SAS Publishing

SAS® 9.1 Open Metadata Interface

Reference

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS® 9.1 Open Metadata Interface: Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.1 Open Metadata Interface: Reference

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, January 2004

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/pubs** or call 1-800-727-3228.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

SAS 9.1 Open Metadata Interface: Reference

This guide describes the SAS 9.1 Open Metadata Interface and contains all of the information you need to create and manage metadata using the SAS 9.1 Open Metadata Interface. This guide describes

- the SAS Metadata Model
- namespaces
- metadata−related, security, and server−control methods
- call interfaces
- client requirements.

In addition, hierarchy and association diagrams depict the relationships between application−related metadata types.

The SAS Open Metadata Architecture is a client−server architecture that uses XML as its transport language. If you are unfamiliar with XML, see the W3C XML Specifications at

```
www.w3.org/TR/1998/REC-xml-19980210
```

Previous Page | Next Page | Top of Page

SAS 9.1 Open Metadata Interface: Reference

Prerequisites

To get the most out of this document, you should be familiar with

- a client−application programming language, such as Visual Basic, C++, or Java
- a software development environment, such as Microsoft's Visual Studio or SAS AppDev Studio
- Extensible Markup Language (XML) 1.0.

See the **SAS 9.1 Metadata Server: Setup Guide** for details about the software that must be installed to support the SAS Open Metadata Architecture.

Previous Page | Next Page | Top of Page

Introduction

What Is the SAS Open Metadata Architecture?

The SAS Open Metadata Architecture is a general−purpose metadata management facility that provides common metadata services to SAS applications. Using the metadata architecture, separate SAS applications can exchange metadata, which makes it easier for these applications to work together. The metadata architecture also saves development effort because applications no longer have to maintain their own metadata facilities.

The metadata architecture includes an application metadata model, a repository metadata model, an application programming interface, and a server.

The application metadata model, called the SAS Metadata Model, provides classes and objects that define different types of application metadata. It models associations between individual metadata objects, it uses inheritance of attributes and methods to effect common behaviors, and it uses subclassing to extend behaviors.

The repository metadata model is a special−purpose model that defines metadata types for repositories and repository managers. Like the SAS Metadata Model, it uses classes, objects, inheritance, and subclassing. However, its purpose is to support repository queries and impose controls on the objects contained in the repositories which the repository objects describe.

The SAS Open Metadata Interface provides methods for reading and writing metadata objects that are stored in repositories. These same methods can be used to maintain the repositories, although this is a secondary task. Another set of methods is provided for administering repositories and the server.

The SAS Metadata Server is a multiuser server that surfaces metadata from one or more repositories via the SAS Open Metadata Interface. The SAS Metadata Server uses the Integrated Object Model (IOM) from SAS Integration Technologies. IOM provides distributed object interfaces to Base SAS software features and enables you to use industry−standard languages, programming tools, and communication protocols to develop client programs that access base features on IOM servers. Its purpose is to provide a central, shared location for accessing metadata.

SAS Open Metadata Architecture

What Can I Do with the SAS Open Metadata Interface?

The SAS Open Metadata Interface enables you to read and write the metadata of applications that comply with the metadata architecture. It also enables you to maintain repositories and to control the SAS Metadata Server, but these tasks are secondary. For the most part, you will use the SAS Open Metadata Interface to read or write the metadata of applications. For example, you can write clients that:

- return a list of data stores that contain a metadata item that you specify, such as a column name of Salary
- export the metadata from one application to another application, so that the metadata can be analyzed
- export the metadata for a data store so that the data store can be accessed by more than one application.

How Does the SAS Open Metadata Architecture Work?

To use the metadata architecture, you write SAS Open Metadata Interface client applications in Java, Visual Basic, C++, or SAS.

Java, Visual Basic, and C++ clients include the appropriate object libraries and method calls that are required to connect to the metadata server and to

- access the metadata within a repository
- access the metadata that defines a repository
- control the SAS Metadata Server.

SAS provides PROC METADATA to access the metadata within a repository. It provides PROC METAOPERATE to access metadata that defines a repository and to control the SAS Metadata Server.

