnalysisServices.MiningStructure**, and the parent **T:Microsoft.AnalysisServices.MiningStructure** object must be updated to the server by using the Update method.

TableMiningStructureColumn

A simple **T:Microsoft.AnalysisServices.TableMiningStructureColumn** object is composed of basic information and scalar columns.

Basic information includes the name and ID (internal identifier) of the

T:Microsoft.AnalysisServices.TableMiningStructureColumn.

Scalar columns are **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn**.

A **T:Microsoft.AnalysisServices.TableMiningStructureColumn** is created by adding it to the parent **T:Microsoft.AnalysisServices.MiningStructure** collection, and updating the parent

T:Microsoft.AnalysisServices.TableMiningStructureColumn object to the server by using the Update method.

To remove a **T:Microsoft.AnalysisServices.ScalarMiningStructureColumn**, it has to be removed from the collection of the parent **T:Microsoft.AnalysisServices.MiningStructure**, and the parent **T:Microsoft.AnalysisServices.MiningStructure** object must be updated to the server by using the Update method.

MiningModel Objects

A **T:Microsoft.AnalysisServices.MiningModel** is the object that allows you to choose which columns from the structure to use, an algorithm to use, and optionally specific parameters to tune the model. For example, you might want to define several mining models in the same mining structure that use the same algorithms, but to ignore some columns from the mining structure in one model, use them as inputs in another model, and use them as input and predict in a third model. This can be useful if in one mining model you want to treat a column as continuous, but in other model you want to treat the column as discretized.

A simple **T:Microsoft.AnalysisServices.MiningModel** object is composed of: basic information, algorithm definition, and columns.

Basic information includes the name and ID (internal identifier) of the mining model.

An algorithm definition refers to any one of the standard algorithms provided in Analysis Services, or any custom algorithms enabled on the server.

Columns are a collection of the columns that are used by the algorithm and their usage definition.

A **T:Microsoft.AnalysisServices.MiningModel** is created by adding it to the **T:Microsoft.AnalysisServices.MiningModelCollection** of the database and updating the **T:Microsoft.AnalysisServices.MiningModel** object to the server by using the Update method.

To remove a **T:Microsoft.AnalysisServices.MiningModel**, it has to be dropped by using the Drop method of the **T:Microsoft.AnalysisServices.MiningModel**. Removing a

T:Microsoft.AnalysisServices.MiningModel from the collection does not affect the server.

After it is created, a **T:Microsoft.AnalysisServices.MiningModel** can be processed by using its own process method, or it can be processed when a parent object processes itself with its own process method.

See Also

[AMO Fundamental Classes](#)

[Programming AMO DataMining objects](#)

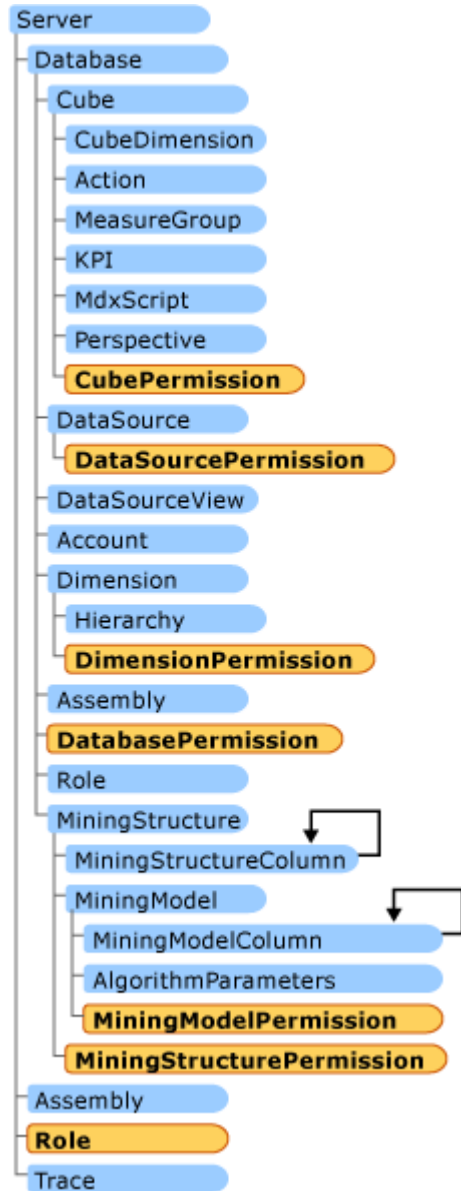
N:Microsoft.AnalysisServices

AMO Security Classes

This topic contains the following sections:

- Role and RoleMember Objects
- Permission Objects

The following illustration shows the relationship of the classes that are explained in this topic.



Role and RoleMember Objects

A **T:Microsoft.AnalysisServices.Role** object is created by adding it to the roles collection of the database, and updating the **T:Microsoft.AnalysisServices.Role** object to the server by using the Update method. A **T:Microsoft.AnalysisServices.Role** object has to be updated before it can be used.

To remove a **T:Microsoft.AnalysisServices.Role** object, it has to be dropped by using the Drop method of the **T:Microsoft.AnalysisServices.Role** object. The Remove method, from the roles collection, only prevents you from seeing the role in your application, but it does not remove the role from the server. A **T:Microsoft.AnalysisServices.Role** object cannot be dropped if there are any permissions associated with it.

A **T:Microsoft.AnalysisServices.RoleMember** object is created by adding a user to the members collection of the role and updating the **T:Microsoft.AnalysisServices.Role** object to the server by using the Update method. Only Server Administrators or Database Administrators are permitted to create roles. A **T:Microsoft.AnalysisServices.Role** object has to be updated to the server before any of its members is allowed to use any the objects to which the user has been granted permission.

To remove a **T:Microsoft.AnalysisServices.RoleMember** object, it has to be removed from the collection by using the Remove method of the collection, and then updating the role by using the Update method.

For more information about methods and properties available for these objects, see **T:Microsoft.AnalysisServices.Role** and **T:Microsoft.AnalysisServices.RoleMember** in the **N:Microsoft.AnalysisServices**.

Permission Objects

A **T:Microsoft.AnalysisServices.Permission** object is created by adding it to the permissions collection of the object and updating the **T:Microsoft.AnalysisServices.Permission** object to the server by using the Update method.

To remove a **T:Microsoft.AnalysisServices.Permission** object, it has to be dropped by using the Drop method of the object. The remove method, from the permissions collection, only prevents you from seeing the permission in your application, but it does not remove the **T:Microsoft.AnalysisServices.Permission** object from the server. A role cannot be deleted if there is any permission associated with it.

For more information about methods and properties available, see **T:Microsoft.AnalysisServices.Permission** in **N:Microsoft.AnalysisServices**.

See Also

[Database Objects \(Analysis Services - Multidimensional Data\)](#)

[Permissions and Access Rights \(SSAS\)](#)

[Security and Protection \(Analysis Services - Multidimensional Data\)](#)

N:Microsoft.AnalysisServices

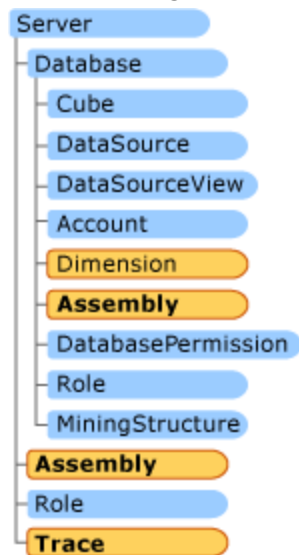
AMO Other Classes and Methods

This section contains common classes that are not specific to OLAP or data mining, and that are helpful when administering or managing objects in Microsoft SQL Server Analysis Services. These classes cover features such as stored procedures, tracing, exceptions, and backup and restore.

This topic contains the following sections:

- Assembly Objects
- Backup and Restore Methods
- Trace Objects
- CaptureLog Class and CaptureXML Attribute
- AMOException Exception Class

The following illustration shows the relationship of the classes that are explained in this topic.



Assembly Objects

An **T:Microsoft.AnalysisServices.Assembly** object is created by adding it to the assemblies collection of the server, and then updating the **T:Microsoft.AnalysisServices.Assembly** object to the server, by using the Update method.

To remove an **T:Microsoft.AnalysisServices.Assembly** object, it has to be dropped by using the Drop method of the **T:Microsoft.AnalysisServices.Assembly** object. Removing an **T:Microsoft.AnalysisServices.Assembly** object from the assemblies collection of the database

does not drop the assembly, it only prevents you from seeing it in your application until the next time that you run your application.

For more information about methods and properties available, see

T:Microsoft.AnalysisServices.Assembly in **N:Microsoft.AnalysisServices** .

noteDXDOC112778PADS **Security Note**

COM assemblies might pose a security risk. Due to this risk and other considerations, COM assemblies were deprecated in SQL Server 2008 Analysis Services (SSAS). COM assemblies might not be supported in future releases.

Backup and Restore Methods

Backup and Restore are methods that can be used to create copies of an Analysis Services database and recover the database by using the copy. The Backup method belongs to the **T:Microsoft.AnalysisServices.Database** object, and the Restore method belongs to the **T:Microsoft.AnalysisServices.Server** object.

Only server and database administrators are permitted to perform a backup of a database. Only server administrators can restore a database onto a different server than it was backed up from. Database administrators can restore a database by overwriting the existing database only if they own the database that is to be overwritten. After a restore, the database administrator may lose access to the restored database if the database is restored with its original security definitions.

Database backup files must have .abf extensions.

Backup Method

To backup a database, use the Backup method of the database object with the name of the backup file as a parameter.

Default values:

AllowOverwrite=**false**

BackupRemotePartitions=**false**

Security=**CopyAll**

ApplyCompression=**true**

Restore Method

To restore a database to a server, use the Restore method of the server with the backup file as a parameter.

Default values:

AllowOverwrite=**false**

DataSourceType=**Remote**

Security=**CopyAll**

Restrictions

1. A local partition cannot be restored as a remote partition.
2. A remote partition cannot be restored as a local partition, but a remote partition be restored on a different server than it was backed up from.

Common Parameters and Properties for Backup and Restore Methods

- **File** is the name of the file to backup (UNC name) into/from.
- **Location** specifies server-specific backup information, such as **BackupFile**. This allows you to specify a separate backup file for a remote database.
- **DatasourceID** specifies the ID of the subordinate database in a remote server.
- **ConnectionString** allows you to adjust the remote datasource in case the remote server has changed. DatasourceID must always be specified when ConnectionString is present.
- **Folder** allows remapping of the folders for partitions on the local hard drive
- **Original** is the original folder for local partitions.
- **New** is the new location for local partitions that used to reside in the corresponding 'Original' old folder.
- **Password**, if non-blank, specifies that the server will encrypt the backup file.

Trace Objects

Trace is a framework used for monitoring, replaying, and managing an instance of Analysis Services. A client application, like SQL Server Profiler, subscribes to a trace and the server sends back trace events as specified in the trace definition.

Each event is described by an event class. The event class describes the type of event generated. Within an event class, event subclasses describe a finer level of categorization. Each event is described by a number of columns. The columns that describe a trace event are consistent for all events and conform to the SQL trace structure. Information recorded in each column may differ depending on the event class; that is, a predefined set of columns is defined for each trace, but the meaning of the column may differ depending on the event class. For example, the TextData column is used to record the original ASSL for all statement events.

A trace definition can include one or more event classes to be traced concurrently. For each event class, one or more data columns can be added to the trace definition, but not all trace columns must be used. The database administrator can decide which of the available columns to include in a trace. Further, event classes can be selectively traced based on filter criteria on any column in the trace.

Traces can be started and deleted. Multiple traces can be run at any one time. Trace events can be captured live or directed to a file for later analysis or replay. SQL Server Profiler is the tool used to analyze and replay Analysis Services trace events. Multiple connections are allowed to receive events from the same trace.

Traces can be divided in two groups: server traces and session traces. Server traces will inform of all events in the server; session traces will inform only events in the current session.

Traces, from the traces collection of the server, are defined the following way:

1. Create a **T:Microsoft.AnalysisServices.Trace** object and populate its basic data, including trace ID, name, log file name, append|overwrite, and others.
2. Add Events to be monitored to the Events collection of the trace object. For each event, data columns are added.

3. Set Filters to exclude unnecessary rows of data by adding them to the filters collection.
4. Start the trace; creating the trace does not start collecting data.
5. Stop the trace.
6. Review the trace file with SQL Server Profiler.

Traces, from the session object, are obtained in the following manner:

1. Define functions to handle the trace events generated in your application by SessionTrace. Possible events are OnEvent and Stopped.
2. Add your defined functions to the event handler.
3. Start the session trace.
4. Do your process and let your function handlers capture the events.
5. Stop the session trace.
6. Continue with your application.

CaptureLog Class and CaptureXML Attribute

All actions to be executed by AMO are sent to the server as XMLA messages. AMO provides the means to capture all these messages without the SOAP headers. For more information, see [Database Objects \(Analysis Services - Multidimensional Data\)](#). CaptureLog is the mechanism in AMO for scripting out objects and operations; objects and operations will be scripted in XMLA.

To start capturing the XML, the CaptureXML server object property needs to be set to **true**. Then all actions that are to be sent to the server will start being captured in the CaptureLog class, without the actions being sent to the server. CaptureLog is considered a class because it has a method, Clear, which is used to clear the capture log.

To read the log, you get the strings collection and start iterating over the strings. Also, you can concatenate all logs into a string by using the server object method ConcatenateCaptureLog. ConcatenateCaptureLog requires has three parameters, two of which are required. The required parameters are transactional, of Boolean type, and parallel, of Boolean type. If transactional is set to **true**, it indicates that the XML batch file will be created as a single transaction instead of each command being treated as a separated transaction. If parallel is set to **true**, it indicates that all commands in the batch file will be recorded for concurrent execution instead of sequentially as they were recorded.

AMOException Exception Class

You can use AMOException exception class to easily catch exceptions in your application that are thrown by AMO.

AMO will throw exceptions at different problems found. The following table lists the kind of exceptions that are handled by AMO. Exceptions are derived from the

T:Microsoft.AnalysisServices.AmoException class.

Exception	Origin	Description
T:Microsoft.AnalysisServices.AmoException	Base class	Application receives this exception when a required parent object is missing, or when a requested item is not found in a collection.
T:Microsoft.AnalysisServices.OutOfSyncException	Derived from AMOException	Application receives this exception when AMO is out of synchronization with the engine and the engine returns an object reference that AMO does not know about.
T:Microsoft.AnalysisServices.OperationException	Derived from AMOException	This an important exception that is frequently received by applications. This exception contains the details of an error coming from the server, probably because of a faulty AMO operation like Update or Process or Drop.
T:Microsoft.AnalysisServices.ResponseFormatException	Derived from AMOException	This exception occurs when the engine returns a message in a format that AMO does not understand.
T:Microsoft.AnalysisServices.ConnectionException	Derived from AMOException	This exception occurs when a connection cannot

Exception	Origin	Description
		be established (with Server.Connect) or when the connection is lost while AMO is communicating with the engine (for example, during an Update or Process or Drop).

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

Programming Administrative Tasks with AMO

Analysis Management Objects (AMO) is a programming library used from client applications to manage Analysis Services. In this section, you will learn to how to program using the AMO objects.

The AMO library can be thought of as logically related groups of objects that are used to accomplish a specific task.

This section includes the following topics:

Chapter	Contents
Programming AMO Fundamental objects	Describes how to program the Server, Database, DataSource, and DataSourceView objects. Also included here is AMOException.
Programming AMO OLAP basic objects	Describes how to program the Dimension, Cube, MeasureGroup, Partition, and Aggregation objects.
Programming AMO OLAP advanced objects	Describes how to program the Action, KPI, Perspective, ProactiveCaching, and

Chapter	Contents
	Translation objects.
Programming AMO DataMining objects	Describes how to program the MiningStructure and MiningModel objects.
Programming AMO Security objects	Describes how to program the Roles, Members, and Permissions objects.
Programming AMO complementary classes and methods	Describes how to program the Assembly object, Backup and Restore methods, Trace class, and also the CaptureLog class and CaptureXML attribute.

See Also

N:Microsoft.AnalysisServices

[Introducing AMO Concepts](#)

[Introducing AMO Classes](#)

[Analysis Services Objects \(SSAS\)](#)

Programming AMO Fundamental Objects

Fundamental objects are generally simple and straightforward objects. These objects are usually created and instantiated, then when they are no longer needed, the user disconnects from them. Fundamental classes include the following objects: **T:Microsoft.AnalysisServices.Server**, **T:Microsoft.AnalysisServices.Database**, **T:Microsoft.AnalysisServices.DataSource**, and **T:Microsoft.AnalysisServices.DataSourceView**. The only complex object in AMO fundamental objects is **T:Microsoft.AnalysisServices.DataSourceView**, which requires detail to build the abstract model that represents the data source view.

T:Microsoft.AnalysisServices.Server and **T:Microsoft.AnalysisServices.Database** objects are usually required to use the contained objects as OLAP objects or data mining objects.

This topic contains the following sections:

- Server Objects
- AMOException Exception Objects
- Database Objects
- DataSource Objects
- DataSourceView Objects

Server Objects

To use a **T:Microsoft.AnalysisServices.Server** object requires the following steps: connecting to the server, verifying whether the **T:Microsoft.AnalysisServices.Server** object is connected to the server, and if so, disconnecting the **T:Microsoft.AnalysisServices.Server** from the server.

Connecting to the Server Object

Connecting to the server consists of having the right connection string.

The following code sample returns a **T:Microsoft.AnalysisServices.Server** object if the connection is successful, or returns **null** if an error occurs. Errors during the connection process are handled in a try/catch construct. AMO errors are caught by using the

T:Microsoft.AnalysisServices.AmoException exception class. In this example, the error is shown to the user on a message box.

```
static Server ServerConnect( String strStringConnection)
{
    string methodCaption = "ServerConnect method";
    Server svr = new Server();
    try
    {
        svr.Connect(strStringConnection);
    }
    #region ErrorHandling
    catch (AmoException e)
    {
        MessageBox.Show( "AMO exception " + e.ToString());
        svr = null;
    }
    catch (Exception e)
    {
        MessageBox.Show("General exception " + e.ToString());
        svr = null;
    }
    #endregion

    return svr;
}
```

The structure of the connection string is:

"Data source= <server name>".

For a more information about connection string, see

P:Microsoft.SqlServer.Management.Common.OlapConnectionInfo.ConnectionString.

Validating the Connection

Before programming the **T:Microsoft.AnalysisServices.Server** objects, you should verify that you are still connected to the server. The following code sample shows you how to do it. The sample assumes that `svr` is a **T:Microsoft.AnalysisServices.Server** object that exists in your code.

```
if ( (svr != null) && ( svr.Connected))
{
    // Do what it is needed if connection is good
}
```

Disconnecting from the Server

As soon as you are finished, you can disconnect from the server by using the `Disconnect` method. The following code sample shows you how to do it. The sample assumes that `svr` is a **T:Microsoft.AnalysisServices.Server** object that exists in your code.

```
if ( (svr != null) && ( svr.Connected))
{
    svr.Disconnect()
}
```

AmoException Exception Objects

AMO will throw exceptions at different problems found. For a detailed explanation of exceptions, see [Database Objects \(Analysis Services - Multidimensional Data\)](#). The following sample code shows the correct way to capture exceptions in AMO:

```
try
{
    //... some AMO code in here
}

catch ( OutOfSynchException e)
{
    // error handling code for OutOfSynchException
}

catch ( OperationException e)
{
    // error handling code for OperationException
}
```

```

}

catch ( ResponseFormatException e)

{
    // error handling code for ResponseFormatException
}

catch ( ConnectionException e)

{
    // error handling code for ConnectionException
}

catch ( AMOException e)

{
    //... here is the place where you end if it is an AMO exception, but none of the previous
exceptions

    // if you start with AMOException in the first catch you will never see any one of the
previous exceptions
}

```

Database Objects

Working with a **T:Microsoft.AnalysisServices.Database** object is very simple and straightforward. You get an existing database from the database collection of the **T:Microsoft.AnalysisServices.Server** object.

Creating, Dropping, and Finding a Database

The following code sample shows how to create a database by using a database name. Before creating the database, query the **T:Microsoft.AnalysisServices.DatabaseCollection** of the server to see whether the database exists. If the database exists, the database is dropped and afterward created; if the database does not exist then it is created. If the database is to be dropped, then the database is first acquired from the databases collection.

```

static Database CreateDatabase(Server svr, String DatabaseName)
{
    Database db = null;

```



```

        if ( (svr != null) && ( svr.Connected))
        {
            // Drop the database if it already exists

            db = svr.Databases.FindByName(DatabaseName);

            if (db != null)
            {
                db.Drop();
            }

            // Create the database

            db = svr.Databases.Add(DatabaseName);

            db.Update();
        }

        return db;
    }
}

```

To determine whether a database exists in the database collection, the `FindByName` method is used. If the database exists, then the method returns the found database object, if not it returns a null object.

As soon as the **T:Microsoft.AnalysisServices.Database** object is added to the databases collection, the server has to be updated by using its `Update` method. Failing to update the server will cause the **T:Microsoft.AnalysisServices.Database** object not to be created in the server.

Processing a Database

Processing a database, with all the children objects, is very simple because the **T:Microsoft.AnalysisServices.Database** object includes a `Process` method.

The `Process` method can include parameters, but they are not required. If no parameters are specified, then all children objects will be processed with their **ProcessDefault** option. For more information about processing options, see **T:Microsoft.AnalysisServices.Database**.

1. The following sample code process a database by its default value.

```

static Database ProcessDatabase(Database db, ProcessType pt)
{
    db.Process( pt);

    return db;
}

```

DataSource Objects

A **T:Microsoft.AnalysisServices.DataSource** object is the link between Analysis Services and the database where the data resides. The schema that represents the underlying model for Analysis Services is defined by the **T:Microsoft.AnalysisServices.DataSourceView** object. A

T:Microsoft.AnalysisServices.DataSource object can be seen as a connection string to the database where the data resides.

The following sample code shows how to create a **T:Microsoft.AnalysisServices.DataSource** object. The sample verifies that the server still exists, the **T:Microsoft.AnalysisServices.Server** object is connected, and the database exists. If the **T:Microsoft.AnalysisServices.DataSource** object exists, then it is dropped and re-created. The **T:Microsoft.AnalysisServices.DataSource** object is created having the same name and internal ID. In this sample, no checking is performed on the connection string to verify it.

```
static string CreateDataSource(Database db, string strDataSourceName, string
strConnectionString)
{
    Server svr = db.Parent;

    DataSource ds = db.DataSources.FindByName(strDataSourceName);

    if (ds != null)
        ds.Drop();

    // Create the data source

    ds = db.DataSources.Add(strDataSourceName, strDataSourceName);

    ds.ConnectionString = strConnectionString;

    // Send the data source definition to the server.

    ds.Update();

    return ds.Name;
}
```

DataSourceView Objects

T:Microsoft.AnalysisServices.DataSourceView object is responsible for holding the schema model for Analysis Services. For the **T:Microsoft.AnalysisServices.DataSourceView** object to hold the schema, the schema must first be constructed. Schemas are constructed over DataSet objects, from the System.Data namespace.

The following sample code will create part of the schema that is included in AdventureWorks Analysis Services Project sample. For more information about installing the samples, see [AdventureWorks2012 Sample Databases](#). The current sample creates schema definitions for tables, computed columns, relations, and composite relations. Schemas are persisted data sets.

The sample code does the following:

1. Create a **T:Microsoft.AnalysisServices.DataSourceView** object.
Verify first if the **T:Microsoft.AnalysisServices.DataSource** object exists; if **true**, then drop the **T:Microsoft.AnalysisServices.DataSource** and create it. If the **T:Microsoft.AnalysisServices.DataSource** does not exist, create it.
2. Open a connection to the database using **T:Microsoft.AnalysisServices.DataSource** connection string.
3. Create the schema.
The schema consists of the following:
 - A table definition, `AddTable()` method.
 - An optional set of calculated columns, `AddComputedColumn()` method.
 - An optional set of relations, `AddRelation`.
 - An optional set of composite relations, `AddCompositeRelations`.
4. Update the server.



Note

The following sample code is trimmed for readability purposes; the complete code is included at the end of this topic.



Note

The following methods are part of the sample code: `AddTable`, `AddComputedColumn`, `AddRelation`, and `AddCompositeRelation`.