Important Concepts

metadata type
: specifies a template that models the metadata for a particular kind of object. For example, the metadata type Column models the metadata for a SAS table column, and the metadata type RepositoryBase models the metadata for a repository.

namespace
: specifies a group of related metadata types and their properties. Names are used to partition metadata into different contexts. The SAS Open Metadata Interface is shipped with two namespaces defined: SAS and REPOS. The SAS namespace contains metadata types for application elements such as tables and columns. The REPOS namespace contains metadata types for repositories.

metadata object
: specifies an instance of a metadata type, such as the metadata for a particular data store or the metadata for a particular metadata repository. All SAS Open Metadata Interface clients use the same methods to read or write a metadata object, whether the object defines an application element or a metadata repository.

Accessing Application Metadata

As shown in the next figure, a SAS Open Metadata Interface client that accesses application metadata

- specifies the SAS namespace in order to access the metadata types for application elements such as tables and columns
- connects to the SAS Metadata Server via a communication standard that is appropriate for the client and the IOM–based server, such as COM/DCOM or CORBA
- uses SAS Open Metadata Interface method calls to access instances of the metadata types that are stored in SAS metadata repositories.

Accessing Metadata Defined in the SAS Namespace

For general information about writing SAS Open Metadata Interface clients, see Client Requirements. For a description of the metadata types in the SAS namespace, see SAS Namespace Types. Details about the methods used to read or write metadata objects are provided in IOMI Class.

Creating Repositories

Before you can read and write metadata objects, at least one repository must be registered in the server's repository manager. The information stored in the repository manager tells the server how to access the repository. You can register repositories by using SAS applications such as SAS Management Console or by writing a Java, Visual Basic, or C++ client that connects to the SAS Metadata Server and issues SAS Open Metadata Interface method calls to register a repository. The information flow necessary for a SAS Open Metadata Interface client to register a repository is similar to that shown in Accessing Metadata Defined in the SAS Namespace, except the REPOS namespace is specified instead of the SAS namespace. The REPOS namespace includes the metadata types for repositories.

For information about registering repositories using SAS Management Console, see the Help for the product. General information about writing SAS Open Metadata Interface clients is provided in Client Requirements. For a description of the metadata types in the REPOS namespace, see REPOS Namespace Types. Details about the methods used to read and write repository objects are provided in IOMI Class.

Administering Repositories and the SAS Metadata Server

A metadata server must be running before any client can access metadata repositories. At many sites, a server administrator starts the SAS Metadata Server and SAS Open Metadata Interface clients simply connect to that server. However, there are times when you might want to refresh the server to change certain configuration or invocation options, or pause and resume specific repositories or the repository manager to temporarily change their state, for example, in preparation for a backup. For this purpose, the SAS Open Metadata Interface provides IServer class methods. The IServer class also provides a Stop server method. The Stop method is the recommended method for stopping the server. Using operating system commands to stop the server process can corrupt your repositories.

You can write a Java, Visual Basic, or C++ client that issues IServer class methods. Or you can use a SAS Open Metadata Interface client such as SAS Management Console or PROC METAOPERATE.

A user must have administrative user status in order to issue IServer class methods, except Status.

For information about controlling access to repositories and the metadata server using SAS Management Console, see the Help for the product. For information about PROC METAOPERATE, see the METAOPERATE Procedure. General information about writing SAS Open Metadata Interface clients is provided in Client Requirements. For reference information on IServer methods, see IServer Class.

Security

The SAS Metadata Server supports a variety of authentication providers to determine who can access the server and uses an authorization facility to control user access to metadata on the server. Only users who have been granted unrestricted user status have unrestricted access to metadata on the server. Only users who have been granted either unrestricted user status or administrative user status can create and delete repositories, modify a repository's registrations, change the state of a repository, and register users. For more information, see "Server Administrative Privileges" and "Controlling User Access to Metadata" in the **SAS 9.1 Metadata Server: Setup Guide.**

Authorization Facility

Authorization processes are insulated from metadata−related processes in the SAS Metadata Server. Authorization decisions are made by an authorization facility. The authorization facility provides an interface

for querying authorization metadata that is on the metadata server and returns authorization decisions based on rules that are stored in the metadata. The SAS Metadata Server consumes this interface to make queries regarding read and write access to metadata and enforces the decisions that are returned by the authorization facility. That is, it is not necessary for SAS Open Metadata Interface clients to write queries or to enforce authorization decisions regarding read and write access to metadata. However, SAS Open Metadata Interface clients can use the interface to request authorization decisions on other types of metadata access, for example, to return decisions regarding administrative access. In addition, clients can use the interface to request authorization decisions on the data represented by the SAS metadata. Applications that use the authorization facility to return authorization decisions on user−defined actions must provide their own authorization enforcement.

The query interface consists of a set of methods that are available in the SAS Open Metadata Interface ISecurity class. For more information, see ISecurity Class.

To learn more information about the authorization facility, see "Understanding the Authorization Facility" in the **SAS 9.1 Metadata Server: Setup Guide**.

Previous Page | Next Page | Top of Page

Client Requirements

Overview

A SAS Open Metadata Interface client is a Java, Visual Basic, C++, or SAS program that connects to the SAS Metadata Server and uses the SAS Open Metadata Interface to issue method calls. You can create three types of clients:

- clients that read and write application metadata objects
- clients that read and write repository objects
- clients that control access to repositories and the SAS Metadata Server.

Most of your clients will read and write application metadata objects. Repository objects enable you to register repositories in the repository manager and to perform tasks, such as defining repository dependencies and enabling or disabling repository auditing.

A client that controls access to a repository or the SAS Metadata Server can temporarily override the repository or repository manager's access state. For example, the client can pause the repository and the repository manager to a read−only or offline state in preparation for a backup. A client can also refresh a server to change certain server configuration and invocation options and stop the server. Shutting down a server by a means other than the Stop server method can damage the contents of your repositories.

Developing Clients

A SAS Open Metadata Interface client establishes communication with the SAS Metadata Server and issues method calls. Therefore, the metadata server must be running.

Java, Visual Basic, and C++ clients include the appropriate object libraries to connect to the server and to issue method calls. The requirements for connecting to the server are described in Connecting to the SAS Metadata Server. The call interfaces for Java, Visual Basic, and C++ clients are described in Call Interfaces.

SAS provides the following clients to simplify use of the SAS Open Metadata Interface:

PROC METADATA
: connects to the metadata server and enables you to issue XML−formatted method calls to create, update, and query metadata objects in SAS metadata repositories.

PROC METAOPERATE
: connects to the metadata server and enables you to pause and resume repositories and the repository manager in order to temporarily change their state; refresh, stop, and get the status of the server; and delete a repository.

SAS Management Console
: provides a graphical user interface for connecting to the metadata server, registering repositories, creating global metadata, including metadata access controls, and managing the server.

SAS Java Metadata Interface
: provides a Java object interface to the SAS Metadata Server. The interface provides a way to access SAS metadata repositories through the use of client Java objects that represent server metadata.

Foundation Services Information Service Interface
: provides a generic interface for interacting with heterogeneous repositories, including SAS Metadata Repositories, Lightweight Directory Access Protocol (LDAP) repositories, and WebDAV repositories, from an application. Using Information Service methods, a client can submit a single query that searches all available repository sources and returns the results in a "smart object" that

provides a uniform interface to common data elements such as the object's name, description, and type. The smart objects hide repository and model details. The interface then uses the SAS Java Metadata Interface to launch further queries.

The information SAS clients need to connect to the SAS Metadata Server is described in Server Connection Properties. SAS also provides system options that enable you to specify default connection and resource options. For more information about these options, see SAS Metadata System Options.

PROC METADATA accepts an XML input string formatted for the *inMetadata* parameter of the DoRequest method. For information about how to format the XML string, see DoRequest method. PROC METADATA and PROC METAOPERATE are documented in this guide. For information about SAS Management Console, see the **SAS 9.1 Management Console: User's Guide**. For information about the SAS Java Metadata Interface, see the **SAS Java Metadata Interface: User's Guide**. Class documentation for the Foundation Services Information Services interface is available on the SAS web at http://bip.pc.sas.com/api/com/sas/services/information/package−summary.html.