Note

The clause 'WHERE 1=0' is to avoid the query from returning rows to the `DataSet` object.

```
static DataSourceView CreateDataSourceView(Database db, string strDataSourceName)
{
    // Create the data source view

    DataSourceView dsv = db.DataSourceViews.FindByName(strDataSourceName);

    if ( dsv != null)

        dsv.Drop();

    dsv = db.DataSourceViews.Add(strDataSourceName);

    dsv.DataSourceID = strDataSourceName;

    dsv.Schema = new DataSet();

    dsv.Schema.Locale = CultureInfo.CurrentCulture;

    // Open a connection to the data source

    OleDbConnection connection

        = new OleDbConnection(dsv.DataSource.ConnectionString);
```

```

connection.Open();

#region Create tables

// Add the DimTime table
AddTable(dsv, connection, "DimTime");

AddComputedColumn(dsv, connection, "DimTime", "SimpleDate", "DATENAME(mm,
FullDateAlternateKey) + ' ' + DATENAME(dd, FullDateAlternateKey) + ',' + ' ' + DATENAME(yy,
FullDateAlternateKey)");

// Add the DimProductCategory table
AddTable(dsv, connection, "DimProductCategory");

// Add the DimProductSubcategory table
AddTable(dsv, connection, "DimProductSubcategory");

AddRelation(dsv, "DimProductSubcategory", "ProductCategoryKey", "DimProductCategory",
"ProductCategoryKey");

// Add the FactInternetSales table
AddTable(dsv, connection, "FactInternetSales");

"DimTime", "TimeKey");

AddRelation(dsv, "FactInternetSales", "ShipDateKey", "DimTime", "TimeKey");
AddRelation(dsv, "FactInternetSales", "DueDateKey", "DimTime", "TimeKey");

// Add the FactInternetSalesReason table
AddTable(dsv, connection, "FactInternetSalesReason");

AddCompositeRelation(dsv, "FactInternetSalesReason", "FactInternetSales",
"SalesOrderNumber", "SalesOrderLineNumber");

dsv.Update();

#endregion

// Send the data source view definition to the server
dsv.Update();

```

```

        return dsv;
    }

    static void AddTable(DataSourceView dsv, OleDbConnection connection, String tableName)
    {
        string strSelectText = "SELECT * FROM [dbo].[" + tableName + "] WHERE 1=0";
        OleDbDataAdapter adapter = new OleDbDataAdapter(strSelectText, connection);
        DataTable[] dataTables = adapter.FillSchema(dsv.Schema,
            SchemaType.Mapped, tableName);
        DataTable dataTable = dataTables[0];

        dataTable.ExtendedProperties.Add("TableType", "Table");
        dataTable.ExtendedProperties.Add("DbSchemaName", "dbo");
        dataTable.ExtendedProperties.Add("DbTableName", tableName);
        dataTable.ExtendedProperties.Add("FriendlyName", tableName);

        dataTable = null;
        dataTables = null;
        adapter = null;
    }

    static void AddComputedColumn(DataSourceView dsv, OleDbConnection connection, String
tableName, String computedColumnName, String expression)
    {
        DataSet tmpDataSet = new DataSet();
        tmpDataSet.Locale = CultureInfo.CurrentCulture;
        OleDbDataAdapter adapter = new OleDbDataAdapter("SELECT ("
            + expression + ") AS [" + computedColumnName + "] FROM [dbo].["
            + tableName + "] WHERE 1=0", connection);
        DataTable[] dataTables = adapter.FillSchema(tmpDataSet,
            SchemaType.Mapped, tableName);
        DataTable dataTable = dataTables[0];
    }

```

```

        DataColumn dataColumn = dataTable.Columns[computedColumnName];

        dataTable.Constraints.Clear();

        dataTable.Columns.Remove(dataColumn);

        dataColumn.ExtendedProperties.Add("DbColumnName", computedColumnName);
        dataColumn.ExtendedProperties.Add("ComputedColumnExpression",
            expression);
        dataColumn.ExtendedProperties.Add("IsLogical", "True");

        dsv.Schema.Tables[tableName].Columns.Add(dataColumn);

        dataColumn = null;
        dataTable = null;
        dataTables = null;
        adapter = null;
        tmpDataSet = null;
    }

    static void AddRelation(DataSourceView dsv, String fkTableName, String fkColumnName,
String pkTableName, String pkColumnName)
    {
        DataColumn fkColumn
            = dsv.Schema.Tables[fkTableName].Columns[fkColumnName];

        DataColumn pkColumn
            = dsv.Schema.Tables[pkTableName].Columns[pkColumnName];

        dsv.Schema.Relations.Add("FK_" + fkTableName + "_"
            + fkColumnName, pkColumn, fkColumn, true);
    }

    static void AddCompositeRelation(DataSourceView dsv, String fkTableName, String
pkTableName, String columnName1, String columnName2)
    {
        DataColumn[] fkColumns = new DataColumn[2];

```

```

        fkColumns[0] = dsv.Schema.Tables[fkTableName].Columns[columnName1];
        fkColumns[1] = dsv.Schema.Tables[fkTableName].Columns[columnName2];

        DataColumn[] pkColumns = new DataColumn[2];
        pkColumns[0] = dsv.Schema.Tables[pkTableName].Columns[columnName1];
        pkColumns[1] = dsv.Schema.Tables[pkTableName].Columns[columnName2];

        dsv.Schema.Relations.Add("FK_" + fkTableName + "_" + columnName1
            + "_" + columnName2, pkColumns, fkColumns, true);
    }

```

In the sample code, the `AddTable` and `AddComputedColumn` methods use the `FillSchema` method of the `DataAdapter` object to add a `DataTable` to a `DataSet` and to configure the schema to match that in the data source. The extended properties add required info to configure the schema for Analysis Services.

In the sample code, the `AddRelation` and `AddCompositeRelation` methods add the relation columns, depending on the existing schema and the existing columns on the model. Columns must be part of the tables defined in the schema for these methods to work.

The following is the complete code sample:

```

static DataSourceView CreateDataSourceView(Database db, string strDataSourceName)
{
    // Create the data source view

    DataSourceView dsv = db.DataSourceViews.FindByName(strDataSourceName);

    if ( dsv != null)
        dsv.Drop();

    dsv = db.DataSourceViews.Add(strDataSourceName);

    dsv.DataSourceID = strDataSourceName;

    dsv.Schema = new DataSet();

    dsv.Schema.Locale = CultureInfo.CurrentCulture;

    // Open a connection to the data source

    OleDbConnection connection
        = new OleDbConnection(dsv.DataSource.ConnectionString);

    connection.Open();
}

```

```

#region Create tables

// Add the DimTime table

AddTable(dsv, connection, "DimTime");

AddComputedColumn(dsv, connection, "DimTime", "SimpleDate", "DATENAME(mm,
FullDateAlternateKey) + ' ' + DATENAME(dd, FullDateAlternateKey) + ',' + ' ' + DATENAME(yy,
FullDateAlternateKey)");

AddComputedColumn(dsv, connection, "DimTime", "CalendarYearDesc", "'CY' + ' ' +
CalendarYear");

AddComputedColumn(dsv, connection, "DimTime", "CalendarSemesterDesc", "CASE WHEN
CalendarSemester = 1 THEN 'H1'+' ' + 'CY' + ' ' + CONVERT(CHAR (4), CalendarYear) ELSE 'H2'+' ' +
'CY' + ' ' + CONVERT(CHAR (4), CalendarYear) END");

AddComputedColumn(dsv, connection, "DimTime", "CalendarQuarterDesc", "'Q' +
CONVERT(CHAR (1), CalendarQuarter) + ' ' + 'CY' + ' ' + CONVERT(CHAR (4), CalendarYear)");

AddComputedColumn(dsv, connection, "DimTime", "MonthName", "EnglishMonthName+' ' +
CONVERT(CHAR (4), CalendarYear)");

AddComputedColumn(dsv, connection, "DimTime", "FiscalYearDesc", "'FY' + ' ' +
FiscalYear");

AddComputedColumn(dsv, connection, "DimTime", "FiscalSemesterDesc", "CASE WHEN
FiscalSemester = 1 THEN 'H1'+' ' + 'FY' + ' ' + CONVERT(CHAR (4), FiscalYear) ELSE 'H2'+' ' + 'FY' +
' ' + CONVERT(CHAR (4), FiscalYear) END");

AddComputedColumn(dsv, connection, "DimTime", "FiscalQuarterDesc", "'Q' +
CONVERT(CHAR (1), FiscalQuarter) + ' ' + 'FY' + ' ' + CONVERT(CHAR (4), FiscalYear)");

AddComputedColumn(dsv, connection, "DimTime", "FiscalMonthNumberOfYear", "CASE WHEN
MonthNumberOfYear = '1' THEN CONVERT(int,'7') WHEN MonthNumberOfYear = '2' THEN
CONVERT(int,'8') WHEN MonthNumberOfYear = '3' THEN CONVERT(int,'9') WHEN MonthNumberOfYear = '4'
THEN CONVERT(int,'10') WHEN MonthNumberOfYear = '5' THEN CONVERT(int,'11') WHEN
MonthNumberOfYear = '6' THEN CONVERT(int,'12') WHEN MonthNumberOfYear = '7' THEN
CONVERT(int,'1') WHEN MonthNumberOfYear = '8' THEN CONVERT(int,'2') WHEN MonthNumberOfYear = '9'
THEN CONVERT(int,'3') WHEN MonthNumberOfYear = '10' THEN CONVERT(int,'4') WHEN MonthNumberOfYear
= '11' THEN CONVERT(int,'5') WHEN MonthNumberOfYear = '12' THEN CONVERT(int,'6') END");

dsv.Update();

// Add the DimGeography table

AddTable(dsv, connection, "DimGeography");

```



```

// Add the DimProductCategory table

AddTable(dsv, connection, "DimProductCategory");

// Add the DimProductSubcategory table

AddTable(dsv, connection, "DimProductSubcategory");

AddRelation(dsv, "DimProductSubcategory", "ProductCategoryKey", "DimProductCategory",
"ProductCategoryKey");

// Add the DimProduct table

AddTable(dsv, connection, "DimProduct");

AddComputedColumn(dsv, connection, "DimProduct", "ProductLineName", "CASE ProductLine
WHEN 'M' THEN 'Mountain' WHEN 'R' THEN 'Road' WHEN 'S' THEN 'Accessory' WHEN 'T' THEN 'Touring'
ELSE 'Components' END");

AddRelation(dsv, "DimProduct", "ProductSubcategoryKey", "DimProductSubcategory",
"ProductSubcategoryKey");

dsv.Update();

// Add the DimCustomer table

AddTable(dsv, connection, "DimCustomer");

AddComputedColumn(dsv, connection, "DimCustomer", "FullName", "CASE WHEN MiddleName
IS NULL THEN FirstName + ' ' + LastName ELSE FirstName + ' ' + MiddleName + ' ' + LastName END");

AddComputedColumn(dsv, connection, "DimCustomer", "GenderDesc", "CASE WHEN Gender =
'M' THEN 'Male' ELSE 'Female' END");

AddComputedColumn(dsv, connection, "DimCustomer", "MaritalStatusDesc", "CASE WHEN
MaritalStatus = 'S' THEN 'Single' ELSE 'Married' END");

AddRelation(dsv, "DimCustomer", "GeographyKey", "DimGeography", "GeographyKey");

// Add the DimReseller table

AddTable(dsv, connection, "DimReseller");

AddComputedColumn(dsv, connection, "DimReseller", "OrderFrequencyDesc", "CASE WHEN
OrderFrequency = 'A' THEN 'Annual' WHEN OrderFrequency = 'S' THEN 'Bi-Annual' ELSE 'Quarterly'
END");

AddComputedColumn(dsv, connection, "DimReseller", "OrderMonthDesc", "CASE WHEN
OrderMonth = '1' THEN 'January' WHEN OrderMonth = '2' THEN 'February' WHEN OrderMonth = '3' THEN
'March' WHEN OrderMonth = '4' THEN 'April' WHEN OrderMonth = '5' THEN 'May' WHEN OrderMonth = '6'

```

```

THEN 'June' WHEN OrderMonth = '7' THEN 'July' WHEN OrderMonth = '8' THEN 'August' WHEN OrderMonth
= '9' THEN 'September' WHEN OrderMonth = '10' THEN 'October' WHEN OrderMonth = '11' THEN
'November' WHEN OrderMonth = '12' THEN 'December' ELSE 'Never Ordered' END");

```

```

// Add the DimCurrency table

```

```

AddTable(dsv, connection, "DimCurrency");

```

```

dsv.Update();

```

```

// Add the DimSalesReason table

```

```

AddTable(dsv, connection, "DimSalesReason");

```

```

// Add the FactInternetSales table

```

```

AddTable(dsv, connection, "FactInternetSales");

```

```

AddRelation(dsv, "FactInternetSales", "ProductKey", "DimProduct", "ProductKey");

```

```

AddRelation(dsv, "FactInternetSales", "CustomerKey", "DimCustomer", "CustomerKey");

```

```

AddRelation(dsv, "FactInternetSales", "OrderDateKey", "DimTime", "TimeKey");

```

```

AddRelation(dsv, "FactInternetSales", "ShipDateKey", "DimTime", "TimeKey");

```

```

AddRelation(dsv, "FactInternetSales", "DueDateKey", "DimTime", "TimeKey");

```

```

AddRelation(dsv, "FactInternetSales", "CurrencyKey", "DimCurrency", "CurrencyKey");

```

```

dsv.Update();

```

```

// Add the FactResellerSales table

```

```

AddTable(dsv, connection, "FactResellerSales");

```

```

AddRelation(dsv, "FactResellerSales", "ProductKey", "DimProduct", "ProductKey");

```

```

AddRelation(dsv, "FactResellerSales", "ResellerKey", "DimReseller", "ResellerKey");

```

```

AddRelation(dsv, "FactResellerSales", "OrderDateKey", "DimTime", "TimeKey");

```

```

AddRelation(dsv, "FactResellerSales", "ShipDateKey", "DimTime", "TimeKey");

```

```

AddRelation(dsv, "FactResellerSales", "DueDateKey", "DimTime", "TimeKey");

```

```

AddRelation(dsv, "FactResellerSales", "CurrencyKey", "DimCurrency", "CurrencyKey");

```

```

// Add the FactInternetSalesReason table

```

```

AddTable(dsv, connection, "FactInternetSalesReason");

```

```

AddCompositeRelation(dsv, "FactInternetSalesReason", "FactInternetSales",
"SalesOrderNumber", "SalesOrderLineNumber");

```

```

        dsv.Update();

        // Add the FactCurrencyRate table
        AddTable(dsv, connection, "FactCurrencyRate");
        AddRelation(dsv, "FactCurrencyRate", "CurrencyKey", "DimCurrency", "CurrencyKey");
        AddRelation(dsv, "FactCurrencyRate", "TimeKey", "DimTime", "TimeKey");

        #endregion

        // Send the data source view definition to the server
        dsv.Update();

        return dsv;
    }

    static void AddTable(DataSourceView dsv, OleDbConnection connection, String tableName)
    {
        string strSelectText = "SELECT * FROM [dbo].[" + tableName + "] WHERE 1=0";
        OleDbDataAdapter adapter = new OleDbDataAdapter(strSelectText, connection);
        DataTable[] dataTables = adapter.FillSchema(dsv.Schema,
            SchemaType.Mapped, tableName);
        DataTable dataTable = dataTables[0];

        dataTable.ExtendedProperties.Add("TableType", "Table");
        dataTable.ExtendedProperties.Add("DbSchemaName", "dbo");
        dataTable.ExtendedProperties.Add("DbTableName", tableName);
        dataTable.ExtendedProperties.Add("FriendlyName", tableName);

        dataTable = null;
        dataTables = null;
        adapter = null;
    }

```

```

static void AddComputedColumn(DataSourceView dsv, OleDbConnection connection, String
tableName, String computedColumnName, String expression)
{
    DataSet tmpDataSet = new DataSet();
    tmpDataSet.Locale = CultureInfo.CurrentCulture;
    OleDbDataAdapter adapter = new OleDbDataAdapter("SELECT ("
        + expression + ") AS [" + computedColumnName + "] FROM [dbo].["
        + tableName + "] WHERE 1=0", connection);
    DataTable[] dataTables = adapter.FillSchema(tmpDataSet,
        SchemaType.Mapped, tableName);
    DataTable dataTable = dataTables[0];
    DataColumn dataColumn = dataTable.Columns[computedColumnName];

    dataTable.Constraints.Clear();
    dataTable.Columns.Remove(dataColumn);

    dataColumn.ExtendedProperties.Add("DbColumnName", computedColumnName);
    dataColumn.ExtendedProperties.Add("ComputedColumnExpression",
        expression);
    dataColumn.ExtendedProperties.Add("IsLogical", "True");

    dsv.Schema.Tables[tableName].Columns.Add(dataColumn);

    dataColumn = null;
    dataTable = null;
    dataTables = null;
    adapter = null;
    tmpDataSet = null;
}

static void AddRelation(DataSourceView dsv, String fkTableName, String fkColumnName,
String pkTableName, String pkColumnName)
{

```

```

        DataColumn fkColumn
            = dsv.Schema.Tables[fkTableName].Columns[fkColumnName];

        DataColumn pkColumn
            = dsv.Schema.Tables[pkTableName].Columns[pkColumnName];

        dsv.Schema.Relations.Add("FK_" + fkTableName + "_"
            + fkColumnName, pkColumn, fkColumn, true);
    }

    static void AddCompositeRelation(DataSourceView dsv, String fkTableName, String
pkTableName, String columnName1, String columnName2)
    {
        DataColumn[] fkColumns = new DataColumn[2];

        fkColumns[0] = dsv.Schema.Tables[fkTableName].Columns[columnName1];
        fkColumns[1] = dsv.Schema.Tables[fkTableName].Columns[columnName2];

        DataColumn[] pkColumns = new DataColumn[2];

        pkColumns[0] = dsv.Schema.Tables[pkTableName].Columns[columnName1];
        pkColumns[1] = dsv.Schema.Tables[pkTableName].Columns[columnName2];

        dsv.Schema.Relations.Add("FK_" + fkTableName + "_" + columnName1
            + "_" + columnName2, pkColumns, fkColumns, true);
    }

```

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[AMO Fundamental Classes](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

Programming AMO OLAP Basic Objects

Creating complex Analysis Services objects is simple and straightforward but requires attention to detail. This topic explains the programming details of OLAP basic objects. This topic contains the following sections:

- Dimension Objects

- Cube Objects
- MeasureGroup Objects
- Partition Objects
- Aggregation Objects

Dimension Objects

To administer or process a dimension, you program the **T:Microsoft.AnalysisServices.Dimension** object.

Creating, Dropping, and Finding a Dimension

Creating a **T:Microsoft.AnalysisServices.Dimension** object is accomplished in four steps:

1. Create the dimension object and populate basic attributes.

Basic attributes are Name, Dimension Type, Storage Mode, Data Source Binding, Attribute All Member Name, and other dimension attributes.

Before creating a dimension, you should verify that the dimension does not already exist. If the dimension exists, then the dimension is dropped and re-created.

2. Create the attributes that define the dimension.

Each attribute has to be added individually to the schema before using it (find `CreateDataItem` method at the end of the sample code), and then can be added to the attributes collection of the dimension.

Key and Name column must be defined in all attributes.

The primary key attribute of the dimension should be defined as `AttributeUsage.Key` to make clear that this attribute is the key access to the dimension.

3. Create the hierarchies that the user will access to navigate the dimension.

When you are creating hierarchies, the level order is defined by the order in which levels are created from top to bottom. The highest level is the first added to the levels collection of the hierarchy.

4. Update the server by using the `Update` method of the current dimension.

The following sample code creates the Product dimension for the [AdventureWorks2012 Sample Databases](#).

```
static void CreateProductDimension(Database db, string datasourceName)
{
    // Create the Product dimension
    Dimension dim = db.Dimensions.FindByName("Product");
    if ( dim != null)
        dim.Drop();
    dim = db.Dimensions.Add("Product");
    dim.Type = DimensionType.Products;
```

```

dim.UnknownMember = UnknownMemberBehavior.Hidden;

dim.AttributeAllMemberName = "All Products";

dim.Source = new DataSourceViewBinding(datasourceName);

dim.StorageMode = DimensionStorageMode.Molap;

#region Create attributes

DimensionAttribute attr;

attr = dim.Attributes.Add("Product Name");
attr.Usage = AttributeUsage.Key;
attr.Type = AttributeType.Product;
attr.OrderBy = OrderBy.Name;
attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProduct",
"ProductKey"));
attr.NameColumn = CreateDataItem(db.DataSourceViews[0], "DimProduct",
"EnglishProductName");

attr = dim.Attributes.Add("Product Line");
attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProduct",
"ProductLine"));
attr.NameColumn = CreateDataItem(db.DataSourceViews[0], "DimProduct",
"ProductLineName");

attr = dim.Attributes.Add("Model Name");
attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProduct",
"ModelName"));
attr.AttributeRelationships.Add(new AttributeRelationship("Product Line"));
attr.AttributeRelationships.Add(new AttributeRelationship("Subcategory"));

attr = dim.Attributes.Add("Subcategory");
attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProductSubcategory",
"ProductSubcategoryKey"));
attr.KeyColumns[0].NullProcessing = NullProcessing.UnknownMember;

```

```

        attr.NameColumn = CreateDataItem(db.DataSourceViews[0], "DimProductSubcategory",
"EnglishProductSubcategoryName");

        attr.AttributeRelationships.Add(new AttributeRelationship("Category"));

        attr = dim.Attributes.Add("Category");

        attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProductCategory",
"ProductCategoryKey"));

        attr.NameColumn = CreateDataItem(db.DataSourceViews[0], "DimProductCategory",
"EnglishProductCategoryName");

        attr = dim.Attributes.Add("List Price");

        attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProduct",
"ListPrice"));

        attr.AttributeHierarchyEnabled = false;

        attr = dim.Attributes.Add("Size");

        attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProduct", "Size"));

        attr.AttributeHierarchyEnabled = false;

        attr = dim.Attributes.Add("Weight");

        attr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "DimProduct", "Weight"));

        attr.AttributeHierarchyEnabled = false;

#endregion

#region Create hierarchies

Hierarchy hier;

hier = dim.Hierarchies.Add("Product Model Categories");

hier.AllMemberName = "All Products";

hier.Levels.Add("Category").SourceAttributeID = "Category";

hier.Levels.Add("Subcategory").SourceAttributeID = "Subcategory";

hier.Levels.Add("Model Name").SourceAttributeID = "Model Name";

```



```

        hier = dim.Hierarchies.Add("Product Categories");

        hier.AllMemberName = "All Products";

        hier.Levels.Add("Category").SourceAttributeID = "Category";
        hier.Levels.Add("Subcategory").SourceAttributeID = "Subcategory";
        hier.Levels.Add("Model Name").SourceAttributeID = "Product Name";

        hier = dim.Hierarchies.Add("Product Model Lines");

        hier.AllMemberName = "All Products";

        hier.Levels.Add("Subcategory").SourceAttributeID = "Product Line";
        hier.Levels.Add("Model Name").SourceAttributeID = "Model Name";

        #endregion

        dim.Update();
    }

    static DataItem CreateDataItem(DataSourceView dsv, string tableName, string columnName)
    {
        DataTable dataTable = ((DataSourceView)dsv).Schema.Tables[tableName];
        DataColumn dataColumn = dataTable.Columns[columnName];

        return new DataItem(tableName, columnName,
            OleDbTypeConverter.GetRestrictedOleDbType(dataColumn.DataType));
    }

```

Processing a Dimension

Processing a dimension is as simple as using the Process method of the **T:Microsoft.AnalysisServices.Dimension** object.

Processing a dimension can affect all cubes that use the dimension. For more information about processing options, see [AdventureWorks Sample Data Warehouse](#) and [Processing Analysis Services Objects](#).

The following code does an incremental update in all dimensions of a supplied database:

```

static void UpdateAllDimensions(Database db)
{

```

```

        foreach (Dimension dim in db.Dimensions)
        {
            dim.Process(ProcessType.ProcessUpdate);
        }
    }
}

```

Cube Objects

To administer or process a cube, you program the **T:Microsoft.AnalysisServices.Cube** object.

Creating, Dropping, and Finding a Cube

Managing cubes is similar to managing dimensions. Creating a **T:Microsoft.AnalysisServices.Cube** object is accomplished in four steps:

1. Create the cube object and populate basic attributes.

Basic attributes are Name, Storage Mode, Data Source Binding, Default Measure, and other cube attributes.

Before creating a cube you should verify that the cube does not exist. In the sample if the cube exists the cube is dropped and then re-created.

2. Add the dimensions of the cube.

Dimensions are added to the current cube dimensions collection from the database; dimensions in the cube are references to the database dimensions collection. Each dimension has to be mapped to the cube individually. In the sample dimensions are mapped providing: the database dimension Internal Identifier, a Name for the dimension in the cube and an Id for the named dimension in the cube.

In the sample code notice that "Date" dimension is added three times, every time is added by using a different cube dimension name: Date, Ship Date, Delivery Date. These dimensions are called "role playing" dimensions. The base dimension is the same (Date), but in the fact table the dimension is used in different "roles" (Order Date, Ship Date, Delivery Date) -see "Creating, dropping and finding a MeasureGroup" later in this document to understand how "role playing" dimensions are defined.

3. Create the Measure Groups that the user will access to browse the data of the cube.

Measure group creation will be explained in "Creating, dropping and finding a MeasureGroup" later in this document. The sample wraps measure group creation in different methods, one for each measure group.

4. Update the server by using the Update method of current cube.

The update method is used with the Update option ExpandFull to make sure that all objects are fully updated in the server.

The following code sample creates the parts of the Adventure Works cube. The code sample does not create all dimensions or measure groups that are included in the Adventure Works Analysis Services Project sample. For more information about installing the samples, see [AdventureWorks2012 Sample Databases](#).

```

static void CreateAdventureWorksCube(Database db, string datasourceName)
{

```

```

// Create the Adventure Works cube

Cube cube = db.Cubes.FindByName("Adventure Works");

if ( cube != null)

    cube.Drop();

db.Cubes.Add("Adventure Works");

cube.DefaultMeasure = "[Reseller Sales Amount]";

cube.Source = new DataSourceViewBinding(datasourceName);

cube.StorageMode = StorageMode.Molap;


#region Create cube dimensions

Dimension dim;

dim = db.Dimensions.GetByName("Date");

cube.Dimensions.Add(dim.ID, "Date", "Order Date Key - Dim Time");

cube.Dimensions.Add(dim.ID, "Ship Date",

    "Ship Date Key - Dim Time");

cube.Dimensions.Add(dim.ID, "Delivery Date",

    "Delivery Date Key - Dim Time");


dim = db.Dimensions.GetByName("Customer");

cube.Dimensions.Add(dim.ID);


dim = db.Dimensions.GetByName("Reseller");

cube.Dimensions.Add(dim.ID);

#endregion


#region Create measure groups

CreateSalesReasonsMeasureGroup(cube);

CreateInternetSalesMeasureGroup(cube);

CreateResellerSalesMeasureGroup(cube);

CreateCustomersMeasureGroup(cube);

```

```

        CreateCurrencyRatesMeasureGroup(cube);

        #endregion

        cube.Update(UpdateOptions.ExpandFull);
    }

```

Processing a Cube

Processing a cube is as simple as using the Process method of the

T:Microsoft.AnalysisServices.Cube object. Processing a cube also processes all measure groups in the cube, and all partitions in the measure group. In a cube, partitions are the only objects that can be processed; for the purposes of processing, measure groups are only containers of partitions. The specified type of processing for the cube propagates to the partitions. Processing of cube and measure group internally is resolved to processing of dimensions and partitions.

For more information about processing options, see [Processing Objects \(XMLA\)](#), and [Processing Analysis Services Objects](#).

The following code will do a full process on all cubes in a specified database:

```

foreach (Cube cube in db.Cubes)
    cube.Process(ProcessType.ProcessFull);
}

```

MeasureGroup Objects

To administer or process a measure group, you program the

T:Microsoft.AnalysisServices.MeasureGroup object.

Creating, Dropping, and Finding a MeasureGroup

Managing measure groups is similar to managing dimensions and cubes. Creating a

T:Microsoft.AnalysisServices.MeasureGroup object is accomplished in the following steps:

1. Create the measure group object and populate the basic attributes.
Basic attributes include Name, Storage Mode, Processing Mode, Default Measure, and other measure group attributes.
Before creating a measure group, verify that the measure group does not exist. In the sample code that follows, if the measure group exists, then the measure group is dropped and re-created.
2. Create the measures of the measure group. For each measure created, the following attributes are assigned: name, aggregation function, source column, format string. Other attributes can also be assigned. Note that in the sample code that follows, the `CreateDataItem` method adds the column to the schema.
3. Add the dimensions of the measure group.

4. Dimensions are added to the current measure group dimensions collection from the parent cube dimensions collection. As soon as the dimension is included in the measure group dimensions collection, a key column from the fact table can be mapped to the dimension so that the measure group can be browsed through the dimension.

In the sample code that follows, see the lines under "Mapping dimension and key column from fact table". The role playing dimensions are implemented by linking different surrogate keys to the same dimension under different names. For each one of the role playing dimensions (Date, Ship Date, Delivery Date), a different surrogate key is linked to it (OrderDateKey, ShipDateKey, DueDateKey). All keys are from the fact table FactInternetSales.

5. Add the designed partitions of the measure group.

The in the sample code that follows, partition creation is wrapped in one method.

6. Update the server by using the Update method of current measure group.

In the sample code that follows, all measure groups are updated when the cube is updated. The following sample code will create the InternetSales measure group of the Adventure Works Analysis Services Project sample. For more information about installing the samples, see [AdventureWorks2012 Sample Databases](#).

```
static void CreateInternetSalesMeasureGroup(Cube cube)
{
    // Create the Internet Sales measure group
    Database db = cube.Parent;

    MeasureGroup mg = cube.MeasureGroups.FindByName("Internet Sales");

    if ( mg != null)
        mg.Drop();

    mg = cube.MeasureGroups.Add("Internet Sales");

    mg.StorageMode = StorageMode.Molap;

    mg.ProcessingMode = ProcessingMode.LazyAggregations;

    mg.Type = MeasureGroupType.Sales;

    #region Create measures

    Measure meas;

    meas = mg.Measures.Add("Internet Sales Amount");

    meas.AggregateFunction = AggregationFunction.Sum;

    meas.FormatString = "Currency";
```

```

        meas.Source = CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"SalesAmount");

    meas = mg.Measures.Add("Internet Order Quantity");
    meas.AggregateFunction = AggregationFunction.Sum;
    meas.FormatString = "#, #";
    meas.Source = CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"OrderQuantity");

    meas = mg.Measures.Add("Internet Unit Price");
    meas.AggregateFunction = AggregationFunction.Sum;
    meas.FormatString = "Currency";
    meas.Visible = false;
    meas.Source = CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"UnitPrice");

    meas = mg.Measures.Add("Internet Total Product Cost");
    meas.AggregateFunction = AggregationFunction.Sum;
    //meas.MeasureExpression = "[Internet Total Product Cost] * [Average Rate]";
    meas.FormatString = "Currency";
    meas.Source = CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"TotalProductCost");

    meas = mg.Measures.Add("Internet Order Count");
    meas.AggregateFunction = AggregationFunction.Count;
    meas.FormatString = "#, #";
    meas.Source = CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"ProductKey");

    #endregion

    #region Create measure group dimensions

    CubeDimension cubeDim;

```

```

RegularMeasureGroupDimension regMgDim;

ManyToManyMeasureGroupDimension mmMgDim;

MeasureGroupAttribute mgAttr;


// Mapping dimension and key column from fact table
// > select dimension and add it to the measure group
cubeDim = cube.Dimensions.GetByName("Date");
regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);
mg.Dimensions.Add(regMgDim);


// > add key column from dimension and map it with
// the surrogate key in the fact table
mgAttr = regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Date").ID);
// this is dimension key column
mgAttr.Type = MeasureGroupAttributeType.Granularity;
mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"OrderDateKey")); // this surrogate key in fact table


cubeDim = cube.Dimensions.GetByName("Ship Date");
regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);
mg.Dimensions.Add(regMgDim);
mgAttr = regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Date").ID);
mgAttr.Type = MeasureGroupAttributeType.Granularity;
mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"ShipDateKey"));


cubeDim = cube.Dimensions.GetByName("Delivery Date");
regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);
mg.Dimensions.Add(regMgDim);
mgAttr = regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Date").ID);
mgAttr.Type = MeasureGroupAttributeType.Granularity;
mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"DueDateKey"));

```

```

cubeDim = cube.Dimensions.GetByName("Customer");
regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);
mg.Dimensions.Add(regMgDim);
mgAttr = regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Full
Name").ID);

mgAttr.Type = MeasureGroupAttributeType.Granularity;
mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"CustomerKey"));


cubeDim = cube.Dimensions.GetByName("Product");
regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);
mg.Dimensions.Add(regMgDim);
mgAttr = regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Product
Name").ID);

mgAttr.Type = MeasureGroupAttributeType.Granularity;
mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"ProductKey"));


cubeDim = cube.Dimensions.GetByName("Source Currency");
regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);
mg.Dimensions.Add(regMgDim);
mgAttr =
regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Currency").ID);
mgAttr.Type = MeasureGroupAttributeType.Granularity;
mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"CurrencyKey"));


cubeDim = cube.Dimensions.GetByName("Sales Reason");
mmMgDim = new ManyToManyMeasureGroupDimension();
mmMgDim.CubeDimensionID = cubeDim.ID;
mmMgDim.MeasureGroupID = cube.MeasureGroups.GetByName("Sales Reasons").ID;
mg.Dimensions.Add(mmMgDim);


cubeDim = cube.Dimensions.GetByName("Internet Sales Order Details");

```



```

        regMgDim = new RegularMeasureGroupDimension(cubeDim.ID);

        mg.Dimensions.Add(regMgDim);

        mgAttr = regMgDim.Attributes.Add(cubeDim.Dimension.Attributes.GetByName("Sales Order
Key").ID);

        mgAttr.Type = MeasureGroupAttributeType.Granularity;

        mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"SalesOrderNumber"));

        mgAttr.KeyColumns.Add(CreateDataItem(db.DataSourceViews[0], "FactInternetSales",
"SalesOrderLineNumber"));

        #endregion

        #region Create partitions

        CreateInternetSalesMeasureGroupPartitions( mg)

        #endregion
    }

```

Processing a Measure Group

Processing a measure group is as simple as using the Process method of the **T:Microsoft.AnalysisServices.MeasureGroup** object. Processing a measure group will process all partitions that belong to the measure group. Processing a measure group internally is resolved to processing dimensions and partitions. See Processing a Partition in this document.

For more information about processing options, see [Processing Objects \(XMLA\)](#), and [Processing Analysis Services Objects](#).

The following code will do a full process in all measure groups of a supplied cube.

```

static void FullProcessAllMeasureGroups(Cube cube)
{
    foreach (MeasureGroup mg in cube.MeasureGroups)
        mg.Process(ProcessType.ProcessFull);
}

```

Partition Objects

To administer or process a partition, you program a **T:Microsoft.AnalysisServices.Partition** object.

Creating, Dropping, and Finding a Partition

Partitions are simple objects that can be created in two steps.

1. Create the partition object and populate the basic attributes.

Basic attributes are Name, Storage Mode, partition source, Slice, as well as other measure group attributes. Partition source defines the SQL select statement for current partition. Slice is an MDX expression specifying a tuple or a set that delimits a part of the dimensions from the parent measure group that are contained in the current partition. For MOLAP partitions, slicing is determined automatically every time that the partition is processed.

Before creating a partition, you should verify that the partition does not exist. In the sample code that follows, if the partition exists, it is dropped and then re-created.

2. Update the server by using the Update method of the current partition.

In the sample code that follows, all partitions are updated when the cube is updated.

The following code sample creates partitions for the 'InternetSales' measure group.

```
static void CreateInternetSalesMeasureGroupPartitions(MeasureGroup mg)
{
    Partition part;

    part = mg.Partitions.FindByName("Internet_Sales_184");
    if ( part != null)
        part.Drop();

    part = mg.Partitions.Add("Internet_Sales_184");
    part.StorageMode = StorageMode.Molap;
    part.Source = new QueryBinding(db.DataSources[0].ID, "SELECT * FROM
[dbo].[FactInternetSales] WHERE OrderDateKey <= '184'");
    part.Slice = "[Date].[Calendar Year].&[2001]";
    part.Annotations.Add("LastOrderDateKey", "184");

    part = mg.Partitions.FindByName("Internet_Sales_549");
    if ( part != null)
        part.Drop();

    part = mg.Partitions.Add("Internet_Sales_549");
    part.StorageMode = StorageMode.Molap;
    part.Source = new QueryBinding(db.DataSources[0].ID, "SELECT * FROM
[dbo].[FactInternetSales] WHERE OrderDateKey > '184' AND OrderDateKey <= '549'");
    part.Slice = "[Date].[Calendar Year].&[2002]";
    part.Annotations.Add("LastOrderDateKey", "549");

    part = mg.Partitions.FindByName("Internet_Sales_914");
```

```

        if ( part != null)

            part.Drop();

        part = mg.Partitions.Add("Internet_Sales_914");

        part.StorageMode = StorageMode.Molap;

        part.Source = new QueryBinding(db.DataSources[0].ID, "SELECT * FROM
[dbo].[FactInternetSales] WHERE OrderDateKey > '549' AND OrderDateKey <= '914'");

        part.Slice = "[Date].[Calendar Year].[2003]";

        part.Annotations.Add("LastOrderDateKey", "914");

    }

```

Processing a Partition

Processing a partition is as simple as using the Process method of the **T:Microsoft.AnalysisServices.Partition** object.

For more information about processing options, see [Processing Objects \(XMLA\)](#) and [Processing Analysis Services Objects](#).

The following code sample does a full process in all partitions of a specified measure group.

```

static void FullProcessAllPartitions(MeasureGroup mg)
{
    foreach (Partition part in mg.Partitions)
        part.Process(ProcessType.ProcessFull);
}

```

Merging Partitions

Merging partitions means performing any operation that results in two or more partitions becoming one partition.

Merging partitions is a method of the **T:Microsoft.AnalysisServices.Partition** object. This command merges the data of one or more source partitions into a target partition and deletes the source partitions.

Partitions can be merged only if they meet all the following criteria:

- Partitions are in the same measure group.
- Partitions are stored in the same mode (MOLAP, HOLAP, and ROLAP).
- Partitions reside on the same server; remote partitions can be merged if on the same server.

Unlike previous versions, in Microsoft SQL Server Analysis Services it is not necessary that all source partitions have identical aggregations design.

The resulting set of aggregations for the target partition is the same set of aggregations as of the state before running merge command.

The following code sample merges all partitions of a specified measure group. The partitions are merged into the first partition of the measure group.

```

static void MergeAllPartitions (MeasureGroup mg)
{
    if (mg.Partitions.Count > 1)
    {
        Partition[] partArray = new Partition[mg.Partitions.Count - 1];
        for (int i = 1; i < mg.Partitions.Count; i++)
            partArray[i - 1] = mg.Partitions[i];
        mg.Partitions[0].Merge (partArray);
        //To have last changes in the server reflected in AMO
        mg.Refresh();
    }
}

```

Aggregation Objects

To design an aggregation and apply it to one or more partitions, you program **T:Microsoft.AnalysisServices.Aggregation** object.

Creating and Dropping Aggregations

Aggregations can easily be created and assigned to measure groups or to partitions by using the `DesignAggregations` method from the **T:Microsoft.AnalysisServices.AggregationDesign** object. The **T:Microsoft.AnalysisServices.AggregationDesign** object is a separate object from partition, the **T:Microsoft.AnalysisServices.AggregationDesign** object is contained in the **T:Microsoft.AnalysisServices.MeasureGroup** object. Aggregations can be designed up to specified level of optimization (0 to 100) or up to specified level of storage (bytes). Multiple partitions can use the same aggregation design.

The following code sample creates aggregations for all partitions of a supplied measure group. Any existing aggregations in partitions are dropped.

```

static public String DesignAggregationsOnPartitions (MeasureGroup mg, double
optimizationWanted, double maxStorageBytes)
{
    double optimization = 0;
    double storage = 0;
    long aggCount = 0;
    bool finished = false;
    AggregationDesign ad = null;
    String aggDesignName;
    String AggregationsDesigned = "";
    aggDesignName = mg.AggregationPrefix + "_" + mg.Name;
    ad = mg.AggregationDesigns.Add();
}

```

```

        ad.Name = aggDesignName;

        ad.InitializeDesign();

        while ((!finished) && (optimization < optimizationWanted) && (storage <
maxStorageBytes))
        {
            ad.DesignAggregations(out optimization, out storage, out aggCount, out finished);
        }

        ad.FinalizeDesign();

        foreach (Partition part in mg.Partitions)
        {
            part.AggregationDesignID = ad.ID;

            AggregationsDesigned += aggDesignName + " = " + aggCount.ToString() + "
aggregations designed\r\n\tOptimization: " + optimization.ToString() + "/" +
optimizationWanted.ToString() + "\r\n\tStorage: " + storage.ToString() + "/" +
maxStorageBytes.ToString() + " ]\n\r";
        }

        return AggregationsDesigned;
    }
}

```

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[AMO OLAP Classes](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

[Processing Analysis Services Objects](#)

[AdventureWorks Sample Data Warehouse](#)

Programming AMO OLAP Advanced Objects

This topic explains the Analysis Management Objects (AMO) programming details of OLAP advanced objects. This topic contains the following sections:

- Action Objects
- Kpi Objects
- Perspective Objects
- ProactiveCaching Objects
- Translation Objects

Action Objects

Action classes are used to create an active response when browsing certain areas of the cube. Action objects can be defined by using AMO, but are used from the client application that browses the data. Actions can be of different types and they have to be created according to their type. Actions can be:

- Drillthrough actions, which return the set of rows that represents the underlying data of the selected cells of the cube where the action occurs.
- Reporting actions, which return a report from Reporting Services that is associated with the selected section of the cube where the action occurs.
- Standard actions, which return the action element (URL, HTML, DataSet, RowSet, and other elements) that is associated with the selected section of the cube where the action occurs.

Creating an action object requires the following steps:

1. Create the derived action object and populate basic attributes.

The following are the basic attributes: type of action, target type or section of the cube, target or specific area of the cube where the action is available, caption and where the caption is an MDX expression.

2. Populate the specific attributes of the action type.

Attributes are different for the three types of actions, see the code sample that follows for parameters.

3. Add the action to the cubes collection and update the cube. The action is not an updatable object.

Testing the action requires a different program application. You can test your action in SQL Server Data Tools (SSDT). First, you must install Reporting Services samples, see [Processing Analysis Services Objects](#).

The following sample code replicates three different actions from the Adventure Works Analysis Services Project sample. For more information about installing the samples, see [AdventureWorks2012 Sample Databases](#). You can differentiate the actions because the ones that you introduce by using the following sample, start with "My".

```
static public void CreateActions(Cube cube)
{
    #region Adding a drillthrough action

    // Verify That action exists and drop it
    if (cube.Actions.ContainsName("My Reseller Details"))
        cube.Actions.Remove("My Drillthrough Action", true);

    //Create a Drillthrough action
    DrillThroughAction dtaction = new DrillThroughAction("My Reseller Details", "My
Drillthrough Action");
```

```

//Define the Action
dtaction.Type = ActionType.DrillThrough;
dtaction.TargetType = ActionTargetType.Cells;
dtaction.Target = "MeasureGroupMeasures(\"Reseller Sales\")";
dtaction.Caption = "My Drillthrough...";
dtaction.CaptionIsMdx = false;

#region create drillthrough action specifics
//Adding Drillthrough columns
//Adding Measure columns to the drillthrough
MeasureGroup mg = cube.MeasureGroups.FindByName("Reseller Sales");
MeasureBinding mb1 = new MeasureBinding();
mb1.MeasureID = mg.Measures.FindByName("Reseller Sales Amount").ID;
dtaction.Columns.Add(mb1);

MeasureBinding mb2 = new MeasureBinding();
mb2.MeasureID = mg.Measures.FindByName("Reseller Order Quantity").ID;
dtaction.Columns.Add(mb2);

MeasureBinding mb3 = new MeasureBinding();
mb3.MeasureID = mg.Measures.FindByName("Reseller Unit Price").ID;
dtaction.Columns.Add(mb3);

//Adding Dimension Columns to the drillthrough
CubeAttributeBinding cb1 = new CubeAttributeBinding();
cb1.CubeID = cube.ID;
cb1.CubeDimensionID = cube.Dimensions.FindByName("Reseller").ID;
cb1.AttributeID = "Reseller Name";
cb1.Type = AttributeBindingType.All;
dtaction.Columns.Add(cb1);

CubeAttributeBinding cb2 = new CubeAttributeBinding();

```

```

cb2.CubeID = cube.ID;

cb2.CubeDimensionID = cube.Dimensions.FindByName("Product").ID;

cb2.AttributeID = "Product Name";

cb2.Type = AttributeBindingType.All;

dtaction.Columns.Add(cb2);

#endregion

//Add the defined action to the cube

cube.Actions.Add(dtaction);

#endregion

#region Adding a Standard action

// Verify That action exists and drop it

if (cube.Actions.ContainsName("My City Map"))

    cube.Actions.Remove("My Action", true);

//Create a Drillthrough action

StandardAction stdaction = new StandardAction("My City Map", "My Action");

//Define the Action

stdaction.Type = ActionType.Url;

stdaction.TargetType = ActionTargetType.AttributeMembers;

stdaction.Target = "[Geography].[City]";

stdaction.Caption = "\"My View Map for \" +
[Geography].[City].CurrentMember.Member_Caption + \"...\";

stdaction.CaptionIsMdx = true;

#region create standard action specifics

stdaction.Expression = "\"http://maps.msn.com/home.aspx?plce1=\" + \" +
    \"[Geography].[City].CurrentMember.Name + \",\" + \" +
    \"[Geography].[State-Province].CurrentMember.Name + \",\" + \" +
    \"[Geography].[Country].CurrentMember.Name + \" +
    \"\"&regnl=\" + \" +
    \"Case \" +

```



```

        "When [Geography].[Country].CurrentMember Is " +
            "[Geography].[Country].&[Australia] " +
            "Then \"3\" " +
        "When [Geography].[Country].CurrentMember Is " +
            "[Geography].[Country].&[Canada] " +
            "Or [Geography].[Country].CurrentMember Is " +
            "[Geography].[Country].&[United States] " +
            "Then \"0\" " +
            "Else \"1\" " +
        "End ";
#endregion

//Add the defined action to the cube
cube.Actions.Add(stdaction);

#endregion

#region Adding a Reporting action
// Verify That action exists and drop it
if (cube.Actions.ContainsName("My Sales Reason Comparisons"))
    cube.Actions.Remove("My Report Action", true);

//Create a Report action
ReportAction rsaction = new ReportAction("My Sales Reason Comparisonsp", "My Report
Action");

//Define the Action
rsaction.Type = ActionType.Report;
rsaction.TargetType = ActionTargetType.AttributeMembers;
rsaction.Target = "[Product].[Category]";
rsaction.Caption = "\"My Sales Reason Comparisons for \" +
[Product].[Category].CurrentMember.Member_Caption + \"...\";
rsaction.CaptionIsMdx = true;

```

```

        #region create Report action specifics

        rsaction.ReportServer = "MyRSSamplesServer";

        rsaction.Path = "ReportServer?/AdventureWorks Sample Reports/Sales Reason
Comparisons";

        rsaction.ReportParameters.Add("ProductCategory", "UrlEscapeFragment(
[Product].[Category].CurrentMember.UniqueName )");

        rsaction.ReportFormatParameters.Add("rs:Command", "Render");

        rsaction.ReportFormatParameters.Add("rs:Renderer", "HTML5");

        #endregion

        //Add the defined action to the cube

        cube.Actions.Add(rsaction);

        #endregion

    }

```

Kpi Objects

A key performance indicator (KPI) is a collection of calculations that are associated with a measure group in a cube and are used to evaluate business success.

T:Microsoft.AnalysisServices.Kpi objects can be defined by AMO, but are used from the client application that browses the data.

Creating a **T:Microsoft.AnalysisServices.Kpi** object requires the following steps:

1. Create the **T:Microsoft.AnalysisServices.Kpi** object and populate the basic attributes.
The following is a list of basic attributes: Description, Display Folder, Associated Measure Group, and Value. Display Folder tells the client application where the KPI should be located for the end-user to find it. The Associated Measure Group indicates the measure group where all MDX calculations should be referred. Value shows the actual value of the performance indicator as an MDX expression.
2. Define KPI Indicators: Goal, Status, and Trend.
Indicators are MDX expressions that should evaluate between -1 to 1, but is the browsing application which defines the range of values for the indicators.
3. When you browse KPIs in SQL Server Data Tools (SSDT), values less than -1 are treated as -1, and values larger than 1 are treated as 1.
4. Define graphic images.
Graphic images are string values, used as reference in the client application to identify the correct set of images to display. The graphic image string also defines the behavior of the display function. Usually the range is split in an odd number of states, from bad to good, and to each state an image, from the set, is assigned.

If you use SQL Server Data Tools (SSDT) to browse your KPIs, then depending on names, the indicator range is split into either three states or five states. In addition, there are names where the range is inverted, that is -1 is 'Good' and 1 is 'Bad'. In SQL Server Data Tools (SSDT), three states within the range are as follows:

- Bad = -1 to -0.5
- OK = -0.4999 to -0.4999
- Good = 0.50 to 1

In SQL Server Data Tools (SSDT), five states within the range are as follows:

- Bad = -1 to -0.75
- Risk = -0.7499 to -0.25
- OK = -0.2499 to 0.2499
- Raising = 0.25 to 0.7499
- Good = 0.75 to 1

The following table lists the Usage, Name, and the number of states associated with the image.

Image usage	Image Name	Number of States
Status	Shapes	3
Status	Traffic Light	3
Status	Road Signs	3
Status	Gauge	3
Status	Reversed Gauge	5
Status	Thermometer	3
Status	Cylinder	3
Status	Faces	3
Status	Variance arrow	3
Trend	Standard Arrow	3
Trend	Status Arrow	3
Trend	Reversed status arrow	5
Trend	Faces	3

1. Add the KPI to the cube collection and update the cube, because the KPI is not an updatable object.

Testing the KPI requires a different program application. You can test your KPI in SQL Server Data Tools (SSDT).

The following sample code creates a KPI in the "Financial Perspective/Grow Revenue" folder for the Adventure Works cube that is included in the Adventure Works Analysis Services Project sample. For more information about installing the samples, see [AdventureWorks2012 Sample Databases](#).

```
static public void CreateKPIs(Cube cube)
{
    Kpi kpi = cube.Kpis.Add("My Internet Revenue", "My Internet Revenue");
    kpi.Description = "(My) Revenue achieved through direct sales via Internet";
    kpi.DisplayFolder = "\\Financial Perspective\\Grow Revenue";
    kpi.AssociatedMeasureGroupID = "Internet Sales";
    kpi.Value = "[Measures].[Internet Sales Amount]";
    #region Goal
    kpi.Goal = "Case" +
        "    When IsEmpty" +
        "        (" +
        "            ParallelPeriod" +
        "            (" +
        "                [Date].[Fiscal Time].[Fiscal Year]," +
        "                1," +
        "                [Date].[Fiscal Time].CurrentMember" +
        "            )" +
        "        )" +
        "    Then [Measures].[Internet Sales Amount]" +
        "    Else 1.10 *" +
        "        (" +
        "            [Measures].[Internet Sales Amount]," +
        "            ParallelPeriod" +
        "            (" +
        "                [Date].[Fiscal Time].[Fiscal Year]," +
        "                1," +
        "                [Date].[Fiscal Time].CurrentMember" +
        "            )" +
        "        )
```

```

        "          ) " +
        " End";

#endregion

#region Status

kpi.Status = "Case" +

        "    When KpiValue( \"Internet Revenue\" ) / KpiGoal ( \"Internet
Revenue\" ) >= .95 " +

        "    Then 1 " +

        "    When KpiValue( \"Internet Revenue\" ) / KpiGoal ( \"Internet
Revenue\" ) < .95 " +

        "        And " +

        "        KpiValue( \"Internet Revenue\" ) / KpiGoal ( \"Internet
Revenue\" ) >= .85 " +

        "    Then 0 " +

        "    Else -1 " +

        "End";

#endregion

#region Trend

kpi.Trend = "Case " +

        "    When VBA!Abs " +

        "        ( " +

        "            KpiValue( \"Internet Revenue\" ) - " +

        "            ( " +

        "                KpiValue ( \"Internet Revenue\" ), " +

        "                ParallelPeriod " +

        "            ( " +

        "                [Date].[Fiscal Time].[Fiscal Year], " +

        "                1, " +

        "                [Date].[Fiscal Time].CurrentMember " +

        "            ) " +

        "        ) / " +

        "        ( " +

        "            KpiValue ( \"Internet Revenue\" ), " +

        "            ParallelPeriod " +

```

```

"          ( " +
"          [Date].[Fiscal Time].[Fiscal Year], " +
"          1, " +
"          [Date].[Fiscal Time].CurrentMember " +
"          ) " +
"          ) " +
"          ) <=.02 " +
"      Then 0 " +
"      When KpiValue( \"Internet Revenue\" ) - " +
"      ( " +
"      KpiValue ( \"Internet Revenue\" ), " +
"      ParallelPeriod " +
"      ( " +
"      [Date].[Fiscal Time].[Fiscal Year], " +
"      1, " +
"      [Date].[Fiscal Time].CurrentMember " +
"      ) " +
"      ) / " +
"      ( " +
"      KpiValue ( \"Internet Revenue\" ), " +
"      ParallelPeriod " +
"      ( " +
"      [Date].[Fiscal Time].[Fiscal Year], " +
"      1, " +
"      [Date].[Fiscal Time].CurrentMember " +
"      ) " +
"      ) >.02 " +
"      Then 1 " +
"      Else -1 " +
"End";

#endregion

kpi.TrendGraphic = "Standard Arrow";
kpi.StatusGraphic = "Cylinder";

```

```
};
```

Perspective Objects

T:Microsoft.AnalysisServices.Perspective objects can be defined by AMO, but are used from the client application that browses the data.

Creating a **T:Microsoft.AnalysisServices.Perspective** object requires the following steps:

1. Create the **T:Microsoft.AnalysisServices.Perspective** object and populate the basic attributes. The following is a list of basic attributes: Name, Default Measure, Description, and annotations.
2. Add all objects from the parent cube that should be seen by end user. Add cube dimensions (attributes and hierarchies), measure groups (measure and measure group), actions, KPIs, and calculations.
3. Add the perspective to the cube collection and update the cube, because perspective is not an updatable object.

Testing the perspective requires a different program application. You can test your perspective in SQL Server Data Tools (SSDT).

The following code sample creates a perspective named "Direct Sales" for the supplied cube.

```
static public void CreatePerspectives(Cube cube)
{
    Perspective perspective = cube.Perspectives.Add("Direct Sales", "Direct Sales");
    CubeDimension dim1 = cube.Dimensions.GetByName("Date");
    PerspectiveDimension pdim1 = perspective.Dimensions.Add(dim1.DimensionID);
    pdim1.Attributes.Add("Date");
    pdim1.Attributes.Add("Calendar Year");
    pdim1.Attributes.Add("Fiscal Year");
    pdim1.Attributes.Add("Calendar Quarter");
    pdim1.Attributes.Add("Fiscal Quarter");
    pdim1.Attributes.Add("Calendar Month Number");
    pdim1.Attributes.Add("Fiscal Month Number");
    pdim1.Hierarchies.Add("Calendar Time");
    pdim1.Hierarchies.Add("Fiscal Time");

    CubeDimension dim2 = cube.Dimensions.GetByName("Product");
    PerspectiveDimension pdim2 = perspective.Dimensions.Add(dim2.DimensionID);
    pdim2.Attributes.Add("Product Name");
    pdim2.Attributes.Add("Product Line");
}
```

```

    pdim2.Attributes.Add("Model Name");
    pdim2.Attributes.Add("List Price");
    pdim2.Attributes.Add("Size");
    pdim2.Attributes.Add("Weight");
    pdim2.Hierarchies.Add("Product Model Categories");
    pdim2.Hierarchies.Add("Product Categories");

    PerspectiveMeasureGroup pmg = perspective.MeasureGroups.Add("Internet Sales");
    pmg.Measures.Add("Internet Sales Amount");
    pmg.Measures.Add("Internet Order Quantity");
    pmg.Measures.Add("Internet Unit Price");

    pmg = perspective.MeasureGroups.Add("Reseller Sales");
    pmg.Measures.Add("Reseler Sales Amount");
    pmg.Measures.Add("Reseller Order Quantity");
    pmg.Measures.Add("Reseller Unit Price");

    PerspectiveAction pact = perspective.Actions.Add("Drilllthrough Action");

    PerspectiveKpi pkpi = perspective.Kpis.Add("Internet Revenue");
    Cube.Update();
}

```

ProactiveCaching Objects

T:Microsoft.AnalysisServices.ProactiveCaching objects can be defined by AMO.