Using Server Output

The SAS Metadata Server returns output in the form of an XML string. Clients can interact with this output by using their favorite XML parser (DOM or SAX) or by using a set of Java classes provided by SAS. To learn more about the Java metadata classes, see the **SAS 9.1 Java Metadata Interface: User's Guide**.

Previous Page | Next Page | Top of Page

Connecting to the SAS Metadata Server

Overview

Before it can issue a method call, a SAS Open Metadata Interface client must connect to the SAS Metadata Server and instantiate objects for method parameters. This section describes the requirements for connecting to the metadata server.

The SAS Metadata Server uses the Integrated Object Model (IOM) provided by SAS Integration Technologies. IOM provides distributed object interfaces to Base SAS software features and enables you to use industry−standard languages, programming tools, and communication protocols to develop client programs that access Base SAS features on IOM servers. The interfaces supported by IOM servers are described in "Connecting Clients to IOM Servers" in the **SAS Integration Technologies Library** at `support.sas.com/rnd/itech/library/`. The SAS Metadata Server supports the IOM's COM/DCOM and CORBA interfaces.

- Windows clients connecting to a SAS Metadata Server use the IOM Bridge for COM. The IOM Bridge for COM supports connections to servers running in Windows and non−Windows operating environments.
- Java clients connect to the SAS Metadata Server by using the IOM Bridge for Java in all operating environments.

To connect to the metadata server, a client must

- invoke the appropriate IOM interface for the programming environment
- supply server connection properties
- reference the SAS Open Metadata Interface method class that is appropriate to the task.

You will need to establish a connection to the server each time you issue a method call. It is recommended that you create a connection object that can be referenced in individual method calls.

Server Connection Properties

The following server connection properties are required by the SAS Metadata Server. Optional properties are described in the SAS Integration Technologies documentation.

host
: The IP address of the machine hosting the SAS Metadata Server.

port=*XXXX*
: The TCP port to which the SAS Metadata Server listens for requests and that clients will use to connect to the server. The *XXXX* must be a unique number from 0−64K.

username
: A valid username on the host machine.

password
: The password corresponding to username on the host machine.

factory number
: The SAS Metadata Server identifier for Java clients. This property must have the value "2887e7d7−4780−11d4−879f−00c04f38f0db".

server identifier
: The SAS Metadata Server identifier for Windows clients. This property must have the value "SASOMI.OMI".

protocol
The network protocol for Java clients. The valid value is "bridge".

SAS Open Metadata Interface Method Classes

The SAS Open Metadata Interface provides methods in three classes:

- The *IOMI class* contains metadata access methods. In the Visual Basic programming environment, this class is referred to as the *OMI class*. A client references the IOMI class to read and write both application and repository metadata objects.
- The *IServer class* contains methods that pause and resume repositories or the repository manager, and refresh, get the status of, or stop the SAS Metadata Server. A client references the IServer class to control access to repositories and the SAS Metadata Server.
- The *ISecurity class* contains methods for requesting authorization decisions from the SAS Open Metadata Architecture authorization facility. A client references the ISecurity class to request user−defined authorization decisions on access controls that are stored as metadata.

For more information about the IOMI methods, see IOMI Class. For more information about the IServer methods, see IServer Class. For more information about the ISecurity methods, see ISecurity Class.

Previous Page | Next Page | Top of Page

Call Interfaces

Introduction

Each metadata−related method takes a set of parameters that drive the behavior of the method. The SAS Open Metadata Interface supports two ways of passing a method and its parameters to the metadata server:

- A client can define object variables for each of the parameters and issue the method call directly from within the client. This involves explicitly setting the name and data type of each parameter in the client, then referencing the variables in the method call. This is referred to as the "standard interface."
- A client can define object variables for and issue a generic DoRequest method in the client and pass metadata−related method calls to the server in a coded XML string. This XML string is passed to the server in the DoRequest method's *inMetadata* parameter.