Creating a **T:Microsoft.AnalysisServices.ProactiveCaching** object requires the following steps:

1. Create the **T:Microsoft.AnalysisServices.ProactiveCaching** object.

There are no basic attributes to define.

2. Add cache specifications.

Specification	Description
AggregationStorage	The type of storage for aggregations. Applies to partition only. On dimension it must be Regular .

Specification	Description
SilenceInterval	Minimum amount of time the cache exists before the MOLAP imaging starts.
Latency	The amount of time between the earliest notification and the moment when the MOLAP images are destroyed.
SilenceOverrideInterval	The time after an initial notification after which the MOLAP imaging kicks in unconditionally.
ForceRebuildInterval	The time (starting after a fresh MOLAP image is dropped) after which MOLAP imaging starts unconditionally (no notifications).
OnlineMode	When the MOLAP image is available. Can be either Immediate or OnCacheComplete .

1. Add the **T:Microsoft.AnalysisServices.ProactiveCaching** object to the parent collection. You will need to update the parent, because **T:Microsoft.AnalysisServices.ProactiveCaching** is not an updatable object.

The following code sample creates a **T:Microsoft.AnalysisServices.ProactiveCaching** object in all partitions from the Internet Sales measure group in the Adventure Works cube in a specified database.

```
static public void SetProactiveCachingSettings(Database db)
{
    ProactiveCaching pc;

    if (db.Cubes.ContainsName("Adventure Works") && db.Cubes.FindByName("Adventure
Works").MeasureGroups.ContainsName("Internet Sales"))
    {
        ProactiveCachingTablesBinding pctb;
        TableNotification tn;

        MeasureGroup mg = db.Cubes.FindByName("Adventure
Works").MeasureGroups.FindByName("Internet Sales");

        foreach(Partition part in mg.Partitions)
        {
```

```

        pc = new ProactiveCaching();

        pc.AggregationStorage = ProactiveCachingAggregationStorage.MolapOnly;

        pc.SilenceInterval = TimeSpan.FromSeconds(10);

        pc.Latency = TimeSpan.FromSeconds(-1);

        pc.SilenceOverrideInterval = TimeSpan.FromMinutes(10);

        pc.ForceRebuildInterval = TimeSpan.FromSeconds(-1);

        pc.Enabled = true;

        pc.OnlineMode = ProactiveCachingOnlineMode.OnCacheComplete;

        pctb = new ProactiveCachingTablesBinding();

        pctb.NotificationTechnique = NotificationTechnique.Server;

        tn = new TableNotification("[FactInternetSales]", "dbo");

        pctb.TableNotifications.Add( tn);

        pc.Source = pctb;

        part.ProactiveCaching = pc;

        part.Update();

    }

}

```

Translation Objects

Translation objects can be defined by AMO, but are used from the client application that browses the data. Translation objects are simple objects to code. Translations for object captions are provided by pairs of Locale Identifier and Translated Caption. For any caption, multiple translations can be enabled. Translations can be provided for most Analysis Services objects, such as dimensions, attributes, hierarchies, cubes, measure groups, measures, and others.

The following code sample provides a Spanish translation for the name of the attribute Product Name.

```

static public void CreateTranslations(Database db)
{
    //Spanish Tranlations for Product Category in Product Dimension

    Dimension dim = db.Dimensions["Product"];

    DimensionAttribute atr = dim.Attributes["Product Name"];

    Translation tran = atr.Translations.Add(3082);

    tran.Caption = "Nombre Producto";
}

```

```
dim.Update(UpdateOptions.ExpandFull);
```

```
}
```

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[AMO OLAP Classes](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

[Processing Analysis Services Objects](#)

Programming AMO Data Mining Objects

Programming data mining objects by using AMO is simple and straightforward. The first step is to create the data structure model to support the mining project. Then you create the data mining model that supports the mining algorithm you want to use in order to predict or to find the unseen relationships underlying your data. With your mining project created (including structure and algorithms), you can then process the mining models to obtain the trained models that you will use later when querying and predicting from the client application.

One thing to remember is that AMO is not for querying; AMO is for managing and administering your mining structures and models. To query your data, use [Database Objects \(Analysis Services - Multidimensional Data\)](#).

This topic contains the following sections:

- MiningStructure Objects
- MiningModel Objects

MiningStructure Objects

A mining structure is the definition of the data structure that is used to create all mining models. A mining structure contains a binding to a data source view that is defined in the database, and contains definitions for all columns participating in the mining models. A mining structure can have more than one mining model.

Creating a **T:Microsoft.AnalysisServices.MiningStructure** object requires the following steps:

1. Create the **T:Microsoft.AnalysisServices.MiningStructure** object and populate the basic attributes. Basic attributes include object name, object ID (internal identification), and data source binding.
2. Create columns for the model. Columns can be either scalar or table definitions. Each column needs a name and internal ID, a type, a content definition, and a binding.
3. Update the **T:Microsoft.AnalysisServices.MiningStructure** object to the server, by using the Update method of the object.

Mining structures can be processed, and when they are processed, the children mining models are processed or retrained.

The following sample code creates a mining structure to forecast sales in a time series. Before running the sample code, make sure that the database db, passed as parameter for CreateSalesForecastingMiningStructure, contains in db.DataSourceViews[0] a reference to the view dbo.vTimeSeries in the Adventure Works DW Multidimensional 2012 sample database.

```
public static MiningStructure CreateSalesForecastingMiningStructure(Database db)
{
    MiningStructure ms = db.MiningStructures.FindByName("Forecasting Sales Structure");
    if (ms != null)
        ms.Drop();

    ms = db.MiningStructures.Add("Forecasting Sales Structure", "Forecasting Sales Structure");
    ms.Source = new DataSourceViewBinding(db.DataSourceViews[0].ID);

    ScalarMiningStructureColumn amount = ms.Columns.Add("Amount", "Amount");
    amount.Type = MiningStructureColumnTypes.Double;
    amount.Content = MiningStructureColumnContents.Continuous;
    amount.KeyColumns.Add("vTimeSeries", "Amount", OleDbType.Currency);

    ScalarMiningStructureColumn modelRegion = ms.Columns.Add("Model Region", "Model Region");
    modelRegion.IsKey = true;
    modelRegion.Type = MiningStructureColumnTypes.Text;
    modelRegion.Content = MiningStructureColumnContents.Key;
    modelRegion.KeyColumns.Add("vTimeSeries", "ModelRegion", OleDbType.WChar, 56);

    ScalarMiningStructureColumn qty = ms.Columns.Add("Quantity", "Quantity");
    qty.Type = MiningStructureColumnTypes.Long;
    qty.Content = MiningStructureColumnContents.Continuous;
    qty.KeyColumns.Add("vTimeSeries", "Quantity", OleDbType.Integer);

    ScalarMiningStructureColumn timeIndex = ms.Columns.Add("TimeIndex", "TimeIndex");
    timeIndex.IsKey = true;
    timeIndex.Type = MiningStructureColumnTypes.Long;
    timeIndex.Content = MiningStructureColumnContents.KeyTime;
}
```

```

timeIndex.KeyColumns.Add("vTimeSeries", "TimeIndex", OleDbType.Integer);

ms.Update();

return ms;

}

```

MiningModel Objects

A mining model is a repository for all columns and column definitions that will be used in a mining algorithm.

Creating a **T:Microsoft.AnalysisServices.MiningModel** object requires the following steps:

1. Create the **T:Microsoft.AnalysisServices.MiningModel** object and populate the basic attributes.
Basic attributes include object name, object ID (internal identification), and mining algorithm specification.
2. Add the columns of the mining model. One of the columns must be defined as the case key.
3. Update the **T:Microsoft.AnalysisServices.MiningModel** object to the server, by using the Update method of the object.

T:Microsoft.AnalysisServices.MiningModel objects can be processed independently of other models in the parent **T:Microsoft.AnalysisServices.MiningStructure**.

The following sample code creates a Microsoft Time Series forecasting model based on the "Forecasting Sales Structure" mining structure:

```

public static MiningModel CreateSalesForecastingMiningModel(MiningStructure ms)
{
    if (ms.MiningModels.ContainsName("Sales Forecasting Model"))
    {
        ms.MiningModels["Sales Forecasting Model"].Drop();
    }

    MiningModel mm = ms.CreateMiningModel(true, "Sales Forecasting Model");
    mm.Algorithm = MiningModelAlgorithms.MicrosoftTimeSeries;
    mm.AlgorithmParameters.Add("PERIODICITY_HINT", "{12}");

    MiningModelColumn amount = new MiningModelColumn();
    amount.SourceColumnID = "Amount";
    amount.Usage = MiningModelColumnUsages.Predict;

    MiningModelColumn modelRegion = new MiningModelColumn();

```

```

modelRegion.SourceColumnID = "Model Region";

modelRegion.Usage = MiningModelColumnUsages.Key;


MiningModelColumn qty = new MiningModelColumn();

qty.SourceColumnID = "Quantity";

qty.Usage = MiningModelColumnUsages.Predict;


MiningModelColumn ti = new MiningModelColumn();

ti.SourceColumnID = "TimeIndex";

ti.Usage = MiningModelColumnUsages.Key;


mm.Update();

mm.Process(ProcessType.ProcessFull);

return mm;

}

```

See Also

[AMO Fundamental Classes](#)

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[AMO DataMining Classes](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

Programming AMO Security Objects

In Microsoft SQL Server Analysis Services, programming security objects or running applications that use AMO security objects requires being a member of the Server Administrator group or the Database Administrator group. Server Administrator and Database Administrator are an access levels supplied by SQL Server Analysis Services.

In Analysis Services the user access to any object is obtained through the combination of Roles and Permissions assigned to that object. For more information, see [Database Objects \(Analysis Services - Multidimensional Data\)](#).

Role and Permission Objects

Server roles contain one and only one role in the collection, the Administrators role. New roles cannot be added to the server roles collection. Membership in the Administrators role permits complete access to every object in the server

T:Microsoft.AnalysisServices.Role objects are created at database level. Role maintenance requires only adding or removing members to or from the role, and adding or dropping roles to the **T:Microsoft.AnalysisServices.Database** object. A role cannot be dropped if there is any **T:Microsoft.AnalysisServices.Permission** object associated with the role. To drop a role, all **T:Microsoft.AnalysisServices.Permission** objects in the **T:Microsoft.AnalysisServices.Database** objects must be searched, and the **T:Microsoft.AnalysisServices.Role** removed from permissions, before the **T:Microsoft.AnalysisServices.Role** can be dropped from the **T:Microsoft.AnalysisServices.Database**.

Permissions define the enabled actions on the object where the permission is supplied. Permissions can be supplied to the following objects: **T:Microsoft.AnalysisServices.Database**, **T:Microsoft.AnalysisServices.DataSource**, **T:Microsoft.AnalysisServices.Dimension**, **T:Microsoft.AnalysisServices.Cube**, **T:Microsoft.AnalysisServices.MiningStructure**, and **T:Microsoft.AnalysisServices.MiningModel**. Permission maintenance involves granting or revoking enabled access by the corresponding access property. For each enabled access, there is a property that can be set to the desired level of access. Access can be defined for the following operations: Process, ReadDefinition, Read, Write, and Administer. Administer access is only defined on the **T:Microsoft.AnalysisServices.Database** object. The database administrator security level is obtained when the role is granted with the Administer database permission.

The following sample creates four roles: Database Administrators, Processors, Writers, and Readers.

Database Administrators can administer the supplied database.

Processors can process all objects in a database and verify results. To verify results, read access to the database object must be explicitly enabled to the supplied cube, because read permission does not apply to children objects.

Writers can read and write to the supplied cube, and cell access is limited to the 'United States' in the customer dimension.

Readers can read on the supplied cube, and cell access is limited to the 'United States' in the customer dimension.

```
static public void CreateRolesAndPermissions(Database db, Cube cube)
{
    Role role;

    DatabasePermission dbperm;

    CubePermission cubeperm;

    #region Create the Database Administrators role

    // Create the Database Administrators role.
    role = db.Roles.Add("Database Administrators");
```

```

role.Members.Add(new RoleMember("")); // e.g. domain\user

role.Update();

// Assign administrative permissions to this role.
// Members of this role can perform any operation within the database.

dbperm = db.DatabasePermissions.Add(role.ID);

dbperm.Administer = true;

dbperm.Update();

#endregion

#region Create the Processors role

// Create the Processors role.

role = db.Roles.Add("Processors");

role.Members.Add(new RoleMember("")); // e.g. myDomain\johndoe

role.Update();

// Assign Read and Process permissions to this role.
// Members of this role can process objects in the database and query them to verify
results.

// Process permission applies to all contained objects, i.e. all dimensions and
cubes.

// Read permission does not apply to contained objects, so we must assign the
permission explicitly on the cubes.

dbperm = db.DatabasePermissions.Add(role.ID);

dbperm.Read = ReadAccess.Allowed;

dbperm.Process = true;

dbperm.Update();

cubeperm = cube.CubePermissions.Add(role.ID);

cubeperm.Read = ReadAccess.Allowed;

cubeperm.Update();

```



```

#endregion

#region Create the Writers role

// Create the Writers role.
role = db.Roles.Add("Writers");
role.Members.Add(new RoleMember("")); // e.g. redmond\johndoe
role.Update();

// Assign Read and Write permissions to this role.
// Members of this role can discover, query and writeback to the Adventure Works
cube.

// However cell access and writeback is restricted to the United States (in the
Customer dimension).

dbperm = db.DatabasePermissions.Add(role.ID);
dbperm.Read = ReadAccess.Allowed;
dbperm.Update();

cubeperm = cube.CubePermissions.Add(role.ID);
cubeperm.Read = ReadAccess.Allowed;
cubeperm.Write = WriteAccess.Allowed;
cubeperm.CellPermissions.Add(new CellPermission(CellPermissionAccess.Read,
"[Customer].[Country-Region].CurrentMember is [Customer].[Country-Region].[Country-
Region].&[United States]"));
cubeperm.CellPermissions.Add(new CellPermission(CellPermissionAccess.ReadWrite,
"[Customer].[Country-Region].CurrentMember is [Customer].[Country-Region].[Country-
Region].&[United States]"));
cubeperm.Update();

#endregion

#region Create the Readers role

// Create the Readers role.

```

```

role = db.Roles.Add("Readers");

role.Members.Add(new RoleMember("")); // e.g. redmond\johndoe

role.Update();


// Assign Read permissions to this role.

// Members of this role can discover and query the Adventure Works cube.

// However the Customer dimension is restricted to the United States.

dbperm = db.DatabasePermissions.Add(role.ID);

dbperm.Read = ReadAccess.Allowed;

dbperm.Update();


cubeperm = cube.CubePermissions.Add(role.ID);

cubeperm.Read = ReadAccess.Allowed;

Dimension dim = db.Dimensions.GetByName("Customer");

DimensionAttribute attr = dim.Attributes.GetByName("Country-Region");

CubeDimensionPermission cubedimperm = cubeperm.DimensionPermissions.Add(dim.ID);

cubedimperm.Read = ReadAccess.Allowed;

AttributePermission attrperm = cubedimperm.AttributePermissions.Add(attr.ID);

attrperm.AllowedSet = "{ [Customer].[Country-Region].[Country-Region].&[United
States]}";

cubeperm.Update();


#endregion

}

```

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[Programming AMO Security objects](#)

[Permissions and Access Rights \(SSAS\)](#)

[Securing the Analysis Services Instance](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

Programming AMO Complementary Classes and Methods

This topic contains the following sections:

- Assembly Class
- Backup and Restore
- Trace Class
- CaptureLog class and CaptureXML attribute

Assembly Class

Assemblies let users extend the functionality of Microsoft SQL Server Analysis Services by adding new stored procedures or Multidimensional Expressions (MDX) functions. For more information, see [Processing Analysis Services Objects](#).

Adding and dropping assemblies is simple and can be performed online. You must be a database administrator to add an assembly to the database or a server administrator to add the assembly to the server object.

The following sample adds an assembly to the provided database and assigns the service account to run the assembly. If the assembly exists in the database, the assembly is dropped before trying to add it.

```
static public void CreateStoredProcedures(Database db)
{
    ClrAssembly clrAssembly;

    // Verify That assembly exist in database and drop it
    if (db.Assemblies.ContainsName("StoredProcedures"))
    {
        clrAssembly = db.Assemblies.FindByName("StoredProcedures");
        clrAssembly.Drop();
    }

    // Create the CLR assembly
    clrAssembly = db.Assemblies.Add("StoredProcedures");
    clrAssembly.ImpersonationInfo = new ImpersonationInfo(
        ImpersonationMode.ImpersonateServiceAccount);
    clrAssembly.PermissionSet = PermissionSet.Unrestricted;

    // Load the assembly files
    clrAssembly.LoadFiles(Environment.CurrentDirectory
```

```

        + @"\\StoredProcedures2.dll", false);

        clrAssembly.Update();
    }
}

```

Backup and Restore Methods

The Backup and Restore methods let administrators back up databases and restore them.

The following sample creates backups for all databases in the specified server. If a backup file already exists, then it is overwritten. Backup files are saved in the BackUp folder in the Analysis Services Data folder.

```

static public void BackUpAllDatabases(Server svr)
{
    string fileName;

    if ((svr != null) && ( svr.Connected))
        foreach (Database db in svr.Databases)
        {
            fileName = db.Name + "_" + ((Int64)(DateTime.Today.Year * 10000 +
DateTime.Today.Month * 100 + DateTime.Today.Day)).ToString() + ".abf";

            db.Backup(fileName, true);
        }
}

```

The following sample restores the "Adventure Works" backup from the previous sample. If the "My Adventure WorksDW" database already exists, then the database is overwritten.

```

static public void RestoreAdventureWorks(Server svr)
{
    svr.Restore("Adventure Works DW_20051025.abf", "My Adventure WorksDW", true);
}

```

Trace Class

Monitoring the server activity requires using two kinds of traces: Session Traces and Server Traces. Tracing the server can tell you how your current task is performing on the server (Session Traces) or the traces can tell you about the overall activity in the server without you even being connected to the server (Server Traces).

When tracing current activity (Session Traces), the server sends notifications to the current application about the events that are occurring in the server that are caused by the application. Events are captured using event handlers in the current application. You first assign the event

handling routines to the **T:Microsoft.AnalysisServices.SessionTrace** object and then start the Session Trace.

The following sample shows how to setup a Session Trace to trace current activities. Event handler routines are located at the end of the sample and will output all trace information to the System.Console object. To generate tracing events the "Adventure Works" sample cube will be fully processed after the trace starts.

```
static public void TestSessionTraces(Server svr)
{
    // Start the default trace

    svr.SessionTrace.OnEvent
        += new TraceEventHandler(DefaultTrace_OnEvent);
    svr.SessionTrace.Stopped
        += new TraceStoppedEventHandler(DefaultTrace_Stopped);
    svr.SessionTrace.Start();

    // Process the databases
    // The trace handlers will output all progress notifications
    // to the console
    Database db = svr.Databases.FindByName("Adventure Works DW Multidimensional 2012");
    Cube cube = db.Cubes.FindByName("Adventure Works");
    cube.Process(ProcessType.ProcessFull);

    // Stop the default trace
    svr.SessionTrace.Stop();
}

static public void DefaultTrace_OnEvent(object sender, TraceEventArgs e)
{
    Console.WriteLine("{0}", e.TextData);
}

static public void DefaultTrace_Stopped(ITrace sender, TraceStoppedEventArgs e)
```

```

{
    switch (e.StopCause)
    {
        case TraceStopCause.StoppedByUser:
        case TraceStopCause.Finished:
            Console.WriteLine("Processing completed successfully");
            break;

        case TraceStopCause.StoppedByException:
            Console.WriteLine("Processing failed: {0}",
                e.Exception.Message);
            break;
    }
}

```

Server traces can be configured to log everything to a trace file and can be automatically restarted when the service restarts.

To set a server trace, you first need to define which events in the server to be monitored, and what data from the event should be saved in the trace file. For each event, you must define the data columns to be saved in the trace file.

Creating a server trace requires four steps:

1. Create the server trace object and populate basic common attributes.

LogFileSize defines the maximum size of the trace file and is defined in MegaBytes;

LogFileRollOver enables the logfile to start on a different file if LogFileSize limit is reached, when enabled the file name is appended with a sequence number; AutoRestart enables the trace to start again if the Service is restarted.

2. Create the events and the corresponding data columns.
3. Start and stop the trace as needed.

Even after the trace has been stopped, the trace exists in the server and should start again if the trace was defined as AutoRestart=**true**.

4. Drop the trace when no longer needed.

In the following sample, if the trace already exists, it is dropped and then recreated. Trace files are saved in the Log folder of Analysis Services data folders.

```

static public void TestServerTraces(Server svr)
{
    Trace trc;
    TraceEvent te;

```

```

trc = svr.Traces.FindByName("TestServerTraces");

if (trc != null)
    trc.Drop();

trc = svr.Traces.Add("TestServerTraces", "TestServerTraces");

trc.LogFileName = ("TestServerTraces_" + ((Int64)(DateTime.Now.Year * 10000 +
DateTime.Now.Month * 100 + DateTime.Now.Day)).ToString() + "_" +
    ((Int64)(DateTime.Now.Hour * 10000 + DateTime.Now.Minute * 100 +
DateTime.Now.Second)).ToString() + ".trc");

trc.LogFileSize = 100;

trc.LogFileRollover = true;

trc.AutoRestart = false;

#region Define Events to trace & data columns per event
te = trc.Events.Add(TraceEventClass.ProgressReportBegin);
te.Columns.Add(TraceColumn.TextData);
te.Columns.Add(TraceColumn.StartTime);
te.Columns.Add(TraceColumn.ObjectName);
te.Columns.Add(TraceColumn.ObjectPath);
te.Columns.Add(TraceColumn.DatabaseName);
te.Columns.Add(TraceColumn.NTCanonicalUserName);

te = trc.Events.Add(TraceEventClass.ProgressReportCurrent);
te.Columns.Add(TraceColumn.TextData);
te.Columns.Add(TraceColumn.CurrentTime);
te.Columns.Add(TraceColumn.ObjectName);
te.Columns.Add(TraceColumn.ObjectPath);
te.Columns.Add(TraceColumn.DatabaseName);

te = trc.Events.Add(TraceEventClass.ProgressReportEnd);
te.Columns.Add(TraceColumn.TextData);
te.Columns.Add(TraceColumn.StartTime);
te.Columns.Add(TraceColumn.CurrentTime);

```

```

        te.Columns.Add(TraceColumn.EndTime);
        te.Columns.Add(TraceColumn.Success);
        te.Columns.Add(TraceColumn.Error);
        te.Columns.Add(TraceColumn.ObjectName);
        te.Columns.Add(TraceColumn.ObjectPath);
        te.Columns.Add(TraceColumn.DatabaseName);
        te.Columns.Add(TraceColumn.NTCanonicalUserName);
    #endregion

    trc.Update();

    trc.Start();

    #region Process the Adventure Works Cube
    // The trace settings will output all progress notifications
    // to the trace file

    Database db = svr.Databases.FindByName("Adventure Works DW Multidimensional
2012");

    Cube cube = db.Cubes.FindByName("Adventure Works");
    cube.Process(ProcessType.ProcessFull);
    #endregion

    trc.Stop();
    trc.Drop();

}

```

CaptureLog and CaptureXml Attributes

The CaptureLog attribute enables you to create XMLA batch files from your AMO operations. CaptureLog enables you to script out server objects as databases, cubes, dimensions, mining structures, and others.

Creating a CaptureLog requires the following steps:

1. Start capturing the XMLA log by setting the server attribute CaptureXml to **true**.

This option will start saving all AMO operations to a string collection instead of sending them to the server.

2. Start AMO activity as usual, but remember that no action is being sent to the server. Activity can be any operation such as processing, creating, deleting, updating, or any other action over an object.
3. Stop capturing the XMLA log by resetting CaptureXml to **false**.
4. Review the captured XMLA, either by reviewing each of the strings in the CaptureLog string collection, or by generating a complete string with the ConcatenateCaptureLog method. ConcatenateCaptureLog enables you to generate the XMLA batch as a single transaction and to add the parallel process option to the batch.

The following sample returns a string with the batch commands to do a Full process on all dimensions and on all cubes on the [Adventure Works DW Multidimensional 2012] database.

```
static public string TestCaptureLog(Server svr)
{
    String capturedXmla = "";
    if ((svr != null) && (svr.Connected))
    {
        svr.CaptureXml = true;

        #region Actions to be captured to an XMLA file
        //No action is executed during CaptureXml = true
        Database db = svr.Databases.FindByName("Adventure Works DW Multidimensional
2012");

        foreach (Dimension dim in db.Dimensions)
            dim.Process(ProcessType.ProcessFull);
        foreach (Cube cube in db.Cubes)
            cube.Process(ProcessType.ProcessFull);
        #endregion

        svr.CaptureXml = false;

        capturedXmla = svr.ConcatenateCaptureLog(true, true);
    }

    return capturedXmla;
}
```

See Also

N:Microsoft.AnalysisServices

[Introducing AMO classes](#)

[AMO Other classes and methods](#)

[Logical Architecture \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Objects \(SSAS\)](#)

[Processing Analysis Services Objects](#)

Developing with Analysis Services Scripting Language (ASSL)

Analysis Services Scripting Language (ASSL) is an extension to XMLA that adds an object definition language and command language for creating and managing Analysis Services structures directly on the server. You can use ASSL in custom application to communicate with Analysis Services over the XMLA protocol. ASSL is made up of two parts:

- A Data Definition Language (DDL), or object definition language, defines and describes an instance of Analysis Services, as well as the databases and database objects that the instance contains.
- A command language that sends action commands, such as **Create**, **Alter**, or **Process**, to an instance of Analysis Services. This command language is discussed in the [Data Sources and Bindings \(Analysis Services - Multidimensional Data\)](#).

To view the ASSL that describes a multidimensional solution in SQL Server Data Tools, you can use the View Code command at the project level. You can also create or edit ASSL script in Management Studio using the XMLA query editor. The scripts you build can be used to manage objects or run commands on the server.

See Also

[Overview of Analysis Services Scripting Language](#)

[Objects and Object Characteristics](#)

[XML Style and Conventions](#)

[Data Sources and Bindings](#)

ASSL Objects and Object Characteristics

Objects in Analysis Services Scripting Language (ASSL) follow specific guidelines in regards to object groups, inheritance, naming, expansion, and processing.

Object Groups

All Microsoft SQL Server Analysis Services objects have an XML representation. The objects are divided into two groups:

Major objects

Major objects can be independently created, altered, and deleted. Major objects include:

- Servers
- Databases
- Dimensions
- Cubes
- Measure groups
- Partitions
- Perspectives
- Mining models
- Roles
- Commands associated with a server or database
- Data sources

Major objects have the following properties to track their history and status.

- **CreatedTimestamp**
- **LastSchemaUpdate**
- **LastProcessed** (where appropriate)



Note

The classification of an object as a major object affects how an instance of Analysis Services treats that object and how that object is handled in the object definition language. However, this classification does not guarantee that Analysis Services management and development tools will allow the independent creation, modification, or deletion of these objects.

Minor objects

Minor objects can only be created, altered, or deleted as part of creating, altering, or deleting the parent major object. Minor objects include:

- Hierarchies and levels
- Attributes
- Measures
- Mining model columns
- Commands associated with a cube
- Aggregations

Object Expansion

The **ObjectExpansion** restriction can be used to control the degree of expansion of ASSL XML returned by the server. This restriction has the options listed in the following table.

Enumeration value	Allowed for <Alter>	Description
ReferenceOnly	no	Returns only the name, ID, and timestamp for the requested object and for all contained major objects recursively.
ObjectProperties	yes	Expands the requested object and minor contained objects, but does not return major contained objects.
ExpandObject	no	Same as ObjectProperties, but also returns the name, ID, and timestamp for contained major objects.
ExpandFull	yes	Fully expands the requested object and all contained objects recursively.

This ASSL reference section describes the ExpandFull representation. All other **ObjectExpansion** levels are derived from this level.

Object Processing

ASSL includes read-only elements or properties (for example, **LastProcessed**) that can be read from the Analysis Services instance, but which are omitted when command scripts are submitted to the instance. Analysis Services ignores modified values for read-only elements without warning or error.

Analysis Services also ignores inappropriate or irrelevant properties without raising validation errors. For example, the X element should only be present when the Y element has a particular value. The Analysis Services instance ignores the X element instead of validating that element against the value of the Y element.

See Also

[Introducing Analysis Services Scripting Language](#)

ASSL XML Conventions

Analysis Services Scripting Language (ASSL) represents the hierarchy of objects as a set of element types, each of which defines the child elements they can contain.

To represent the object hierarchy, ASSL uses the following XML conventions:

- All objects and properties are represented as elements, except for standard XML attributes such as 'xml:lang'.
- Both element names and enumeration values follow the Microsoft .NET Framework naming convention of Pascal casing with no underscores.
- The case of all values is preserved. Values for enumerations are also case-sensitive.

In addition to this list of conventions, Analysis Services also follows certain conventions regarding cardinality, inheritance, whitespace, data types, and default values.



Note

For more information on each element's description, type, cardinality, and default value, as well as any relevant additional information, see [Introducing Analysis Services Scripting Language](#).

Cardinality

When an element has a cardinality that is greater than 1, there is an XML element collection that encapsulates this element. The name of collection uses the plural form of the elements contained in the collection. For example, the following XML fragment represents the

Dimensions collection within a **Database** element:

```
<Database>
...
<Dimensions>
  <Dimension>
    ...
  </Dimension>
  <Dimension>
    ...
  </Dimension>
</Dimensions>
</Database>
```

The order in which elements appear is unimportant.

Inheritance

Inheritance is used when there are distinct objects that have overlapping but significantly different sets of properties. Examples of such overlapping but distinct objects are virtual cubes, linked cubes, and regular cubes. For overlapping but distinct object, Analysis Services uses the standard **type** attribute from the XML Instance Namespace to indicate the inheritance. For example, the following XML fragment shows how the **type** attribute identifies whether a **Cube** element inherits from a regular cube or from a virtual cube:

```
<Cubes>
```

```

    <Cube xsi:type="RegularCube">
        <Name>Sales</Name>
        ...
    </Cube>
    <Cube xsi:type="VirtualCube">
        <Name>SalesAndInventory</Name>
        ...
    </Cube>
</Cubes>

```

Inheritance is generally not used when multiple types have a property of the same name. For example, the **Name** and **ID** properties appear on many elements, but these properties have not been promoted to an abstract type.

Whitespace

Whitespace within an element value is preserved. However, leading and trailing whitespace is always trimmed. For example, the following elements have the same text but differing amounts of whitespace within that text, and are therefore treated as if they have different values:

```

<Description>My text<Description>
<Description>My text<Description>

```

However, the following elements vary only in leading and trailing whitespace, and are therefore treated as if they have equivalent values:

```

<Description>My text<Description>
<Description> My text <Description>

```

Data Types

Analysis Services uses the following standard XML Schema definition language (XSD) data types:

Int

An integer value in the range of -231 to 231 – 1.

Long

An integer value in range of -263 to 263 – 1.

String

A string value that conforms to the following global rules:

- Control characters are stripped out.
- Leading and trailing white space is trimmed.

- Internal white space is preserved.

Name and **ID** properties have special limitations on valid characters in string elements. For additional information about **Name** and **ID** conventions, see [Objects and Object Characteristics](#).

DateTime

A **DateTime** structure from the .NET Framework. A **DateTime** value cannot be NULL. The lowest date supported by the **DateTime** data type is January 1, 1601, which is available to programmers as **DateTime.MinValue**. The lowest supported date indicates that a **DateTime** value is missing.

Boolean

An enumeration with only two values, such as {true, false} or {0, 1}.

Default Values

Analysis Services uses the defaults listed in the following table.

XML data type	Default value
Boolean	False
String	"" (empty string)
Integer or Long	0 (zero)
Timestamp	12:00:00 AM, 1/1/0001 (corresponding to a the .NET Frameworks System.DateTime with 0 ticks)

An element that is present but empty is interpreted as having a value of a null string, not the default value.

Inherited Defaults

Some properties that are specified on an object provide default values for the same property on child or descendant objects. For example, **Cube.StorageMode** provides the default value for **Partition.StorageMode**. The rules that Analysis Services applies for inherited default values are as follows:

- When the property for the child object is null in the XML, its value defaults to the inherited value. However, if you query the value from the server, the server returns the null value of the XML element.
- It is not possible to determine programmatically whether the property of a child object has been set directly on the child object or inherited.

Some elements have defined defaults that apply when the element is missing. For example, the **Dimension** elements in the following XML fragment are equivalent even though one **Dimension** element contains a **Visible** element, but the other **Dimension** element does not.

```
<Dimension>
  <Name>Product</Name>
</Dimension>

<Dimension>
  <Name>Product</ Name>
  <Visible>true</Visible>
</Dimension>
```

For more information on inherited defaults, see [Objects and Object Characteristics](#).

See Also

[Introducing Analysis Services Scripting Language](#)

XMLA Concepts

The XML for Analysis (XMLA) open standard supports data access to data sources that reside on the World Wide Web. Microsoft SQL Server Analysis Services implements XMLA per the XMLA 1.1 specification.

XML for Analysis (XMLA) is a Simple Object Access Protocol (SOAP)-based XML protocol, designed specifically for universal data access to any standard multidimensional data source residing on the Web. XMLA also eliminates the need to deploy a client component that exposes Component Object Model (COM) or Microsoft .NET Framework interfaces. XMLA is optimized for the Internet, when round trips to the server are expensive in terms of time and resources, and when stateful connections to a data source can limit user connections on the server.

XMLA is the native protocol for Microsoft SQL Server Analysis Services, used for all interaction between a client application and an instance of Analysis Services. Analysis Services fully supports XML for Analysis 1.1, and also provides extensions to support metadata management, session management, and locking capabilities. Both Analysis Management Objects (AMO) and ADOMD.NET use the XMLA protocol when communicating with an instance of Analysis Services.

Handling XMLA Communications

The XMLA open standard describes two generally accessible methods: **Discover** and **Execute**. These methods use the loosely-coupled client and server architecture supported by XML to handle incoming and outgoing information on an instance of Analysis Services.

The **Discover** method obtains information and metadata from a Web service. This information can include a list of available data sources, as well as information about any of the data source

providers. Properties define and shape the data that is obtained from a data source. The **Discover** method is a common method for defining the many types of information a client application may require from data sources on Analysis Services instances. The properties and the generic interface provide extensibility without requiring you to rewrite existing functions in a client application.

The **Execute** method allows applications to run provider-specific commands against XMLA data sources.

Although the XMLA protocol is optimized for Web applications, it can also be leveraged for LAN-oriented applications. The following applications can benefit from this XML-based API:

- Client/server applications that require flexible technology between clients and the server
- Client/server applications that target multiple operating systems
- Clients that do not require significant state in order to increase server capacity

XMLA and the Unified Dimensional Model

XMLA is the protocol used by business intelligence applications that employ the Unified Dimensional Model (UDM) methodology

Developing with XMLA in Analysis Services

XML for Analysis (XMLA) is a SOAP-based XML protocol, designed specifically for universal data access to any standard multidimensional data source that can be accessed over an HTTP connection. Analysis Services uses XMLA as its only protocol when communicating with client applications. Fundamentally, all client libraries supported by Analysis Services formulate requests and responses in XMLA.

As a developer, you can use XMLA to integrate a client application with Analysis Services, without any dependencies on the .NET Framework or COM interfaces. Application requirements that include hosting on a wide range of platforms can be satisfied by using XMLA and an HTTP connection to Analysis Services.

Analysis Services is fully compliant with the 1.1 specification of XMLA, but also extends it to enable data definition, data manipulation, and data control support. Analysis Services extensions are referred to as the Analysis Services Scripting Language (ASSL). Using XMLA and ASSL together enables a broader set of functionality than what XMLA alone provides. For more information about ASSL, see [Developing with Analysis Services Scripting Language \(ASSL\)](#).

In This Section

Topic	Description
Managing Connections and Sessions	Describes how to connect to an Analysis Services instance, and how to manage sessions and statefulness in XMLA.

Topic	Description
Handling Errors and Warnings (XMLA)	Describes how Analysis Services returns error and warning information for methods and commands in XMLA.
Defining and Identifying Objects (XMLA)	Describes object identifiers and object references, and how to use identifiers and references within XMLA commands.
Managing Transactions (XMLA)	Details how to use the BeginTransaction , CommitTransaction , and RollbackTransaction commands to explicitly define and manage a transaction on the current XMLA session.
Canceling Commands (XMLA)	Describes how to use the Cancel command to cancel commands, sessions, and connections in XMLA.
Performing Batch Operations (XMLA)	Describes how to use the Batch command to run multiple XMLA commands, in serial or in parallel, either within the same transaction or as separate transactions, using a single XMLA Execute method.
Creating and Altering Objects (XMLA)	Describes how to use the Create , Alter , and Delete commands, along with Analysis Services Scripting Language (ASSL) elements, to define, change, or remove objects from an Analysis Services instance.
Locking and Unlocking Databases (XMLA)	Details how to use the Lock and Unlock commands to lock and unlock an Analysis Services database.
Processing Objects (XMLA)	Describes how to use the Process command to process an Analysis Services object.
Merging Partitions (XMLA)	Describes how to use the MergePartitions command to merge partitions on an Analysis Services instance.
Designing Aggregations (XMLA)	Describes how to use the DesignAggregations command, either in iterative or batch mode, to design aggregations for an aggregation design in

Topic	Description
	Analysis Services.
Backing Up, Restoring, and Synchronizing Databases (XMLA)	Describes how to use the Backup and Restore commands to back up and restore an Analysis Services database from a backup file. Also describes how to use the Synchronize command to synchronize an Analysis Services database with an existing database on the same instance or on a different instance.
Inserting, Updating, and Dropping Members (XMLA)	Describes how to use the Insert , Update , and Drop commands to add, change, or delete members from a write-enabled dimension.
Updating Cells (XMLA)	Describes how to use the UpdateCells command to change the values of cells in a write-enabled partition.
Managing Caches (XMLA)	Details how to use the ClearCache command to clear the caches of Analysis Services objects.
Monitoring Traces (XMLA)	Describes how to use the Subscribe command to subscribe to and monitor an existing trace on an Analysis Services instance.

Data Mining with XMLA

XML for Analysis fully supports data mining schema rowsets. These rowsets provide information for querying data mining models using the [Discover](#) method. For more information about data mining schema rowsets, see [Data Mining Schema Rowsets](#)

For more information about DMX, see [Data Mining Extensions \(DMX\) Reference](#).

Namespace and Schema

Namespace

The schema defined in this specification uses the XML namespace <http://schemas.microsoft.com/AnalysisServices/2003/Engine> and the standard abbreviation "DDL."

Schema

The definition of an XML Schema definition language (XSD) schema for the Analysis Services object definition language is based on the definition of the schema elements and hierarchy in this section.

Extensibility

Extensibility of the object definition language schema is provided by means of an **Annotation** element that is included on all objects. This element can contain any valid XML from any XML namespace (other than the target namespace that defines the DDL), subject to the following rules:

- The XML can contain only elements.
- Each element must have a unique name. It is recommended that the value of **Name** reference the target namespace.

These rules are imposed so that the contents of the **Annotation** tag can be exposed as a set of Name/Value pairs through Decision Support Objects (DSO) 9.0.

Comments and white space within the **Annotation** tag that are not enclosed with a child element may not be preserved. In addition, all elements must be read-write; read-only elements are ignored.

The object definition language schema is closed, in that the server does not allow substitution of derived types for elements defined in the schema. Therefore, the server accepts only the set of elements defined here, and no other elements or attributes. Unknown elements cause the Analysis Services engine to raise an error.

See Also

[ASSL](#)

[Planning and Architecture \(Analysis Services - Multidimensional Data\)](#)

Managing Connections and Sessions (XMLA)

Statefulness is a condition during which the server preserves the identity and context of a client between method calls. *Statelessness* is a condition during which the server does not remember the identity and context of a client after a method call finishes.

To provide statefulness, XML for Analysis (XMLA) supports *sessions* that allow a series of statements to be performed together. An example of such a series of statements would be the creation of a calculated member that is to be used in subsequent queries.

In general, sessions in XMLA follow the following behavior outlined by the OLE DB 2.6 specification:

- Sessions define transaction and command context scope.
- Multiple commands can be run in the context of a single session.
- Support for transactions in the XMLA context is through provider-specific commands sent with the [Execute](#) method.

XMLA defines a way to support sessions in a Web environment in a mode similar to the approach used by the Distributed Authoring and Versioning (DAV) protocol to implement locking in a loosely coupled environment. This implementation parallels DAV in that the provider is allowed to expire sessions for various reasons (for example, a timeout or connection error). When sessions are supported, Web services must be aware and ready to handle interrupted sets of commands that must be restarted.

The World Wide Web Consortium (W3C) Simple Object Access Protocol (SOAP) specification recommends using SOAP headers for building up new protocols on top of SOAP messages. The following table lists the SOAP header elements and attributes that XMLA defines for initiating, maintaining, and closing a session.

SOAP header	Description
BeginSession	This header requests the provider to create a new session. The provider should respond by constructing a new session and returning the session ID as part of the Session header in the SOAP response.
SessionId	The value area contains the session ID that must be used in each method call for the rest of the session. The provider in the SOAP response sends this tag and the client must also send this attribute with each Session header element.
Session	For every method call that occurs in the session, this header must be used, and the session ID must be included in the value area of the header.
EndSession	To terminate the session, use this header. The session ID must be included with the value area.



Note

A session ID does not guarantee that a session stays valid. If the session expires (for example, if it times out or the connection is lost), the provider can choose to end and roll back that session's actions. As a result, all subsequent method calls from the client on a session ID fail with an error signaling a session that is not valid. A client should handle this condition and be prepared to resend the session method calls from the beginning.

Legacy Code Example

The following example shows how sessions are supported.

1. To begin the session, add a BeginSession header in SOAP to the outbound XMLA method call from the client. The value area is initially blank because the session ID is not yet known.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  <SOAP-ENV:Header>
    <XA:BeginSession
      xmlns:XA="urn:schemas-microsoft-com:xml-analysis"
      xsi:type="xsd:int"
      mustUnderstand="1"/>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      ...<!-- Discover or Execute call goes here.-->
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

2. The SOAP response message from the provider includes the session ID in the return header area, using the XMLA header tag <SessionId>.

```
<SOAP-ENV:Header>
  <XA:Session
    xmlns:XA="urn:schemas-microsoft-com:xml-analysis"
    SessionId="581"/>
  </SOAP-ENV:Header>
```

3. For each method call in the session, the Session header must be added, containing the session ID returned from the provider.

```
<SOAP-ENV:Header>
  <XA:Session
    xmlns:XA="urn:schemas-microsoft-com:xml-analysis"
    mustUnderstand="1"
    SessionId="581"/>
  </SOAP-ENV:Header>
```

4. When the session is complete, the <EndSession> tag is used, containing the related session ID value.

```
<SOAP-ENV:Header>
  <XA:EndSession
```

```

xmlns:XA="urn:schemas-microsoft-com:xml-analysis"

xsi:type="xsd:int"

mustUnderstand="1"

SessionId="581"/>

</SOAP-ENV:Header>

```

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Handling Errors and Warnings (XMLA)

Error handling is required when an XML for Analysis (XMLA) [Discover](#) or [Execute](#) method call does not run, runs successfully but generates errors or warnings, or runs successfully but returns results that contain errors.

Error	Reporting
The XMLA method call does not run	Microsoft SQL Server Analysis Services returns a SOAP fault message that contains the details of the failure. For more information, see the section, Handling SOAP Faults .
Errors or warnings on a successful method call	Analysis Services includes an error or warning element for each error or warning, respectively, in the Messages property of the root element that contains the results of the method call. For more information, see the section, Handling Errors and Warnings .
Errors in the result for a successful method call	Analysis Services includes an inline error or warning element for the error or warning, respectively, within the appropriate Cell or row element of the results of the method call. For more information, see the section, Handling Inline Errors and Warnings .

Handling SOAP Faults

Analysis Services returns a SOAP fault when the following situations occur:

- The SOAP message that contains the XMLA method was not well-formed or could not be validated by the Analysis Services instance.
- A communications or other error occurred involving the SOAP message that contains the XMLA method.
- The XMLA method did not run on the Analysis Services instance.

The SOAP fault codes for XMLstartA start with "XMLForAnalysis", followed by a period and the hexadecimal HRESULT result code. For example, an error code of "0x80000005" is formatted as "XMLForAnalysis.0x80000005". For more information about the SOAP fault format, see Soap Fault in the W3C Simple Object Access Protocol (SOAP) 1.1.

Fault Code Information

The following table shows the XMLA fault code information that is contained in the detail section of the SOAP response. The columns are the attributes of an error in the detail section of a SOAP fault.

Column name	Type	Description	Null allowed ¹
ErrorCode	UnsignedInt	Return code that indicates the success or failure of the method. The hexadecimal value must be converted to an UnsignedInt value.	No
WarningCode	UnsignedInt	Return code that indicates a warning condition. The hexadecimal value must be converted to an UnsignedInt value.	Yes
Description	String	Error or warning text and description returned by the component that generated the error.	Yes
Source	String	Name of the component that generated the error or warning.	Yes
HelpFile	String	Path or URL to the	Yes

Column name	Type	Description	Null allowed ¹
		Help file or topic that describes the error or warning.	

¹ Indicates whether the data is required and must be returned, or whether the data is optional and a null string is allowed if the column does not apply.

The following is an example of a SOAP fault that occurred when a method call failed:

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

  <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    <SOAP-ENV:Fault>

      <faultcode>XMLAnalysisError.0x80000005</faultcode>

      <faultstring>The XML for Analysis provider encountered an error.</faultstring>

      <faultactor>XML for Analysis Provider</faultactor>

      <detail>

<Error
  ErrorCode="2147483653"
  Description="An unexpected error has occurred."
  Source="XML for Analysis Provider"
  HelpFile="" />

      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Envelope>
```

Handling Errors and Warnings

Analysis Services returns the **Messages** property in the **root** element for a command if the following situations occur after that command runs:

- The method itself did not fail, but a failure occurred on the Analysis Services instance after the method call succeeded.
- The Analysis Services instance returns a warning when the command is successful.

The **Messages** property follows all other properties that are contained by the **root** element, and can contain one or more **Message** elements. In turn, each **Message** element can contain either a single **error** or **warning** element describing any errors or warnings, respectively, that occurred for the specified command.

For more information about errors and warnings that are contained in the **Messages** property, see [XML for Analysis Overview \(XMLA\) - deleted](#).

Handling Errors During Serialization

If an error occurs after the Analysis Services instance has already begun serializing the output of a successfully run command, Analysis Services returns an [Exception](#) element in a different namespace at the point of the error. The Analysis Services instance then closes all open elements so that the XML document sent to the client is a valid document. The instance also returns a **Messages** element that contains the description of the error.

Handling Inline Errors and Warnings

Analysis Services returns an inline **error** or **warning** for a command if the XMLA method itself did not fail, but an error specific to a data element in the results returned by the method occurred on the Analysis Services instance after the XMLA method call succeeded.

Analysis Services supplies inline **error** and **warning** elements if issues specific to a cell or to other data that are contained within a **root** element using the [MDDDataSet](#) data type occur, such as a security error or formatting error for a cell. In these cases, Analysis Services returns an **error** or **warning** element in the **Cell** or **row** element that contains the error or warning, respectively.

The following example illustrates a result set that contains an error in the rowset returned from an **Execute** method using the [Statement](#) command.

```
<return>

...
<root>

...
  <CellData>

...
    <Cell CellOrdinal="10">
      <Value>
        <Error>
          <ErrorCode>2148497527</ErrorCode>
          <Description>Security Error.</Description>
        </Error>
      </Value>
    </Cell>
  </CellData>