Although object variables must still be defined for its parameters, the DoRequest method provides a program−independent way of submitting metadata−related method calls. The DoRequest method also provides performance benefits in that it enables the client to submit multiple methods in the input XML string, simply by enclosing the string in a <Multiple_Requests> XML element. The format of this input XML string is described in DoRequest.

Comparison of the Standard Interface and the DoRequest Method

The following Java code fragment illustrates the steps required to issue a GetRepositories method call using the standard interface. (The GetRepositories method returns a list of the repositories registered in a repository manager.)

```
private void getRepositories() {
     int returnCodeFromOMI = -999;
     int flags = 0;
     String options = " ";
     StringHolder returnInfoFromOMI = new org.omg.CORBA.StringHolder();
   try {
     returnCodeFromOMI = connection.GetRepositories(returnInfoFromOMI,flags,options);
     System.out.println("returnCodeFromOMI = " + returnCodeFromOMI);
     System.out.println("returnInfoFromOMI = " = returnInfoFromOMI.value);
   }
}
```

The GetRepositories method has three parameters −− *Repositories*, *Flags*, and *Options* −− in addition to a return code. When using the standard interface, the client explicitly sets the variable name and data type for each parameter and also issues the GetRepositories call.

The following code fragment depicts a GetRepositories call that is issued via the DoRequest method.

```
private void getRepositories() {
int returnCodeFromOMI = -999;
String inMetadata = "<GetRepositories>
                        <Repositories/>
                        <Flags>0</Flags>
                        <Options/>
                    </GetRepositories>";
StringHolder outMetadata = new org.omg.CORBA.StringHolder();
   try {
```

```
      returnCodeFromOMI = connection.DoRequest(inMetadata, outMetadata);
      System.out.println("returnCodeFromOMI = " + returnCodeFromOMI);
      System.out.println("outMetadata = " + returnInfoFromOMI.value);
   }
}
```

When using the DoRequest method, the client sets names and data types for the DoRequest method parameters (*inMetadata* and *outMetadata*), and submits the GetRepositories method and all of its parameters in an XML string in the *inMetadata* parameter.

The following is an example of using the <Multiple_Requests> XML element in a DoRequest call. The request issues a GetRepositories method and a GetTypes method. The GetTypes method lists the metadata types defined for a namespace.

```
private void getRepositories() {
int returnCodeFromOMI = -999;
String inMetadata = "<Multiple_Requests>
                        <GetRepositories>
                          <Repositories/>
                          <Flags>0</Flags>
                          <Options/>
                       </GetRepositories>
                        <GetTypes>
                          <Types/>
                          <NS>SAS</NS>
                          <Flags>0</Flags>
                          <Options/>
                        </GetTypes>
                     </Multiple_Requests>";
StringHolder outMetadata = new org.omg.CORBA.StringHolder();
   try {
     returnCodeFromOMI = connection.DoRequest(inMetadata, outMetadata);
     System.out.println("returnCodeFromOMI = " + returnCodeFromOMI);
     System.out.println("outMetadata = " + returnInfoFromOMI.value);
   }
}
```

Data Type Requirements

The following tables summarize data type requirements for metadata–related methods in the supported programming environments:

- Java IOMI Class Signature Summary Table
- Visual Basic OMI Class Signature Summary Table
- Visual C++ IOMI Class Signature Summary Table

IOMI Parameter Names

The following rules apply when you declare object variables for IOMI methods:

- When you declare names for method parameters in the standard interface, the SAS Open Metadata Interface does not require you to use the published names for the method parameters; however, if you use a different name, the name in the object variable declaration must match the parameter name used in the method call.
- When you reference method parameters in an XML string used with the DoRequest method, the SAS Open Metadata Interface does not require you to use the published parameter names *with one*

exception: <Metadata> must be used to represent the *inMetadata* parameter in the XML string. The method parameters must also be supplied in the order given in the method documentation.

Previous Page | Next Page | Top of Page

Java IOMI Class Signature Summary Table

The Java programming environment requires the following data types for IOMI method parameters.