...
</root>

...
```

</return>

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

[XML for Analysis Overview \(XMLA\)](#)

Defining and Identifying Objects (XMLA)

Objects are identified in XML for Analysis (XMLA) commands by using object identifiers and object references, and are defined by using Analysis Services Scripting Language (ASSL) elements XMLA commands.

Object Identifiers

An object is identified by using the unique identifier of the object as defined on an instance of Microsoft SQL Server Analysis Services. Object identifiers can either be explicitly specified or determined by the Analysis Services instance when Analysis Services creates the object. You can use the [Discover](#) method to retrieve object identifiers for subsequent **Discover** or [Execute](#) method calls.

Object References

Several XMLA commands, such as [Delete](#) or [Process](#), use an object reference to refer to an object in an unambiguous manner. An object reference contains the object identifier of the object on which a command is executed and the object identifiers of the ancestors for that object. For example, the object reference for a partition contains the object identifier of the partition, as well as the object identifiers of that partition's parent measure group, cube, and database.

Object Definitions

The [Create](#) and [Alter](#) commands in XMLA create or alter, respectively, objects on an Analysis Services instance. The definitions for those objects are represented by an [ObjectDefinition](#) element that contains elements from ASSL. Object identifiers can be explicitly specified for all major and many minor objects by using the [ID](#) element. If the **ID** element is not used, the Analysis Services instance provides a unique identifier, with a naming convention that depends on the object to be identified. For more information about how to use the **Create** and **Alter** commands to define objects, see [Using XML for Analysis in Analysis Services \(XMLA\)](#).

See Also

[Object Element \(XMLA\)](#)

[ParentObject Element \(XMLA\)](#)

[Source Element \(XMLA\)](#)

[Target Element \(XMLA\)](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Managing Transactions (XMLA)

Every XML for Analysis (XMLA) command sent to an instance of Microsoft SQL Server Analysis Services runs within the context of a transaction on the current implicit or explicit session. To manage each of these transactions, you use the [BeginTransaction](#), [CommitTransaction](#), and [RollbackTransaction](#) commands. By using these commands, you can create implicit or explicit transactions, change the transaction reference count, as well as start, commit, or roll back transactions.

Implicit and Explicit Transactions

A transaction is either implicit or explicit:

Implicit transaction

Analysis Services creates an *implicit* transaction for an XMLA command if the **BeginTransaction** command does not specify the start of a transaction. Analysis Services always commits an implicit transaction if the command succeeds, and rolls back an implicit transaction if the command fails.

Explicit transaction

Analysis Services creates an *explicit* transaction if the **BeginTransaction** command starts of a transaction. However, Analysis Services only commits an explicit transaction if a **CommitTransaction** command is sent, and rolls back an explicit transaction if a **RollbackTransaction** command is sent.

In addition, Analysis Services rolls back both implicit and explicit transactions if the current session ends before the active transaction completes.

Transactions and Reference Counts

Analysis Services maintains a transaction reference count for each session. However, Analysis Services does not support nested transactions in that only one active transaction is maintained per session. If the current session does not have an active transaction, the transaction reference count is set to zero.

In other words, each **BeginTransaction** command increments the reference count by one, while each **CommitTransaction** command decrements the reference count by one. If a **CommitTransaction** command sets the transaction count to zero, Analysis Services commits the transaction.

However, the **RollbackTransaction** command rolls back the active transaction regardless of the current value of the transaction reference count. In other words, a single **RollbackTransaction** command rolls back the active transaction, no matter how many **BeginTransaction** commands or **CommitTransaction** commands were sent, and sets the transaction reference count to zero.

Beginning a Transaction

The **BeginTransaction** command begins an explicit transaction on the current session and increments the transaction reference count for the current session by one. All subsequent commands are considered to be within the active transaction, until either enough

CommitTransaction commands are sent to commit the active transaction or a single **RollbackTransaction** command is sent to roll back the active transaction.

Committing a Transaction

The **CommitTransaction** command commits the results of commands that are run after the **BeginTransaction** command was run on the current session. Each **CommitTransaction** command decrements the reference count for active transactions on a session. If a **CommitTransaction** command sets the reference count to zero, Analysis Services commits the active transaction. If there is no active transaction (in other words, the transaction reference count for the current session is already set to zero), a **CommitTransaction** command results in an error.

Rolling Back a Transaction

The **RollbackTransaction** command rolls back the results of commands that are run after the **BeginTransaction** command was run on the current session. The **RollbackTransaction** command rolls back the active transaction, regardless of the current transaction reference count, and sets the transaction reference count to zero. If there is no active transaction (in other words, the transaction reference count for the current session is already set to zero), a **RollbackTransaction** command results in an error.

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Canceling Commands (XMLA)

Depending on the administrative permissions of the user issuing the command, the [Cancel](#) command in XML for Analysis (XMLA) can cancel a command on a session, a session, a connection, a server process, or an associated session or connection.

Canceling Commands

A user can cancel the currently executing command within the context of the current explicit session by sending a **Cancel** command with no specified properties.



Note

A command running in an implicit session cannot be canceled by a user.

Canceling Batch Commands

If a user cancels a **Batch** command, then all remaining commands not yet executed within the **Batch** command are canceled. If the **Batch** command was transactional, any commands that were executed before the **Cancel** command runs are rolled back.

Canceling Sessions

By specifying a session identifier for an explicit session in the [SessionID](#) property of the **Cancel** command, a database administrator or server administrator can cancel a session, including the currently executing command. A database administrator can only cancel sessions for databases on which he or she has administrative permissions.

A database administrator can retrieve the active sessions for a specified database by retrieving the DISCOVER_SESSIONS schema rowset. To retrieve the DISCOVER_SESSIONS schema rowset, the database administrator uses the XMLA **Discover** method and specifies the appropriate database identifier for the SESSION_CURRENT_DATABASE restriction column in the [Restrictions](#) property of the **Discover** method.

Canceling Connections

By specifying a connection identifier in the [ConnectionID](#) property of the **Cancel** command, a server administrator can cancel all of the sessions associated with a given connection, including all running commands, and cancel the connection.



Note

If the instance of Microsoft SQL Server Analysis Services cannot locate and cancel the sessions associated with a connection, such as when the data pump opens multiple sessions while providing HTTP connectivity, the instance cannot cancel the connection. If this case is encountered during the execution of a **Cancel** command, an error occurs.

A server administrator can retrieve the active connections for an Analysis Services instance by retrieving the DISCOVER_CONNECTIONS schema rowset using the XMLA **Discover** method.

Canceling Server Processes

By specifying a server process identifier (SPID) in the [SPID](#) property of the **Cancel** command, a server administrator can cancel the commands associated with a given SPID.

Canceling Associated Sessions and Connections

You can set the [CancelAssociated](#) property to true to cancel the connections, sessions, and commands associated with the connection, session, or SPID specified in the **Cancel** command.

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Performing Batch Operations (XMLA)

You can use the [Batch](#) command in XML for Analysis (XMLA) to run multiple XMLA commands using a single XMLA [Execute](#) method. You can run multiple commands contained in the **Batch** command either as a single transaction or in individual transactions for each command, in serial or in parallel. You can also specify out-of-line bindings and other properties in the **Batch** command for processing multiple Microsoft SQL Server Analysis Services objects.

Running Transactional and Nontransactional Batch Commands

The **Batch** command executes commands in one of two ways:

Transactional

If the **Transaction** attribute of the **Batch** command is set to true, the **Batch** command runs all of the commands contained by the **Batch** command in a single transaction—a *transactional batch*.

If any command fails in a transactional batch, Analysis Services rolls back any command in the **Batch** command that ran before the command that failed and the **Batch** command immediately ends. Any commands in the **Batch** command that have not yet run are not executed. After the **Batch** command ends, the **Batch** command reports any errors that occurred for the failed command.

Nontransactional

If the **Transaction** attribute is set to false, the **Batch** command runs each command contained by the **Batch** command in a separate transaction—a *nontransactional* batch. If any command fails in a nontransactional batch, the **Batch** command continues to run commands after the command which failed. After the **Batch** command tries to run all the commands that the **Batch** command contains, the **Batch** command reports any errors that occurred.

All results returned by commands contained in a **Batch** command are returned in the same order in which the commands are contained in the **Batch** command. The results returned by a **Batch** command vary based on whether the **Batch** command is transactional or nontransactional.

Note

If a **Batch** command contains a command that does not return output, such as the [Lock](#) command, and that command successfully runs, the **Batch** command returns an empty [root](#) element within the results element. The empty **root** element ensures that each command contained in a **Batch** command can be matched with the appropriate **root** element for that command's results.

Returning Results from Transactional Batch Results

Results from commands run within a transactional batch are not returned until the entire **Batch** command is completed. The results are not returned after each command runs because any command that fails within a transactional batch would cause the entire **Batch** command and all containing commands to be rolled back. If all commands start and run successfully, the [return](#) element of the [ExecuteResponse](#) element returned by the **Execute** method for the **Batch** command contains one [results](#) element, which in turn contains one **root** element for each successfully run command contained in the **Batch** command. If any command in the **Batch** command cannot be started or fails to complete, the **Execute** method returns a SOAP fault for the **Batch** command that contains the error of the command that failed.

Returning Results from Nontransactional Batch Results

Results from commands run within a nontransactional batch are returned in the order in which the commands are contained within the **Batch** command and as they are returned by each command. If no command contained in the **Batch** command can be successfully started, the **Execute** method returns a SOAP fault that contains an error for the **Batch** command. If at least one command is successfully started, the **return** element of the **ExecuteResponse** element returned by the **Execute** method for the **Batch** command contains one **results** element, which in turn contains one **root** element for each command contained in the **Batch** command. If one

or more commands in a nontransactional batch cannot be started or fails to complete, the **root** element for that failed command contains an [error](#) element describing the error.

Note

As long as at least one command in a nontransactional batch can be started, the nontransactional batch is considered to have successfully run, even if every command contained in the nontransactional batch returns an error in the results of the **Batch** command.

Using Serial and Parallel Execution

You can use the **Batch** command to run included commands in serial or in parallel. When the commands are run in serial, the next command included in the **Batch** command cannot start until the currently running command in the **Batch** command is completed. When the commands are run in parallel, multiple commands can be executed simultaneously by the **Batch** command.

To run commands in parallel, you add the commands to be run in parallel to the [Parallel](#) property of the **Batch** command. Currently, Analysis Services can run only contiguous, sequential [Process](#) commands in parallel. Any other XMLA command, such as [Create](#) or [Alter](#), included in the **Parallel** property is run serially.

Analysis Services tries to run all **Process** commands included in the **Parallel** property in parallel, but cannot guarantee that all included **Process** commands can be run in parallel. The instance analyzes each **Process** command and, if the instance determines that the command cannot be run in parallel, the **Process** command is run in serial.

Note

To run commands in parallel, the **Transaction** attribute of the **Batch** command must be set to true because Analysis Services supports only one active transaction per connection and nontransactional batches run each command in a separate transaction. If you include the **Parallel** property in a nontransactional batch, an error occurs.

Limiting Parallel Execution

An Analysis Services instance tries to run as many **Process** commands in parallel as possible, up to the limits of the computer on which the instance runs. You can limit the number of concurrently executing **Process** commands by setting the **maxParallel** attribute of the **Parallel** property to a value indicating the maximum number of **Process** commands that can be run in parallel.

For example, a **Parallel** property contains the following commands in the sequence listed:

1. **Create**
2. **Process**
3. **Alter**
4. **Process**
5. **Process**
6. **Process**

7. **Delete**
8. **Process**
9. **Process**

The **maxParallel** attribute of this **Parallel** property is set to 2. Therefore, the instance runs the previous lists of commands as described in the following list:

- Command 1 runs serially because command 1 is a **Create** command and only **Process** commands can be run in parallel.
- Command 2 runs serially after command 1 is completed.
- Command 3 runs serially after command 2 is completed.
- Commands 4 and 5 run in parallel after command 3 is completed. Although command 6 is also a **Process** command, command 6 cannot run in parallel with commands 4 and 5 because the **maxParallel** property is set to 2.
- Command 6 runs serially after both commands 4 and 5 are completed.
- Command 7 runs serially after command 6 is completed.
- Commands 8 and 9 run in parallel after command 7 is completed.

Using the Batch Command to Process Objects

The **Batch** command contains several optional properties and attributes specifically included to support processing multiple Analysis Services projects:

- The **ProcessAffectedObjects** attribute of the **Batch** command indicates whether the instance should also process any object that requires reprocessing as a result of a **Process** command included in the **Batch** command processing a specified object.
- The [Bindings](#) property contains a collection of out-of-line bindings used by all of the **Process** commands in the **Batch** command.
- The [DataSource](#) property contains an out-of-line binding for a data source used by all of the **Process** commands in the **Batch** command.
- The [DataSourceView](#) property contains an out-of-line binding for a data source view used by all of the **Process** commands in the **Batch** command.
- The [ErrorConfiguration](#) property specifies the way in which the **Batch** command handles errors encountered by all **Process** commands contained in the **Batch** command.



Important

A **Process** command cannot include the **Bindings**, **DataSource**, **DataSourceView**, or **ErrorConfiguration** properties, if the **Process** command is contained in a **Batch** command. If you must specify these properties for a **Process** command, provide the necessary information in the corresponding properties of the **Batch** command that contains the **Process** command.

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

[Process Element \(XMLA\)](#)

Creating and Altering Objects (XMLA)

Major objects can be independently created, altered, and deleted. Major objects include the following objects:

- Servers
- Databases
- Dimensions
- Cubes
- Measure groups
- Partitions
- Perspectives
- Mining models
- Roles
- Commands associated with a server or database
- Data sources

You use the [Create](#) command to create a major object on an instance of Microsoft SQL Server Analysis Services, and the [Alter](#) command to alter an existing major object on an instance. Both commands are run using the [Execute](#) method.

Creating Objects

When you create objects by using the **Create** method, you must first identify the parent object that contains the Analysis Services object to be created. You identify the parent object by providing an object reference in the [ParentObject](#) property of the **Create** command. Each object reference contains the object identifiers needed to uniquely identify the parent object for the **Create** command. For more information about object references, see [Using XML for Analysis in Analysis Services \(XMLA\)](#).

For example, you must provide an object reference to a cube to create a new measure group for the cube. The object reference for the cube in the **ParentObject** property contains both a database identifier and a cube identifier, as the same cube identifier could potentially be used on a different database.

The [ObjectDefinition](#) element contains Analysis Services Scripting Language (ASSL) elements that define the major object to be created. For more information about ASSL, see [ASSL](#).

If you set the **AllowOverwrite** attribute of the **Create** command to true, you can overwrite an existing major object that has the specified identifier. Otherwise, an error occurs if a major object that has the specified identifier already exists in the parent object.

For more information about the **Create** command, see [Create Element \(XMLA\)](#).

Creating Session Objects

Session objects are temporary objects that are available only to the explicit or implicit session used by a client application and are deleted when the session is ended. You can create session objects by setting the **Scope** attribute of the **Create** command to Session.



Note

When using the Session setting, the **ObjectDefinition** element can only contain [Dimension](#), [Cube](#), or [MiningModel](#) ASSL elements.

Altering Objects

When modifying objects by using the **Alter** method, you must first identify the object to be modified by providing an object reference in the [Object](#) property of the **Alter** command. Each object reference contains the object identifiers needed to uniquely identify the object for the **Alter** command. For more information about object references, see [Defining and Identifying Objects \(XMLA\)](#).

For example, you must provide an object reference to a cube in order to modify the structure of a cube. The object reference for the cube in the **Object** property contains both a database identifier and a cube identifier, as the same cube identifier could potentially be used on a different database.

The **ObjectDefinition** element contains ASSL elements that define the major object to be modified. For more information about ASSL, see [ASSL](#).

If you set the **AllowCreate** attribute of the **Alter** command to true, you can create the specified major object if the object does not exist. Otherwise, an error occurs if a specified major object does not already exist.

Using the ObjectExpansion Attribute

If you are changing only the properties of the major object and are not redefining minor objects that are contained by the major object, you can set the **ObjectExpansion** attribute of the **Alter** command to **ObjectProperties**. The **ObjectDefinition** property then only has to contain the elements for the properties of the major object, and the **Alter** command leaves minor objects associated with the major object untouched.

To redefine minor objects for a major object, you must set the **ObjectExpansion** attribute to **ExpandFull** and the object definition must include all minor objects that are contained by the major object. If the **ObjectDefinition** property of the **Alter** command does not explicitly include a minor object that is contained by the major object, the minor object that was not included is deleted.

Altering Session Objects

To modify session objects created by the **Create** command, set the **Scope** attribute of the **Alter** command to Session.



Note

When using the Session setting, the **ObjectDefinition** element can only contain [Dimension](#), [Cube](#), or [MiningModel](#) ASSL elements.

Creating or Altering Subordinate Objects

Although a **Create** or **Alter** command creates or alters only one topmost major object, the major object being created or modified can contain definitions within the enclosing **ObjectDefinition** property for other major and minor objects that are subordinate to it. For example, if you define a cube, you specify the parent database in **ParentObject**, and within the cube definition in **ObjectDefinition** you can define measure groups for the cube, and within the measure groups you can define partitions for each measure group. A minor object can be defined only under the major object that contains it. For more information about major and minor objects, see [Analysis Services Objects \(SSAS\)](#).

Examples

Description

The following example creates a relational data source that references the Adventure Works DW Multidimensional 2012 sample Microsoft SQL Server database.

Code

```
<Create xmlns="http://schemas.microsoft.com/analysiservices/2003/engine">

  <ParentObject>

    <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

  </ParentObject>

  <ObjectDefinition>

    <DataSource xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="RelationalDataSource">

      <ID>AdventureWorksDW2012</ID>

      <Name>AdventureWorksDW2012</Name>

      <ConnectionString>Data Source=localhost;Initial
Catalog=AdventureWorksDW2008R2;Integrated Security=True</ConnectionString>

      <ImpersonationInfo>

        <ImpersonationMode>ImpersonateServiceAccount</ImpersonationMode>

      </ImpersonationInfo>

      <ManagedProvider>System.Data.SqlClient</ManagedProvider>

      <Timeout>PT0S</Timeout>

    </DataSource>

  </ObjectDefinition>

</Create>
```

Description

The following example alters the relational data source created in the previous example to set the query time-out for the data source to 30 seconds.

Code

```

<Alter ObjectExpansion="ObjectProperties"
xmlns="http://schemas.microsoft.com/analysiservices/2003/engine">

  <Object>

    <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

    <DataSourceID>AdventureWorksDW2012</DataSourceID>

  </Object>

  <ObjectDefinition>

    <DataSource xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="RelationalDataSource">

      <ID>AdventureWorksDW2012</ID>

      <Name>AdventureWorksDW2012</Name>

      <ConnectionString>Data Source=fr-dwk-02;Initial
Catalog=AdventureWorksDW2008R2;Integrated Security=True</ConnectionString>

      <ManagedProvider>System.Data.SqlClient</ManagedProvider>

      <Timeout>PT30S</Timeout>

    </DataSource>

  </ObjectDefinition>

</Alter>

```

Comments

The **ObjectExpansion** attribute of the **Alter** command was set to **ObjectProperties**. This setting allows the [ImpersonationInfo](#) element, a minor object, to be excluded from the data source defined in **ObjectDefinition**. Therefore, the impersonation information for that data source remains set to the service account, as specified in the first example.

See Also

[Execute Method \(XMLA\)](#)

[ASSL](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Locking and Unlocking Databases (XMLA)

You can lock and unlock databases using, respectively, the [Lock](#) and [Unlock](#) commands in XML for Analysis (XMLA). Typically, other XMLA commands automatically lock and unlock objects as needed to complete the command during execution. You can explicitly lock or unlock a database to perform multiple commands within a single transaction, such as a [Batch](#) command, while preventing other applications from committing a write transaction to the database.

Locking Databases

The **Lock** command locks an object, either for shared or exclusive use, within the context of the currently active transaction. A lock on an object prevents transactions from committing until the lock is removed. Microsoft SQL Server Analysis Services supports two types of locks, shared locks and exclusive locks. For more information about the lock types supported by Analysis Services, see [Using XML for Analysis in Analysis Services \(XMLA\)](#).

Analysis Services allows only databases to be locked. The [Object](#) element must contain an object reference to an Analysis Services database. If the **Object** element is not specified or if the **Object** element refers to an object other than a database, an error occurs.

noteDXDOC112778PADS **Security Note**

Only database administrators or server administrators can explicitly issue a **Lock** command.

Other commands implicitly issue a **Lock** command on an Analysis Services database. Any operation that reads data or metadata from a database, such as any [Discover](#) method or an [Execute](#) method running a [Statement](#) command, implicitly issues a shared lock on the database. Any transaction that commits changes in data or metadata to an object on an Analysis Services database, such as an **Execute** method running an [Alter](#) command, implicitly issues an exclusive lock on the database.

Unlocking Objects

The **Unlock** command removes a lock established within the context of the currently active transaction.

noteDXDOC112778PADS **Security Note**

Only database administrators or server administrators can explicitly issue an **Unlock** command.

All locks are held in the context of the current transaction. When the current transaction is committed or rolled back, all locks defined within the transaction are automatically released.

See Also

[Lock Element \(XMLA\)](#)

[Unlock Element \(XMLA\)](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Processing Objects (XMLA)

In Microsoft SQL Server Analysis Services, processing is the step or series of steps that turn data into information for business analysis. Processing is different depending on the type of object, but processing is always part of turning data into information.

To process an Analysis Services object, you can use the [Process](#) command. The **Process** command can process the following objects on an Analysis Services instance:

- Cubes
- Databases

- Dimensions
- Measure groups
- Mining models
- Mining structures
- Partitions

To control the processing of objects, the **Process** command has various properties that can be set. The **Process** command has properties that control: how much processing will be done, which objects will be processed, whether to use out-of-line bindings, how to handle errors, and how to manage writeback tables.

Specifying Processing Options

The [Type](#) property of the **Process** command specifies the processing option to use when processing the object. For more information about processing options, see [WritebackTableCreation Element \(XMLA\)](#).

The following table lists the constants for the **Type** property and the various objects that can be processed using each constant.

Type value	Applicable objects
ProcessFull	Cube, database, dimension, measure group, mining model, mining structure, partition
ProcessAdd	Dimension, partition
ProcessUpdate	Dimension
ProcessIndexes	Dimension, cube, measure group, partition
ProcessData	Dimension, cube, measure group, partition
ProcessDefault	Cube, database, dimension, measure group, mining model, mining structure, partition
ProcessClear	Cube, database, dimension, measure group, mining model, mining structure, partition
ProcessStructure	Cube, mining structure
ProcessClearStructureOnly	Mining structure
ProcessScriptCache	Cube

For more information about processing Analysis Services objects, see [Processing Analysis Services Objects](#).

Specifying Objects to be Processed

The [Object](#) property of the **Process** command contains the object identifier of the object to be processed. Only one object can be specified in a **Process** command, but processing an object also processes any child objects. For example, processing a measure group in a cube processes all the partitions for that measure group, while processing a database processes all the objects, including cubes, dimensions, and mining structures, that are contained by the database.

If you set the **ProcessAffectedObjects** attribute of the **Process** command to true, any related object affected by processing the specified object is also processed. For example, if a dimension is incrementally updated by using the **ProcessUpdate** processing option in the **Process** command, any partition whose aggregations are invalidated because of members being added or deleted is also processed by Analysis Services if **ProcessAffectedObjects** is set to true. In this case, a single **Process** command can process multiple objects on an Analysis Services instance, but Analysis Services determines which objects besides the single object specified in the **Process** command must also be processed.

However, you can process multiple objects, such as dimensions, at the same time by using multiple **Process** commands within a **Batch** command. Batch operations provide a finer level of control for serial or parallel processing of objects on an Analysis Services instance than using the **ProcessAffectedObjects** attribute, and let you to tune your processing approach for larger Analysis Services databases. For more information about performing batch operations, see [Performing Batch Operations \(XMLA\)](#).

Specifying Out-of-Line Bindings

If the **Process** command is not contained by a **Batch** command, you can optionally specify out-of-line bindings in the [Bindings](#), [DataSource](#), and [DataSourceView](#) properties of the **Process** command for the objects to be processed. Out-of-line bindings are references to data sources, data source views, and other objects in which the binding exists only during the execution of the **Process** command, and which override any existing bindings associated with the objects being processed. If out-of-line bindings are not specified, the bindings currently associated with the objects to be processed are used.

Out-of-line bindings are used in the following circumstances:

- Incrementally processing a partition, in which an alternative fact table or a filter on the existing fact table must be specified to make sure that rows are not counted twice.
- Using a data flow task in Microsoft SQL Server Integration Services to provide data while processing a dimension, mining model, or partition.

Out-of-line bindings are described as part of the Analysis Services Scripting Language (ASSL). For more information about out-of-line bindings in ASSL, see [Data Sources and Bindings](#).

Incrementally Updating Partitions

Incrementally updating an already processed partition typically requires an out-of-line binding because the binding specified for the partition references fact table data already aggregated within the partition. When incrementally updating an already processed partition by using the **Process** command, Analysis Services performs the following actions:

- Creates a temporary partition with a structure identical to that of the partition to be incrementally updated.
- Processes the temporary partition, using the out-of-line binding specified in the **Process** command.
- Merges the temporary partition with the existing selected partition.

For more information about merging partitions using XML for Analysis (XMLA), see [Merging Partitions \(XMLA\)](#).

Handling Processing Errors

The [ErrorConfiguration](#) property of the **Process** command lets you specify how to handle errors encountered while processing an object. For example, while processing a dimension, Analysis Services encounters a duplicate value in the key column of the key attribute. Because attribute keys must be unique, Analysis Services discards the duplicate records. Based on the [KeyDuplicate](#) property of **ErrorConfiguration**, Analysis Services could:

- Ignore the error and continue processing the dimension.
- Return a message that states Analysis Services encountered a duplicate key and continue processing.

There are many similar conditions for which **ErrorConfiguration** provides options during a **Process** command.

Managing Writeback Tables

If the **Process** command encounters a write-enabled partition, or a cube or measure group for such a partition, that is not already fully processed, a writeback table may not already exist for that partition. The [WritebackTableCreation](#) property of the **Process** command determines whether Analysis Services should create a writeback table.

Examples

Description

The following example fully processes the Adventure Works DW Multidimensional 2012 sample Analysis Services database.

Code

```
<Process xmlns="http://schemas.microsoft.com/analysiservices/2003/engine">
  <Object>
    <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>
  </Object>
  <Type>ProcessFull</Type>
  <WriteBackTableCreation>UseExisting</WriteBackTableCreation>
</Process>
```

Description

The following example incrementally processes the **Internet_Sales_2004** partition in the **Internet Sales** measure group of the **Adventure Works DW** cube in the Adventure Works DW Multidimensional 2012 sample Analysis Services database. The **Process** command is adding aggregations for order dates later than December 31, 2006 to the partition by using an out-of-line query binding in the **Bindings** property of the **Process** command to retrieve the fact table rows from which to generate aggregations to add to the partition.

Code

```
<Process ProcessAffectedObjects="true"
xmlns="http://schemas.microsoft.com/analysiservices/2003/engine">

  <Object>

    <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

    <CubeID>Adventure Works DW</CubeID>

    <MeasureGroupID>Fact Internet Sales 1</MeasureGroupID>

    <PartitionID>Internet_Sales_2006</PartitionID>

  </Object>

  <Bindings>

    <Binding>

      <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

      <CubeID>Adventure Works DW</CubeID>

      <MeasureGroupID>Fact Internet Sales 1</MeasureGroupID>

      <PartitionID>Internet_Sales_2006</PartitionID>

      <Source xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="QueryBinding">

        <DataSourceID>Adventure Works DW</DataSourceID>

        <QueryDefinition>

          SELECT

            [dbo].[FactInternetSales].[ProductKey],

            [dbo].[FactInternetSales].[OrderDateKey],

            [dbo].[FactInternetSales].[DueDateKey],

            [dbo].[FactInternetSales].[ShipDateKey],

            [dbo].[FactInternetSales].[CustomerKey],

            [dbo].[FactInternetSales].[PromotionKey],

            [dbo].[FactInternetSales].[CurrencyKey],

            [dbo].[FactInternetSales].[SalesTerritoryKey],

            [dbo].[FactInternetSales].[SalesOrderNumber],
```

```

        [dbo].[FactInternetSales].[SalesOrderLineNumber],
        [dbo].[FactInternetSales].[RevisionNumber],
        [dbo].[FactInternetSales].[OrderQuantity],
        [dbo].[FactInternetSales].[UnitPrice],
        [dbo].[FactInternetSales].[ExtendedAmount],
        [dbo].[FactInternetSales].[UnitPriceDiscountPct],
        [dbo].[FactInternetSales].[DiscountAmount],
        [dbo].[FactInternetSales].[ProductStandardCost],
        [dbo].[FactInternetSales].[TotalProductCost],
        [dbo].[FactInternetSales].[SalesAmount],
        [dbo].[FactInternetSales].[TaxAmt],
        [dbo].[FactInternetSales].[Freight],
        [dbo].[FactInternetSales].[CarrierTrackingNumber],
        [dbo].[FactInternetSales].[CustomerPONumber]
    FROM [dbo].[FactInternetSales]
    WHERE OrderDateKey > '1280'
</QueryDefinition>
</Source>
</Binding>
</Bindings>
<Type>ProcessAdd</Type>
<WriteBackTableCreation>UseExisting</WriteBackTableCreation>
</Process>

```

Merging Partitions (XMLA)

If partitions have the same aggregation design and structure, you can merge the partition by using the [MergePartitions](#) command in XML for Analysis (XMLA). Merging partitions is an important action to perform when you manage partitions, especially those partitions that contain historical data partitioned by date.

For example, a financial cube may use two partitions:

- One partition represents financial data for the current year, using real-time relational OLAP (ROLAP) storage settings for performance.
- Another partition contains financial data for previous years, using multidimensional OLAP (MOLAP) storage settings for storage.

Both partitions use different storage settings, but use the same aggregation design. Instead of processing the cube across years of historical data at the end of the year, you can instead use the **MergePartitions** command to merge the partition for the current year into the partition for previous years. This preserves the aggregation data without requiring a potentially time-consuming full processing of the cube.

Specifying Partitions to Merge

When the **MergePartitions** command runs, the aggregation data stored in the source partitions specified in the [Source](#) property is added to the target partition specified in the [Target](#) property.



Note

The **Source** property can contain more than one partition object reference. However, the **Target** property cannot.

To be successfully merged, the partitions specified in both the **Source** and **Target** must be contained by the same measure group and use the same aggregation design. Otherwise, an error occurs.

The partitions specified in the **Source** are deleted after the **MergePartitions** command is successfully completed.

Examples

Description

The following example merges all the partitions in the **Customer Counts** measure group of the **Adventure Works** cube in the **Adventure Works DW** sample Microsoft SQL Server Analysis Services database into the **Customers_2004** partition.

Code

```
<MergePartitions xmlns="http://schemas.microsoft.com/analysisisservices/2003/engine">