Java IOMI Class Signature Summary Table

Method	Parameter	Definition
AddMetadata	inMetadata	java.lang.String
	reposid	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
CheckinMetadata	inMetadata	java.lang.String
	projectReposid	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	changeName	java.lang.String
	changeDesc	java.lang.String
	changeId	java.lang.String
	ns	java.lang.String
	flags	int
	options	java.lang.String
CheckoutMetadata	inMetadata	java.lang.String
	projectReposid	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
CopyMetadata	inMetadata	java.lang.String
	targetReposid	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String

	flags	int
	options	java.lang.String
DeleteMetadata	inMetadata	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
DoRequest	inString	java.lang.String
	outString	org.omg.CORBA.StringHolder
FetchMetadata	inMetadata	java.lang.String
	projectReposid	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
GetMetadata	inMetadata	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
GetMetadataObjects	reposid	java.lang.String
	type	java.lang.String
	objects	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
GetNamespaces	ns	java.lang.String
	flags	int
	options	java.lang.String
GetRepositories	repositories	org.omg.CORBA.StringHolder

	flags	int
	options	java.lang.String
GetSubtypes	type	java.lang.String
	subtypes	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
GetTypeProperties	type	java.lang.String
	properties	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
GetTypes	type	java.lang.String
	ns	java.lang.String
	flags	int
	options	java.lang.String
IsSubtypeOf	type	java.lang.String
	supertype	java.lang.String
	result	org.omg.CORBA.BooleanHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
UndoCheckoutMetadata	inMetadata	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String
	flags	int
	options	java.lang.String
UpdateMetadata	inMetadata	java.lang.String
	outMetadata	org.omg.CORBA.StringHolder
	ns	java.lang.String

	flags	int
	options	java.lang.String

Previous Page | Next Page | Top of Page

Visual Basic IOMI Class Signature Summary Table

The Visual Basic programming environment requires the following data types for OMI method parameters.

Visual Basic OMI Class Signature Summary Table

Method	Parameter	Definition
AddMetadata	(InMetadata	As String
	Reposid	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
CheckinMetadata	(InMetadata	As String
	ProjectReposid	As String
	OutMetadata	As String
	ChangeName	As String
	ChangeDesc	As String
	ChangeId	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
CheckoutMetadata	(InMetadata	As String
	ProjectReposid	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long

CopyMetadata	(InMetadata	As String
	TargetReposid	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
DeleteMetadata	(InMetadata	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
DoRequest	(InString	As String
	OutString	As String) As Long
FetchMetadata	(InMetadata	As String
	ProjectReposid	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
GetMetadata	(InMetadata	As String,
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long

GetMetadataObjects	(Reposid	As String
	Type	As String
	Objects	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
GetNamespaces	(Ns	As String
	Flags	As Long
	Options	As String) As Long
GetRepositories	(Repositories	As String
	Flags	As Long
	Options	As String) As Long
GetSubtypes	(Type	As String
	Subtypes	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
GetTypeProperties	(Type	As String
	Properties	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
GetTypes	(Types	As String
	Ns	As String

	Flags	As Long
	Options	As String) As Long
IsSubtypeOf	(Type	As String
	Supertype	As String
	Result	As Boolean
	Ns	As String
	Flags	As Long
	Options	As String) As Long
UndoCheckoutMetadata	(InMetadata	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long
UpdateMetadata	(InMetadata	As String
	OutMetadata	As String
	Ns	As String
	Flags	As Long
	Options	As String) As Long

Previous Page | Next Page | Top of Page

Visual C++ IOMI Class Signature Summary Table

The Visual C++ programming environment requires the following data types for IOMI method parameters.

Visual C++ IOMI Class Signature Summary Table

Method	Parameter	Definition
AddMetadata	InMetadata	BSTR [in]
	Reposid	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
CheckinMetadata	InMetadata	BSTR [in]
	projectReposid	BSTR [in]
	OutMetadata	BSTR* [out]
	ChangeName	BSTR [in]
	ChangeDesc	BSTR [in]
	ChangeId	BSTR [in]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
CheckoutMetadata	InMetadata	BSTR [in]
	ProjectReposid	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
CopyMetadata	InMetadata	BSTR [in]