  <Sources>

    <Source>

      <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

      <CubeID>Adventure Works DW</CubeID>

      <MeasureGroupID>Fact Internet Sales 1</MeasureGroupID>

      <PartitionID>Internet_Sales_2001</PartitionID>

    </Source>

    <Source>

      <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

      <CubeID>Adventure Works DW</CubeID>

      <MeasureGroupID>Fact Internet Sales 1</MeasureGroupID>

      <PartitionID>Internet_Sales_2002</PartitionID>

    </Source>

  </Sources>

</MergePartitions>
```

```

<Source>

  <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

  <CubeID>Adventure Works DW</CubeID>

  <MeasureGroupID>Fact Internet Sales 1</MeasureGroupID>

  <PartitionID>Internet_Sales_2003</PartitionID>

</Source>

</Sources>

<Target>

  <DatabaseID>Adventure Works DW Multidimensional 2012</DatabaseID>

  <CubeID>Adventure Works DW</CubeID>

  <MeasureGroupID>Fact Internet Sales 1</MeasureGroupID>

  <PartitionID>Internet_Sales_2004</PartitionID>

</Target>

</MergePartitions>

```

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Designing Aggregations (XMLA)

Aggregation designs are associated with the partitions of a particular measure group to make sure that the partitions use the same structure when storing aggregations. Using the same storage structure for partitions lets you to easily define partitions that can be later merged using the [MergePartitions](#) command. For more information about aggregation designs, see [Using XML for Analysis in Analysis Services \(XMLA\)](#).

To define aggregations for an aggregation design, you can use the [DesignAggregations](#) command in XML for Analysis (XMLA). The **DesignAggregations** command has properties that identify which aggregation design to use as a reference and how to control the design process based upon that reference. Using the **DesignAggregations** command and its properties, you can design aggregations iteratively or in batch, and then view the resulting design statistics to evaluate the design process.

Specifying an Aggregation Design

The [Object](#) property of the **DesignAggregations** command must contain an object reference to an existing aggregation design. The object reference contains a database identifier, cube identifier, measure group identifier, and aggregation design identifier. If the aggregation design does not already exist, an error occurs.

Controlling the Design Process

You can use the following properties of the **DesignAggregations** command to control the algorithm used to define aggregations for the aggregation design:

- The [Steps](#) property determines how many iterations the **DesignAggregations** command should take before it returns control to the client application.
- The [Time](#) property determines how many milliseconds the **DesignAggregations** command should take before it returns control to the client application.
- The [Optimization](#) property determines the estimated percentage of performance improvement the **DesignAggregations** command should try to achieve. If you are iteratively designing aggregations, you only have to send this property on the first command.
- The [Storage](#) property determines the estimated amount of disk storage, in bytes, used by the **DesignAggregations** command. If you are iteratively designing aggregations, you only have to send this property on the first command.
- The [Materialize](#) property determines whether the **DesignAggregations** command should create the aggregations defined during the design process. If you are iteratively designing aggregations, this property should be set to false until you are ready to save the designed aggregations. When set to true, the current design process ends and the defined aggregations are added to the specified aggregation design.

Specifying Queries

The **DesignAggregations** command supports usage-based optimization command by including one or more **Query** elements in the [Queries](#) property. The **Queries** property can contain one or more [Query](#) elements. If the **Queries** property does not contain any **Query** elements, the aggregation design specified in the **Object** element uses a default structure that contains a general set of aggregations. This general set of aggregations is designed to meet the criteria specified in the **Optimization** and **Storage** properties of the **DesignAggregations** command.

Each **Query** element represents a goal query that the design process uses to define aggregations that target the most frequently used queries. You can either specify your own goal queries, or you can use the information stored by an instance of Microsoft SQL Server Analysis Services in the query log to retrieve information about the most frequently used queries. The Usage-Based Optimization Wizard uses the query log to retrieve goal queries based on time, usage, or a specified user when it sends a **DesignAggregations** command. For more information, see [Usage-Based Optimization Wizard F1 Help \(SSAS\)](#).

If you are iteratively designing aggregations, you only have to pass goal queries in the first **DesignAggregations** command because the Analysis Services instance stores these goal queries and uses these queries during subsequent **DesignAggregations** commands. After you pass goal queries in the first **DesignAggregations** command of an iterative process, any subsequent **DesignAggregations** command that contains goal queries in the **Queries** property generates an error.

The **Query** element contains a comma-delimited value that contains the following arguments: Frequency, Dataset[, Dataset...]

Frequency

A weighting factor that corresponds to the number of times that the query has previously been executed. If the **Query** element represents a new query, the Frequency value represents

the weighting factor used by the design process to evaluate the query. As the frequency value becomes larger, the weight that is put on the query during the design process increases.

Dataset

A numeric string that specifies which attributes from a dimension are to be included in the query. This string must have the same number of characters as the number of attributes in the dimension. Zero (0) indicates that the attribute in the specified ordinal position is not included in the query for the specified dimension, while one (1) indicates that the attribute in the specified ordinal position is included in the query for the specified dimension.

For example, the string "011" would refer to a query involving a dimension with three attributes, from which the second and third attributes are included in the query.



Note

Some attributes are excluded from consideration in the dataset. For more information about excluded attributes, see [Query Element \(XMLA\)](#).

Each dimension in the measure group that contains the aggregation design is represented by a Dataset value in the **Query** element. The order of Dataset values must match the order of dimensions included in the measure group.

Designing Aggregations Using Iterative or Batch Processes

You can use the **DesignAggregations** command as part of an iterative process or a batch process, depending on the interactivity required by the design process.

Designing Aggregations Using an Iterative Process

To iteratively design aggregations, you send multiple **DesignAggregations** commands to provide fine control over the design process. The Aggregation Design Wizard uses this same approach to provide fine control over the design process. For more information, see [Aggregation Design Wizard F1 Help \(SSAS\)](#).



Note

An explicit session is required to iteratively design aggregations. For more information about explicit sessions, see [Managing Connections and Sessions \(XMLA\)](#).

To start the iterative process, you first send a **DesignAggregations** command that contains the following information:

- The **Storage** and **Optimization** property values on which the whole design process is targeted.
- The **Steps** and **Time** property values on which the first step of the design process is limited.
- If you want usage-based optimization, the **Queries** property that contains the goal queries on which the whole design process is targeted.
- The **Materialize** property set to false. Setting this property to false indicates that the design process does not save the defined aggregations to the aggregation design when the command is completed.

When the first **DesignAggregations** command finishes, the command returns a rowset that contains design statistics. You can evaluate these design statistics to determine whether the design process should continue or whether the design process is finished. If the process should continue, you then send another **DesignAggregations** command that contains the **Steps** and **Time** values with which this step of the design process is limited. You evaluate the resulting statistics and then determine whether the design process should continue. This iterative process of sending **DesignAggregations** commands and evaluating the results continues until you reach your goals and have a appropriate set of aggregations defined.

After you have reached the set of aggregations that you want, you send one final **DesignAggregations** command. This final **DesignAggregations** command should have its **Steps** property set to 1 and its **Materialize** property set to true. By using these settings, this final **DesignAggregations** command completes the design process and saves the defined aggregation to the aggregation design.

Designing Aggregations Using a Batch Process

You can also design aggregations in a batch process by sending a single **DesignAggregations** command that contains the **Steps**, **Time**, **Storage**, and **Optimization** property values on which the whole design process is targeted and limited. If you want usage-based optimization, the goal queries on which the design process is targeted should also be included in the **Queries** property. Also make sure that the **Materialize** property is set to true, so that the design process saves the defined aggregations to the aggregation design when the command finishes.

You can design aggregations using a batch process in either an implicit or explicit session. For more information about implicit and explicit sessions, see [Managing Connections and Sessions \(XMLA\)](#).

Returning Design Statistics

When the **DesignAggregations** command returns control to the client application, the command returns a rowset that contains a single row representing the design statistics for the command. The rowset contains the columns listed in the following table.

Column	Data type	Description
Steps	Integer	The number of steps taken by the command before returning control to the client application.
Time	Long integer	The number of milliseconds taken by the command before returning control to the client application.
Optimization	Double	The estimated percentage of performance improvement

Column	Data type	Description
		achieved by the command before returning control to the client application.
Storage	Long integer	The estimated number of bytes taken by the command before returning control to the client application.
Aggregations	Long integer	The number of aggregations defined by the command before returning control to the client application.
LastStep	Boolean	Indicates whether the data in the rowset represents the last step in the design process. If the Materialize property of the command was set to true, the value of this column is set to true.

You can use the design statistics that are contained in the rowset returned after each **DesignAggregations** command in both iterative and batch design. In iterative design, you can use the design statistics to determine and display progress. When you are designing aggregations in batch, you can use the design statistics to determine the number of aggregations created by the command.

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Backing Up, Restoring, and Synchronizing Databases (XMLA)

In XML for Analysis, there are three commands that back up, restore, and synchronize databases:

- The [Backup](#) command backs up a Microsoft SQL Server Analysis Services database using an Analysis Services backup file (.abf), as described in the section, Backing Up Databases.
- The [Restore](#) command restores an Analysis Services database from an .abf file, as described in the section, Restoring Databases.
- The [Synchronize](#) command synchronizes one Analysis Services database with the data and metadata of another database, as described in the section, Synchronizing Databases.

Backing Up Databases

As mentioned earlier, the **Backup** command backs up a specified Analysis Services database to a backup file. The **Backup** command has various properties that let you specify the database to be backed up, the backup file to use, how to back up security definitions, and the remote partitions to be backed up.

noteDXDOC112778PADS **Security Note**

The Analysis Services service account must have permission to write to the backup location specified for each file. Also, the user must have one of the following roles: administrator role on the Analysis Services instance, or a member of a database role with Full Control (Administrator) permissions on the database to be backed up.

Specifying the Database and Backup File

To specify the database to be backed up, you set the [Object](#) property of the **Backup** command. The **Object** property must contain an object identifier for a database, or an error occurs.

To specify the file that is to be created and used by the backup process, you set the [File](#) property of the **Backup** command. The **File** property should be set to a UNC path and file name for the backup file to be created.

Besides specifying which file to use for backup, you can set the following options for the specified backup file:

- If you set the [AllowOverwrite](#) property to true, the **Backup** command overwrites the backup file if the specified file already exists. If you set the **AllowOverwrite** property to false, an error occurs if the specified backup file already exists.
- If you set the [ApplyCompression](#) property to true, the backup file is compressed after the file is created.
- If you set the [Password](#) property to any non-blank value, the backup file is encrypted by using the specified password.

noteDXDOC112778PADS **Security Note**

If **ApplyCompression** and **Password** properties are not specified, the backup file stores user names and passwords that are contained in connection strings in clear text. Data that is stored in clear text may be retrieved. For increased security, use the **ApplyCompression** and **Password** settings to both compress and encrypt the backup file.

Backing Up Security Settings

The [Security](#) property determines whether the **Backup** command backs up the security definitions, such as roles and permissions, defined on an Analysis Services database. The **Security** property also determines whether the backup file includes the Windows user accounts and groups defined as members of the security definitions.

The value of the **Security** property is limited to one of the strings listed in the following table.

Value	Description
SkipMembership	Include security definitions, but exclude membership information, in the backup file.
CopyAll	Include security definitions and membership information in the backup file.
IgnoreSecurity	Exclude security definitions from the backup file.

Backing Up Remote Partitions

To back up remote partitions in the Analysis Services database, you set the [BackupRemotePartitions](#) property of the **Backup** command to true. This setting causes the **Backup** command to create a remote backup file for each remote data source that is used to store remote partitions for the database.

For each remote data source to be backed up, you can specify its corresponding backup file by including a [Location](#) element in the [Locations](#) property of the **Backup** command. The **Location** element should have its **File** property set to the UNC path and file name of the remote backup file, and its [DataSourceID](#) property set to the identifier of the remote data source defined in the database.

Restoring Databases

The **Restore** command restores a specified Analysis Services database from a backup file. The **Restore** command has various properties that let you specify the database to restore, the backup file to use, how to restore security definitions, the remote partitions to be stored, and the relocation relational OLAP (ROLAP) objects.

noteDXDOC112778PADS **Security Note**

For each backup file, the user who runs the restore command must have permission to read from the backup location specified for each file. To restore an Analysis Services database that is not installed on the server, the user must also be a member of the server role for that Analysis Services instance. To overwrite an Analysis Services database, the user must have one of the following roles: a member of the server role for the Analysis Services instance or a member of a database role with Full Control (Administrator) permissions on the database to be restored.

Note

After restoring an existing database, the user who restored the database might lose access to the restored database. This loss of access can occur if, at the time that the backup was performed, the user was not a member of the server role or was not a member of the database role with Full Control (Administrator) permissions.

Specifying the Database and Backup File

The **DatabaseName** property of the **Restore** command must contain an object identifier for a database, or an error occurs. If the specified database already exists, the **AllowOverwrite** property determines whether the existing database is overwritten. If the **AllowOverwrite** property is set to false and the specified database already exists, an error occurs.

You should set the **File** property of the **Restore** command to a UNC path and file name for the backup file to be restored to the specified database. You can also set the **Password** property for the specified backup file. If the **Password** property is set to any non-blank value, the backup file is decrypted by using the specified password. If the backup file was not encrypted, or if the specified password does not match the password used to encrypt the backup file, an error occurs.

Restoring Security Settings

The **Security** property determines whether the **Restore** command restores the security definitions, such as roles and permissions, defined on an Analysis Services database. The **Security** property also determines whether the **Restore** command includes the Windows user accounts and groups defined as members of the security definitions as part of the restore process.

The value of this element is limited to one of the strings listed in the following table.

Value	Description
SkipMembership	Include security definitions, but exclude membership information, in the database.
CopyAll	Include security definitions and membership information in the database.
IgnoreSecurity	Exclude security definitions from the database.

Restoring Remote Partitions

For each remote backup file created during a previous **Backup** command, you can restore its associated remote partition by including a **Location** element in the **Locations** property of the **Restore** command. The [DataSourceType](#) property for each **Location** element must be excluded or explicitly set to Remote.

For each specified **Location** element, the Analysis Services instance contacts the remote data source specified in the **DataSourceID** property to restore the partitions defined in the remote backup file specified in the **File** property. Besides the **DataSourceID** and **File** properties, the following properties are available for each **Location** element used to restore a remote partition:

- To override the connection string for the remote data source specified in **DataSourceID**, you can set the **ConnectionString** property of the **Location** element to a different connection string. The **Restore** command will then use the connection string that is contained in the

ConnectionString property. If **ConnectionString** is not specified, the **Restore** command uses the connection string stored in the backup file for the specified remote data source. You can use the **ConnectionString** setting to move a remote partition to a different remote instance. However, you cannot use the **ConnectionString** setting to restore a remote partition to the same instance that contains the restored database. In other words, you cannot use the **ConnectionString** property to make a remote partition into a local partition.

- For each original folder used to store the remote partitions on the remote data source, you can specify a [Folder](#) element to indicate the new folder in which to restore all the remote partitions stored in the original folder. If a **Folder** element is not specified, the **Restore** command uses the original folders specified for the remote partitions that are contained in the remote backup file.

Relocating ROLAP Objects

The **Restore** command cannot restore aggregations or data for objects that use ROLAP storage because such information is stored in tables on an underlying relational data source. However, the metadata for ROLAP objects can be restored. To restore the metadata for ROLAP object, the **Restore** command re-creates the table structure on a relational data source.

You can use the **Location** element in a **Restore** command to relocate ROLAP objects. For each **Location** element used to relocate a data source, the **DataSourceType** property must be explicitly set to Local. You also have to set the **ConnectionString** property of the **Location** element to the connection string of the new location. During the restore, the **Restore** command will replace the connection string for the data source identified by the **DataSourceID** property of the **Location** element with the value of the **ConnectionString** property of the **Location** element.

Synchronizing Databases

The **Synchronize** command synchronizes the data and metadata of a specified Analysis Services database with another database. The **Synchronize** command has various properties that let you specify the source database, how to synchronize security definitions, the remote partitions to be synchronized, and the synchronization of ROLAP objects.

Note

The **Synchronize** command can be executed only by server administrators and database administrators. Both the source and destination database must have the same database compatibility level.

Specifying the Source Database

The [Source](#) property of the **Synchronize** command contains two properties, **ConnectionString** and **Object**. The **ConnectionString** property contains the connection string of the instance that contains the source database, and the **Object** property contains the object identifier for the source database.

The destination database is the current database for the session in which the **Synchronize** command runs.

If the **ApplyCompression** property of the **Synchronize** command is set to true, the information sent from the source database to the destination database is compressed before being sent.

Synchronizing Security Settings

The [SynchronizeSecurity](#) property determines whether the **Synchronize** command synchronizes the security definitions, such as roles and permissions, defined on the source database. The **SynchronizeSecurity** property also determines whether the **Synchronize** command includes the Windows user accounts and groups defined as members of the security definitions.

The value of this element is limited to one of the strings listed in the following table.

Value	Description
SkipMembership	Include security definitions, but exclude membership information, in the destination database.
CopyAll	Include security definitions and membership information in the destination database.
IgnoreSecurity	Exclude security definitions from the destination database.

Synchronizing Remote Partitions

For each remote data source that exists on the source database, you can synchronize each associated remote partition by including a **Location** element in the **Locations** property of the **Synchronize** command. For each **Location** element, the **DataSourceType** property must be excluded or explicitly set to Remote.

To define and connect to a remote data source in the destination database, the **Synchronize** command uses the connection string defined in the **ConnectionString** property of the **Location** element. The **Synchronize** command then uses the **DataSourceID** property of the **Location** element to identify which remote partitions to synchronize. The **Synchronize** command synchronizes the remote partitions on the remote data source specified in the **DataSourceID** property on the source database with the remote data source specified in the **DataSourceID** property on the destination database.

For each original folder used to store the remote partitions on the remote data source on the source database, you can also specify a **Folder** element in the **Location** element. The **Folder** element indicates the new folder for the destination database in which to synchronize all the remote partitions stored in the original folder on the remote data source. If a **Folder** element is not specified, the Synchronize command uses the original folders specified for remote partitions that are contained in the source database.

Synchronizing ROLAP Objects

The **Synchronize** command cannot synchronize aggregations or data for objects that use ROLAP storage because such information is stored in tables on an underlying relational data source. However, the metadata for ROLAP objects can be synchronized. To synchronize the metadata, the **Synchronize** command recreates the table structure on a relational data source.

You can use the **Location** element in a Synchronize command to synchronize ROLAP objects. For each **Location** element used to relocate a data source, the **DataSourceType** property must be explicitly set to Local. . You also have to set the **ConnectionString** property of the **Location** element to the connection string of the new location. During synchronization, the **Synchronize** command will replace the connection string for the data source identified by the **DataSourceID** property of the **Location** element with the value of the **ConnectionString** property of the **Location** element.

See Also

[Managing Backing Up and Restoring \(Analysis Services\)](#)

[Restore Element \(XMLA\)](#)

[Synchronize Element \(XMLA\)](#)

[Backing Up and Restoring an Analysis Services Database](#)

Inserting, Updating, and Dropping Members (XMLA)

You can use the [Insert](#), [Update](#), and [Drop](#) commands in XML for Analysis (XMLA) to respectively insert, update, or delete members from a write-enabled dimension. For more information about write-enabled dimensions, see [Using XML for Analysis in Analysis Services \(XMLA\)](#).

Inserting New Members

The **Insert** command inserts new members into specified attributes in a write-enabled dimension.

Before constructing the **Insert** command, you should have the following information available for the new members to be inserted:

- The dimension in which to insert the new members.
- The dimension attribute in which to insert the new members.
- The names of the new members, including any applicable translations for the name.
- The keys of the new members. If an attribute uses a composite key, the key may require multiple values.
- Values for any applicable attribute properties that are not implemented as other attributes within the dimension. Such attribute properties include unary operations, translations, custom rollups, custom rollup properties, and skipped levels.

The **Insert** command takes only two properties:

- The [Object](#) property, which contains an object reference for the dimension in which the members are to be inserted. The object reference contains the database identifier, cube identifier, and dimension identifier for the dimension.

- The [Attributes](#) property, which contains one or more [Attribute](#) elements to identify the attributes in which members are to be inserted. Each **Attribute** element identifies an attribute and provides the name, value, translations, unary operator, custom rollup, custom rollup properties, and skipped levels for a single member to be added to the identified attribute.



Note

All properties for the **Attribute** element must be included. Otherwise, an error may occur.

Updating Existing Members

The **Update** command updates existing members in specified attributes, based on relationships with other members in other attributes, in a write-enabled dimension. The **Update** command can move members to other levels in hierarchies contained by the dimension, and can be used to restructure parent-child hierarchies defined by parent attributes.

Before constructing the **Update** command, you should have the following information available for the members to be updated:

- The dimension in which to update existing members.
- The dimension attributes in which to update existing members.
- The keys of the existing members. If an attribute uses a composite key, the key may require multiple values.
- Values for any applicable attribute properties that are not implemented as other attributes within the dimension. Such attribute properties include unary operations, translations, custom rollups, custom rollup properties, and skipped levels.

The **Update** command takes only three required properties:

- The **Object** property, which contains an object reference for the dimension in which the members are to be updated. The object reference contains the database identifier, cube identifier, and dimension identifier for the dimension.
- The **Attributes** property, which contains one or more **Attribute** elements to identify the attributes in which members are to be updated. The **Attribute** element identifies an attribute and provides the name, value, translations, unary operator, custom rollup, custom rollup properties, and skipped levels for a single member updated for the identified attribute.



Note

All properties for the **Attribute** element must be included. Otherwise, an error may occur.

- The [Where](#) property, which contains one or more **Attribute** elements that constrain the attributes in which members are to be updated. The **Where** property is crucial to limiting an **Update** command to specific instances of a member. If the **Where** property is not specified, all instances of a given member are updated. For example, there are three customers for whom you want to change the city name from Redmond to Bellevue. To change the city

name, you must provide a **Where** property that identifies the three members in the Customer attribute for which the members in the City attribute should be changed. If you do not provide this **Where** property, every customer whose city name is currently Redmond would have the city name of Bellevue after the **Update** command runs.



Note

With the exception of new members, the **Update** command can only update attribute key values for attributes not included in the **Where** clause. For example, the city name cannot be updated when a customer is updated; otherwise, the city name is changed for all customers.

Updating Members in Parent Attributes

To support parent attributes, the **Update** command the optional [MoveWithDescendants](#) `MoveWithDescendants` properties. Setting the **MoveWithDescendants** property to true indicates that the descendants of the parent member should also be moved with the parent member when the identifier of that parent member changes. If this value is set to false, moving a parent member causes the immediate descendants of that parent member to be promoted to the level in which the parent member formerly resided.

When updating members in a parent attribute, the **Update** command cannot update members in other attributes.

Dropping Existing Members

Before constructing the **Drop** command, you should have the following information available for the members to be dropped:

- The dimension in which to drop existing members.
- The dimension attributes in which to drop existing members.
- The keys of the existing members to be dropped. If an attribute uses a composite key, the key may require multiple values.

The **Drop** command takes only two required properties:

- The **Object** property, which contains an object reference for the dimension in which the members are to be dropped. The object reference contains the database identifier, cube identifier, and dimension identifier for the dimension.
- The **Where** property, which contains one or more **Attribute** elements to constrain the attributes in which members are to be deleted. The **Where** property is crucial to limiting a **Drop** command to specific instances of a member. If the **Where** command is not specified, all instances of a given member are dropped. For example, there are three customers that you want to drop from Redmond. To drop these customers, you must provide a **Where** property that identifies the three members in the Customer attribute to be removed and the Redmond member of the City attribute from which the three customers are to be removed. If the **Where** property only specifies the Redmond member of the City attribute, every customer associated with Redmond would be dropped by the **Drop** command. If the **Where**

property only specifies the three members in the Customer attribute, the three customers would be deleted entirely by the **Drop** command.



Note

The **Attribute** elements included in a **Drop** command must contain only the **AttributeName** and **Keys** properties. Otherwise, an error may occur.

Dropping Members in Parent Attributes

Setting the [DeleteWithDescendants](#) property indicates that the descendants of a parent member should also be deleted with the parent member. If this value is set to false, the immediate descendants of the parent member are instead promoted to the level in which the parent member formerly resided.

noteDXDOC112778PADS **Security Note**

A user needs only to have delete permissions for the parent member to delete both the parent member and its descendants. A user does not need delete permissions on the descendants.

See Also

[Drop Element \(XMLA\)](#)

[Insert Element \(XMLA\)](#)

[Update Element \(XMLA\)](#)

[Defining and Identifying Objects \(XMLA\)](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Updating Cells (XMLA)

You can use the [UpdateCells](#) command to change the value of one or more cells in a cube enabled for cube writeback. Microsoft SQL Server Analysis Services stores the updated information in a separate writeback table for each partition that contains cells to be updated.



Note

The **UpdateCells** command does not support allocations during cube writeback. To use allocated writeback, you should use the [Statement](#) command to send a Multidimensional Expressions (MDX) UPDATE statement. For more information, see [Using XML for Analysis in Analysis Services \(XMLA\)](#).

Specifying Cells

The [Cell](#) property of the **UpdateCells** command contains the cells to be updated. You identify each cell in the **Cell** property using that cell's ordinal number. Conceptually, Analysis Services numbers cells in a cube as if the cube were a p-dimensional array, where p is the number of axes. Cells are addressed in row-major order. The following illustration shows the formula for calculating the ordinal number of a cell.

If axis k has U_k members, the ordinal number of a cell whose tuple ordinals are $(S_0, S_1, S_2, \dots, S_{p-1})$ is

$$\sum_{i=0}^{p-1} S_i \times E_i \text{ where } E_0 = 1 \text{ and } E_i = \prod_{k=0}^{i-1} U_k$$

Σ represents the sum of the terms in the series and Π the product.

Once you know a cell's ordinal number, you can indicate the intended value of the cell in the [Value](#) property of the [Cell](#) property.

See Also

[Update Element \(XMLA\)](#)

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Managing Caches (XMLA)

You can use the [ClearCache](#) command in XML for Analysis (XMLA) to clear the cache of a specified dimension or partition. Clearing the cache forces Microsoft SQL Server Analysis Services to rebuild the cache for that object.

Specifying Objects

The [Object](#) property of the **ClearCache** command can contain an object reference only for one of the following objects. An error occurs if an object reference is for an object other than one of following objects:

Database

Clears the cache for all dimensions and partitions contained in the database.

Dimension

Clears the cache for the specified dimension.

Cube

Clears the cache for all partitions contained in the measure groups for the cube.

Measure group

Clears the cache for all partitions contained in the measure group.

Partition

Clears the cache for the specified partition.

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Monitoring Traces (XMLA)

You can use the [Subscribe](#) command in XML for Analysis (XMLA) to monitor an existing trace defined on an instance of Microsoft SQL Server Analysis Services. The **Subscribe** command returns the results of a trace as a rowset.

Specifying a Trace

The [Object](#) property of the **Subscribe** command must contain an object reference to either an Analysis Services instance or a trace on an Analysis Services instance. If the **Object** property is not specified, or a trace identifier is not specified in the **Object** property, the **Subscribe** command monitors the default session trace for the explicit session specified in the SOAP header for the command.

Returning Results

The **Subscribe** command returns a rowset containing the trace events captured by the specified trace. The **Subscribe** command returns trace results until the command is canceled by the [Cancel](#) command.

The rowset contains the columns listed in the following table.

Column	Data type	Description
EventClass	Integer	The event class of the event received by the trace.
EventSubclass	Long integer	The event subclass of the event received by the trace.
CurrentTime	Datetime	The time at which the event started, when available. For filtering, expected formats are 'YYYY-MM-DD' and 'YYYY-MM-DD HH:MM:SS'.
StartTime	Datetime	The time at which the event started, when available. For filtering, expected formats are 'YYYY-MM-DD' and 'YYYY-MM-DD HH:MM:SS'.
EndTime	Datetime	The time at which the event ended, when available. For filtering, expected formats are 'YYYY-MM-DD' and 'YYYY-MM-DD HH:MM:SS'. This column is not populated for event classes that describe

Column	Data type	Description
		the start of a process or action.
Duration	Long integer	The amount of total time (in milliseconds) elapsed for the event.
CPUTime	Long integer	The amount of processor time (in milliseconds) elapsed for the event.
JobID	Long integer	The job identifier for the process.
SessionID	String	The identifier of the session for which the event occurred.
SessionType	String	The type of the session for which the event occurred.
ProgressTotal	Long integer	The total number or amount of progress reported by the event.
IntegerData	Long integer	Integer data associated with the event. The contents of this column depend on the event class and subclass of the event.
ObjectID	String	The identifier of the object for which the event occurred.
ObjectType	String	The type of the object specified in ObjectName.
ObjectName	String	The name of the object for which the event occurred.
ObjectPath	String	The hierarchical path of the object for which the event occurred. The path is represented as a comma-delimited string of object identifiers for the parents of the object specified in ObjectName.
ObjectReference	String	The XML representation of the object reference for the object specified in ObjectName.

Column	Data type	Description										
NestLevel	Integer	The level of the transaction for which the event occurred.										
NumSegments	Long integer	The number of data segments affected or accessed by the command for which the event occurred.										
Severity	Integer	The severity level of an exception for the event. The column can contain one of the following values:<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Success</td></tr><tr><td>1</td><td>Information</td></tr><tr><td>2</td><td>Warning</td></tr><tr><td>3</td><td>Error</td></tr></table>	Value	Description	0	Success	1	Information	2	Warning	3	Error
Value	Description											
0	Success											
1	Information											
2	Warning											
3	Error											
Success	Boolean	Indicates whether a command succeeded or failed.										
Error	Long integer	The error number of the event, if applicable.										
ConnectionID	String	The identifier of the connection for which the event occurred.										
DatabaseName	String	The name of the database for which the event occurred.										
NTUserName	String	The Windows user name of the user associated with the event.										
NTDomainName	String	The Windows domain of the user associated with the event.										
ClientHostName	String	The name of the computer on which the client application is running. This column is										

Column	Data type	Description
		populated with the values passed by the client application.
ClientProcessID	Long integer	The process identifier of the client application.
ApplicationName	String	The name of the client application that created the connection to the Analysis Services instance. This column is populated with the values passed by the client application, rather than the displayed name of the program.
NTCanonicalUserName	String	The Windows canonical user name of the user associated with the event.
SPID	String	The server process ID (SPID) of the session for which the event occurred. The value of this column directly corresponds to the session ID specified in the SOAP header of the XMLA message for which the event occurred.
TextData	String	The text data associated with the event. The contents of this column depend on the event class and subclass of the event.
ServerName	String	The name of the Analysis Services instance for which the event occurred.
RequestParameters	String	The parameters of the parameterized query or XMLA command for which the event occurred.
RequestProperties	String	The properties of the XMLA method for which the event occurred.

See Also

[Using XML for Analysis in Analysis Services \(XMLA\)](#)

Extending OLAP functionality

As a programmer, you can extend Analysis Services by writing assemblies, personalized extensions, and stored procedures that provide functionality you want to use and repurpose in multiple database applications. Assemblies are used to extend multidimensional models functionality by adding new procedures and functions to the MDX language or by means of the personalization addin.

Stored procedures can be used to call external routines, simplifying Analysis Services database development and implementation by allowing common code to be developed once and stored in a single location. Stored procedures can be used to add business functionality to your applications that is not provided by the native functionality of MDX.

Personalizations are custom objects that you add to a cube to provide a behavior that varies by user. Personalizations are not permanent objects in the cube, but are objects that the client application applies dynamically during the user's session. Examples include changing the currency of a monetary value depending on the person accessing the data, providing individualized KPIs, or a targeted suggestion list for regular customers who purchase online.

In this Section

[Extending OLAP through personalizations](#)

[Analysis Services Personalization Extensions](#)

[Defining Stored Procedures](#)

Extending OLAP through personalizations

Microsoft SQL Server 2012 Analysis Services (SSAS) supplies many intrinsic functions for use with the Multidimensional Expressions (MDX) and Data Mining Extensions (DMX) languages. These functions are designed to accomplish everything from standard statistical calculations to traversing members in a hierarchy. However, as with any other complex and robust product, there is always the need to extend the functionality of such a product further.

Therefore, Analysis Services provides you with the ability to add assemblies and personalized extensions to an instance of the service, in order to complete your business needs whenever the standard functionality is not enough.

Assemblies

Assemblies enable you to extend the business functionality of MDX and DMX. You build the functionality that you want into a library, such as a dynamic link library (DLL), then add the

library as an assembly to an instance of Analysis Services or to an Analysis Services database. The public methods in the library are then exposed as user-defined functions to MDX and DMX expressions, procedures, calculations, actions, and client applications.

Personalized Extensions

SQL Server Analysis Services personalization extensions are the foundation of the idea of implementing a plug-in architecture. Analysis Services personalization extensions are a simple and elegant modification to the existing managed assembly architecture and are exposed throughout the Analysis Services **N:Microsoft.AnalysisServices.AdomdServer** object model, Multidimensional Expressions (MDX) syntax, and schema rowsets.

See Also

[Assemblies \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Personalization Extensions](#)

Analysis Services Personalization Extensions

SQL Server Analysis Services personalization extensions are the foundation of the idea of implementing a plug-in architecture. In a plug-in architecture, you can develop new cube objects and functionality dynamically and share them easily with other developers. As such, Analysis Services personalization extensions provide the functionality that makes it possible to achieve the following:

- **Dynamic design and deployment** Immediately after you design and deploy Analysis Services personalization extensions, users have access to the objects and functionality at the start of the next user session.
- **Interface independence** Regardless of the interface that you use to create the Analysis Services personalization extensions, users can use any interface to access the objects and functionality.
- **Session context** Analysis Services personalization extensions are not permanent objects in the existing infrastructure and do not require the cube to be reprocessed. They become exposed and created for the user at the time that the user connects to the database, and remain available for the length of that user session.
- **Rapid distribution** Share Analysis Services personalization extensions with other software developers without having to go into detailed specifications about where or how to find this extended functionality.

Analysis Services personalization extensions have many uses. For example, your company has sales that involve different currencies. You create a calculated member that returns the consolidated sales in the local currency of the person who is accessing the cube. You create this member as a personalization extension. You then share this calculated member to a group of users. Once shared, those users have immediate access to the calculated member as soon as they connect to the server. They have access even if they are not using the same interface as the one that was used to create the calculated member.

Analysis Services personalization extensions are a simple and elegant modification to the existing managed assembly architecture and are exposed throughout the Analysis Services **N:Microsoft.AnalysisServices.AdomdServer** object model, Multidimensional Expressions (MDX) syntax, and schema rowsets.

Logical Architecture

The architecture for Analysis Services personalization extensions is based on the managed assembly architecture and the following four basic elements:

The [PlugInAttribute] custom attribute

When starting the service, Analysis Services loads the required assemblies and determines which classes have the

T:Microsoft.AnalysisServices.AdomdServer.PlugInAttribute custom attribute.



Note

The .NET Framework defines custom attributes as a way to describe your code and affect run-time behavior. For more information, see the topic, "[Attributes Overview](#)," in the .NET Framework Developer's Guide on MSDN.

For all classes with the **T:Microsoft.AnalysisServices.AdomdServer.PlugInAttribute** custom attribute, Analysis Services invokes their default constructors. Invoking all the constructors at startup provides a common location from which to build new objects and that is independent of any user activity.

In addition to building a small cache of information about authoring and managing personalization extensions, the class constructor typically subscribes to the

E:Microsoft.AnalysisServices.AdomdServer.Server.SessionOpened and

E:Microsoft.AnalysisServices.AdomdServer.Server.SessionClosing events. Failing to subscribe to these events may cause the class to be inappropriately marked for cleanup by the common language runtime (CLR) garbage collector.

Session context

For those objects that are based on personalization extensions, Analysis Services creates an execution environment during the client session and dynamically builds most of those objects in this environment. Like any other CLR assembly, this execution environment also has access to other functions and stored procedures. When the user session ends, Analysis Services removes the dynamically created objects and closes the execution environment.

Events

Object creation is triggered by the session events **On-Cube-OpenedCubeOpened** and **On-Cube-ClosingCubeClosing**.

Communication between the client and the server occurs through specific events. These events make the client aware of the situations that lead to the client's objects being built. The client's environment is dynamically created using two sets of events: session events and cube events.

Session events are associated with the server object. When a client logs on to a server,

Analysis Services creates a session and triggers the

E:Microsoft.AnalysisServices.AdomdServer.Server.SessionOpened event. When a client ends the session on the server, Analysis Services triggers the

E:Microsoft.AnalysisServices.AdomdServer.Server.SessionClosing event.

Cube events are associated with the connection object. Connecting to a cube triggers the

E:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.CubeOpened event. Closing the connection to a cube, by either closing the cube or by changing to a different cube, triggers a

E:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.CubeClosing event.

Traceability and error handling

All activity is traceable by using SQL Server Profiler. Unhandled errors are reported to the Windows event log.

All object authoring and management is independent of this architecture and is the sole responsibility of the developers of the objects.

Infrastructure Foundations

Analysis Services personalization extensions are based on existing components. The following is a summary of enhancements and improvements that provide the personalization extensions functionality.

Assemblies

The custom attribute, **T:Microsoft.AnalysisServices.AdomdServer.PluginAttribute**, can be added to your custom assemblies to identify Analysis Services personalization extensions classes.

Changes to the AdomdServer Object Model

The following objects in the **N:Microsoft.AnalysisServices.AdomdServer** object model have been enhanced or added to the model.

New AdomdConnection Class

The **T:Microsoft.AnalysisServices.AdomdServer.AdomdConnection** class is new and exposes several personalization extensions through both properties and events.

Properties

- **P:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.SessionID**, a read-only string value representing the session Id of the current connection.
- **P:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.ClientCulture**, a read-only reference to the client culture associated with current session.
- **P:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.User**, a read-only reference to the identity interface representing the current user.

Events

- **E:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.CubeOpened**

- **E:Microsoft.AnalysisServices.AdomdServer.AdomdConnection.CubeClosing**

New Properties in the Context class

The **T:Microsoft.AnalysisServices.AdomdServer.Context** class has two new properties:

- **P:Microsoft.AnalysisServices.AdomdServer.Context.Server**, a read-only reference to the new server object.
- **P:Microsoft.AnalysisServices.AdomdServer.Context.CurrentConnection**, a read-only reference to the new **T:Microsoft.AnalysisServices.AdomdServer.AdomdConnection** object.

New Server class

The **T:Microsoft.AnalysisServices.AdomdServer.Server** class is new and exposes several personalization extensions through both class properties and events.

Properties

- **P:Microsoft.AnalysisServices.AdomdServer.Server.Name**, a read-only string value representing the server name.
- **P:Microsoft.AnalysisServices.AdomdServer.Server.Culture**, A read-only reference to the global culture associated with the server.

Events

- **E:Microsoft.AnalysisServices.AdomdServer.Server.SessionOpened**
- **E:Microsoft.AnalysisServices.AdomdServer.Server.SessionClosing**

AdomdCommand class

The **T:Microsoft.AnalysisServices.AdomdServer.AdomdCommand** class now supports of the following MDX commands:

- [CREATE MEMBER Statement \(MDX\)](#)
- [UPDATE MEMBER Statement \(MDX\)](#)
- [DROP MEMBER Statement \(MDX\)](#)
- [CREATE SET Statement \(MDX\)](#)
- [DROP SET Statement \(MDX\)](#)
- [CREATE KPI Statement \(MDX\)](#)
- [DROP KPI Statement \(MDX\)](#)

MDX extensions and enhancements

The CREATE MEMBER command is enhanced with the **caption** property, the **display_folder** property, and the **associated_measure_group** property.

The UPDATE MEMBER command is added to avoid member re-creation when an update is needed with the consequent loss of precedence in solving calculations. Updates cannot change the scope of the calculated member, move the calculated member to a different parent, or define a different **solveorder**.

The CREATE SET command is enhanced with the **caption** property, the **display_folder** property, and the new **STATIC | DYNAMIC** keyword. *Static* means that set is evaluated only at creation time. *Dynamic* means that the set is evaluated every time that the set is used in a query. The default value is **STATIC** if a keyword is omitted.

CREATE KPI and DROP KPI commands are added to the MDX syntax. KPIs can be created dynamically from any MDX script.

Schema Rowsets extensions

On MDSHEMA_MEMBERS scope column is added. Scope values are as follows:
MDMEMBER_SCOPE_GLOBAL=1, MDMEMBER_SCOPE_SESSION=2.

On MDSHEMA_SETS set_evaluation_context column is added. Set evaluation context values are as follows: MDSET_RESOLUTION_STATIC = 1, MDSET_RESOLUTION_DYNAMIC = 2.

On MDSHEMA_KPIS scope column is added. Scope values are as follows:
MDKPI_SCOPE_GLOBAL=1, MDKPI_SCOPE_SESSION=2.

Defining Stored Procedures

You can use stored procedures to call external routines from Microsoft SQL Server Analysis Services. You can write an external routines called by a stored procedure in any common language runtime (CLR) language, such as C, C++, C#, Visual Basic, or Visual Basic .NET. A stored procedure can be created once and called from many contexts, such as other stored procedures, calculated measures, or client applications. Stored procedures simplify Analysis Services database development and implementation by allowing common code to be developed once and stored in a single location. Stored procedures can be used to add business functionality to your applications that is not provided by the native functionality of MDX.

This section provides the information necessary to understand, design, and implement stored procedures.

Topic	Description
Assemblies (Analysis Services - Multidimensional Data)	Describes how to design assemblies for use with Analysis Services.
Creating Stored Procedures	Describes how to create assemblies for Analysis Services.
Calling Stored Procedures	Provides information on how to use assemblies in Analysis Services.
Accessing Query Context in Stored Procedures	Describes how to access scope and context information with assemblies.
Setting Security for Stored Procedures	Describes how to configure security for assemblies in Analysis Services.

Topic	Description
Debugging Stored Procedures	Describes how to debug assemblies in Analysis Services.

See Also

[Stored Procedures](#)

Designing Stored Procedures

Both the administrative object model Analysis Management Objects (AMO) and the client oriented object model Microsoft ActiveX® Data Objects (Multidimensional) (ADO MD) are available in stored procedures.

Stored procedures must be in scope (either the server or the database) to be visible at the Multidimensional Expressions (MDX) level to be called. However, once a stored procedure is invoked, its scope is not limited to actions under its parent. A stored procedure may make changes or modifications anywhere on the server, subject only to the security limitations of the user process that invokes it or to the limitations of the transaction in which it is operating.

Server scope procedures are available in all contexts on the server. Database scope stored procedures are visible only in the database context of the database in which they are defined.

As with any MDX function, stored procedure must be resolved before an MDX session can continue; stored procedures lock MDX sessions while executing. Unless a specific reason exists to halt an MDX session pending user interaction, then user interactions (such as dialog boxes) are discouraged.

Dependent Assemblies

All dependent assemblies must be loaded into an instance of Analysis Services to be found by the common language runtime (CLR). Analysis Services stores the dependent assemblies in the same folder as the main assembly, so the CLR automatically resolves all function references to functions in those assemblies.

See Also

[Defining Stored Procedures](#)

[Working with Stored Procedures](#)

Creating Stored Procedures

All stored procedures must be associated with a common language runtime (CLR) or Component Object Model (COM) class in order to be used. The class must be installed on the server — usually in the form of a Microsoft ActiveX® dynamic link library (DLL) — and registered as an assembly on the server or in an Analysis Services database.

Stored procedures are registered on a server or on a database. Server stored procedures can be called from any query context. Database stored procedures can only be accessed if the database context is the database under which the stored procedure is defined. If functions in one assembly call functions in a different assembly, you must register both assemblies in the same context (server or database). For a server or a deployed Microsoft SQL Server Analysis Services database on a server, you can use SQL Server Management Studio to register an assembly. For an Analysis Services project, you can use Analysis Services Designer to register an assembly in the project.

noteDXDOC112778PADS **Security Note**

COM assemblies might pose a security risk. Due to this risk and other considerations, COM assemblies were deprecated in SQL Server 2008 Analysis Services (SSAS). COM assemblies might not be supported in future releases.

Registering a Server Assembly

In Object Explorer in SQL Server Management Studio, server assemblies are listed in the Assemblies folder under an instance of Analysis Services. Server assemblies can contain both .NET (CLR) assemblies and COM libraries.

To create a server assembly

1. Expand the instance of Analysis Services in Object Explorer, right-click the **Assemblies** folder, and then click **New Assembly**. This displays the **Register Server Assembly** dialog box.
2. For **Type** specify the type of assembly:
 - For a managed code (CLR) DLL, specify .NET Assembly.
 - For a native code (COM) DLL, specify COM DLL.
3. For **File name**, specify the DLL containing the stored procedures.
4. For **Assembly name**, specify a name for the assembly.
5. If this is a debug build of the library that you are going to use to debug stored procedures, select the **Include debug information** check box. For more information about debugging stored procedures, see [Defining Stored Procedures](#).
6. You can click **OK** to register the assembly immediately, or on the dialog box toolbar, you can click a command on the **Script** menu to script the registration action to a query window, a file, or the Clipboard.

After you register a server assembly, you can configure it by right-clicking the assembly in Object Explorer and then clicking **Properties**.

Registering a Database Assembly on the Server

In Object Explorer in SQL Server Management Studio, database assemblies are listed in the Assemblies folder under an Analysis Services database. Database assemblies can contain both .NET (CLR) assemblies and COM libraries.

To create a database assembly on a server

1. Expand the instance the Analysis Services database in Object Explorer, right-click the **Assemblies** folder, and then click **New Assembly**. This displays the **Register Database Assembly** dialog box.
2. For **Type** specify the type of assembly:
 - For a managed code (CLR) DLL, specify .NET Assembly.
 - For a native code (COM) DLL, specify COM DLL.
3. For **File name**, specify the DLL containing the stored procedures.
4. For **Assembly name**, specify a name for the assembly.
5. If this is a debug build of the library that you are going to use to debug stored procedures, select the **Include debug information** check box. For more information about debugging stored procedures, see [Debugging Stored Procedures](#).
6. You can click **OK** to register the assembly immediately, or on the dialog box toolbar, you can click a command on the **Script** menu to script the registration action to a query window, a file, or the Clipboard.

After you register a database assembly, you can configure it by right-clicking the assembly in Object Explorer and then clicking **Properties**.

Registering a Database Assembly in a Project

In Solution Explorer in SQL Server Data Tools (SSDT), database assemblies are listed in the Assemblies folder under an Analysis Services project. Database assemblies can contain both .NET (CLR) assemblies and COM libraries.

To create a database assembly in an Analysis Service project

1. Expand the instance the Analysis Services database in Object Explorer, right-click the **Assemblies** folder, and then click **New Assembly Reference**. This displays the **Add Reference** dialog box. The **.NET** tab of the **Add Reference** dialog box lists existing .NET (CLR) assemblies, while the **Projects** tab lists projects.
2. You can click an existing component or project and then click **Add** to add it to the Analysis Services project. To add a reference to a COM DLL, click the **Browse** tab to find the file. The **Selected projects and components** list shows the name, type, version, and location for each component that you are adding to the project.
3. When you are finished selecting components to add, click **OK** to add them to the Analysis Services project.

Script Format For an Assembly

Registering a .NET assembly is fairly simple. A .NET assembly is added to a database in binary format using the following format:

```
<Create>

  <ObjectDefinition>

    <Assembly>

      <Files>
```



```

<File>

  <Name>filename</Name>

  <Type>filetype</Type>

  <Data>

    <Block>binarydatablock</Block>

    <Block>binarydatablock</Block>

    ...

  </Data>

</File>

</Files>

<PermissionSet>PermissionSet</PermissionSet>

</Assembly>

<ObjectDefinition>

</Create>

```

See Also

[Stored Procedures](#)

[Working with Stored Procedures](#)

Calling Stored Procedures

Stored procedures can be called on the server or from client application. In either case, stored procedures always run on the server, either the context of the server or of a database. There are no special permissions required to execute a stored procedure. Once a stored procedure is added by an assembly to the server or database context, any user can execute the stored procedure as long as the role for the user permits the actions performed by the stored procedure.

Calling a stored procedure in MDX is done in the same manner as calling an intrinsic MDX function. For a stored procedure that takes no parameters, the name of the procedure and an empty pair of parentheses are used, as shown here:

```
MyStoredProcedure()
```

If the stored procedure takes one or more parameters, then the parameters are supplied, in order, separated by commas. The following example demonstrates a sample stored procedure with three parameters:

```
MyStoredProcedure("Parameter1", 2, 800)
```

Calling Stored Procedures in MDX Queries

In all MDX queries, the stored procedure must return the syntactically correct type required by an MDX expression. If a stored procedure does not return the correct type, an MDX error occurs.

The following examples demonstrate stored procedures that return a set, a member, and the result of a mathematical operation.

Returning a Set

The following examples implement a stored procedure, called `MySproc`, that returns a set. In the first example, `MySproc` returns the set directly in the `SELECT` expression. In the second two examples, `MySproc` returns the set as an argument for the `Crossjoin` and `DrilldownLevel` functions.

```
SELECT MySetProcedure(a,b,c) ON 0 FROM Sales
```

```
SELECT Crossjoin(MySetProcedure(a,b,c)) ON 0 FROM Sales
```

```
SELECT DrilldownLevel(MySetProcedure(a,b,c)) ON 0 FROM Sales
```

Returning a Member

The following example shows a function `MySproc` function that returns a member:

```
SELECT Descendants(MySproc(a,b,c),3) ON 0 FROM Sales
```

Returning the Result of a Math Operation

```
SELECT Country.Members on 0, MySproc(Measures.Sales) ON 1 FROM Sales
```

Calling Stored Procedures with the Call Statement

Stored procedures can be called outside of the context of an MDX query using the MDX **Call** statement.

You can use this method to either instantiate the side effects of a stored query or for the application to get the results of a stored query. A common use of the **Call** statement would be to use Analysis Management Objects (AMO) to perform administrative functions that do not have a return result. For example, the following command calls a stored procedure:

```
Call MyStoredProcedure(a,b,c)
```

The only supported type returned from stored procedure in a **Call** statement is a rowset. The serialization for a rowset is defined by XML for Analysis. If a stored procedure in a **Call** statement returns any other type, it is ignored and not returned in XML to the calling application. For more information about XML for Analysis rowsets, see, XML for Analysis Schema Rowsets.

If a stored procedure returns a .NET rowset, Analysis Services converts the result on the server to an XML for Analysis rowset. The XML for Analysis rowset is always returned by a stored procedure in the **Call** function. If a dataset contains features that cannot be expressed in the XML for Analysis rowset, a failure results.

Procedures that return void values (for example, subroutines in Visual Basic) can also be employed with the `CALL` keyword. If, for example, you wanted to use the function `MyVoidFunction()` in an MDX statement, the following syntax would be employed:

```
CALL(MyVoidFunction)
```

See Also

[Defining Stored Procedures](#)

[Working with Stored Procedures](#)

Accessing Query Context in Stored Procedures

The execution context of a stored procedure is available within the code of the stored procedure as the **Context** object of the ADOMD.NET server object model. This is a read-only context and cannot be modified by the stored procedure. The following properties are available on this object.

Property	Type	Description
CurrentCube	Cube	The cube for the current query context.
CurrentDatabaseName	String	The identifier of the current database.
CurrentConnection	Connection	A reference to the connection object in the current context.
Pass	Integer	The pass number for the current context.

The **Context** object exists when the Multidimensional Expressions (MDX) object model is used in a stored procedure. It is not available when the MDX object model is used on a client. The **Context** object is not explicitly passed to or returned by the stored procedure. It is available during the execution of the stored procedure.

See Also

[Defining Stored Procedures](#)

[Working with Stored Procedures](#)

Setting Security for Stored Procedures

Security for stored procedures is set with the **PermissionSet** property on a stored procedure for an instance of Analysis Services (server level), an Analysis Services database, or an Analysis Services project.

See Also

[Defining Stored Procedures](#)

[Working with Stored Procedures](#)

Debugging Stored Procedures

Analysis Services stored procedures are actually CLR or COM libraries (normally DLLs) that are written in C# (or any other CLR or COM language). Therefore, debugging a stored procedure is

much like debugging any other application in the Visual Studio debugging environment. You debug stored procedures in the Visual Studio development environment using the integrated debugging functions. These allow you to stop at procedure locations, inspect memory and register values, change variables, observe message traffic and get a close look at how your code works.

Procedures

► To debug a stored procedure

1. Open the project used to create the DLL in Visual Studio.
2. Create breakpoints in the method or function corresponding to the procedure you want to debug.
3. Use Visual Studio to create a debug build of a stored procedure DLL.
4. Deploy the DLL to the server. For more information about deploying the DLL to the server, see [Defining Stored Procedures](#).
5. You need an application that calls the stored procedure that you want to test. If you do not have one ready, you can use the MDX Query Editor in SQL Server Management Studio to create an MDX query that calls the stored procedure that you want to test.
6. In Visual Studio, attach to the Analysis Services process (Msmdsrv.exe).
 - a. From the **Debug** menu, choose **Attach to Process**.
 - b. In the **Attach to Process** dialog box, select **Show processes from all users**.
 - c. In the **Available Processes** list, in the **Process** column, click **Msmdsrv.exe**. If there is more than one instance of Analysis Services running on the server, you need to identify the process by the ID of the instance you want to use.
 - d. In the **Attach to** text box, make sure that the appropriate program type is selected. For a CLR DLL, click **Select**, then click **Debug these code types**, then click **Managed**, then click **OK**. For a COM DLL, click **Select**, then click **Debug these code types**, then click **Native**, then click **OK**.
 - e. Click **Attach**.
7. In Analysis Services, invoke the program or MDX script that calls the stored procedure. The debugger breaks when it reaches a line containing a breakpoint. You can evaluate variables in the watch window, view locals, and step through the code.

If you have problems debugging a library, make sure that the corresponding program database (PDB) file was copied to the deployment location on the server. If this file was not copied during registration or deployment, you must copy it manually to the same location as the DLL. For native code (COM DLL), the PDB file resides in the \debug subdirectory. For managed code (CLR DLL), it resides in the \WINDEBUG subdirectory.

See Also

[Stored Procedures](#)

[Working with Stored Procedures](#)

Analysis Services OLE DB Provider

The Analysis Services OLE DB Provider is an interface for applications interacting with Microsoft Analysis Services. It is used to build client applications that interact with multidimensional data. This provider also provides methods for online and offline data mining analysis of multidimensional data and relational data, and is included as part of Analysis Services. It can be redistributed by third-party client applications.

The Analysis Services OLE DB Provider 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, the Analysis Services OLE DB Provider 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 server running the instance of Analysis Services.

See Also

[Analysis Services Data Access Interfaces \(Analysis Services - Multidimensional Data\)](#)