	TargetReposid	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
DeleteMetadata	InMetadata	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
DoRequest	InString	BSTR [in]
	OutString	BSTR* [out]
	retval	long* [out, retval]
FetchMetadata	InMetadata	BSTR [in]
	ProjectReposid	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetMetadata	InMetadata	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetMetadataObjects	Reposid	BSTR [in]
	Type	BSTR [in]

	Objects	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetNamespaces	Ns	BSTR* [out]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetRepositories	Repositories	BSTR* [out]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetSubtypes	Type	BSTR [in]
	Subtypes	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetTypeProperties	Type	BSTR [in]
	Properties	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
GetTypes	Types	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]

IsSubtypeOf	Type	BSTR [in]
	Supertype	BSTR [in]
	Result	VARIANT_BOOL* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
UndoCheckoutMetadata	InMetadata	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]
UpdateMetadata	InMetadata	BSTR [in]
	OutMetadata	BSTR* [out]
	Ns	BSTR [in]
	Flags	long [in]
	Options	BSTR [in]
	retval	long* [out, retval]

Previous Page | Next Page | Top of Page

Sample Java IOMI Client

This section describes how to create a Java IOMI client in a UNIX operating environment.

To create a connection object for a Java client, you must instantiate a CORBA Object Request Broker (ORB) and define a stub that implements the SAS Open Metadata Interface class that you want to use. You must also supply server connection properties to the ORB. To learn more about the CORBA (Common Object Request Broker Architecture), see "Using Java CORBA Stubs for IOM Objects" in the **SAS Integration Technologies Library** at `support.sas.com/rnd/itech/library/`.

Class Libraries

SAS Open Metadata Interface classes are provided in class libraries that are shipped on your installation media in the sas.oma.omi.jar and IOMJAVA.zip files. Before you can use them, you must unzip and install the IOMJAVA package on your machine, then set the CLASSPATH= environment variable to point to the location of all the JAR files.

In an IOMI client, reference the following libraries as import classes:

com.sas.meta.SASOMI.IOMI
: contains CORBA stubs for IOMI class methods.

com.sas.meta.SASOMI.IOMIHelper
: contains definitions for an OMI IOMI ORB.

org.omg.CORBA.*
: contains definitions for the IOM CORBA services.

java.util.Properties
java.util.Enumeration
java.net.URLEncoder
: contain standard Java application programming interfaces.

Sample Java IOMI Class Connection Program

The following sample program demonstrates the steps for connecting a Java client to the SAS Metadata Server using the IOMI interface. The program creates a connection object and uses the connection object to issue a GetRepositories method call. The GetRepositories method is the first method that you will want to issue because it returns the information necessary to begin interrogating a specific repository.

The sample program uses the standard call interface. Numbers have been inserted at the beginning of each step. Select a number to see a description of a step.

[1]

```
import com.sas.meta.SASOMI.IOMI;
import com.sas.meta.SASOMI.IOMIHelper;
import org.omg.CORBA.*;
import java.util.Properties;
import java.util.Enumeration;
import java.net.URLEncoder;

class runToOMI {
```

[2]

```
java.util.Properties connectionProperties = new java.util.Properties();
String connectionString = "";
IOMI connection = null;
```

[3]

```
/*************************************************************************
 Put the connection properties into a properties object.
*************************************************************************/
private void setConnectionProperties(String host,
                                     String port,
                                     String username,
                                     String password) {


   String protocol = "bridge";
   connectionProperties.put("host",host);
   connectionProperties.put("port",port);
   connectionProperties.put("protocol",protocol);
   connectionProperties.put("userName",username);
   connectionProperties.put("password",password);
}
```

[4]

```
/*************************************************************************
Take the connection properties from the properties object and return
a URL string.
*************************************************************************/
private void propertiesToUrl() {

   StringBuffer buffer = new StringBuffer("bridge://");
   buffer.append(connectionProperties.getProperty("host"));
   buffer.append(":");
   buffer.append(connectionProperties.getProperty("port"));
   buffer.append("/");
   buffer.append("2887e7d7-4780-11d4-879f-00c04f38f0db");

   StringBuffer query = null;
   Enumeration propertyNames = connectionProperties.propertyNames();

   while (propertyNames.hasMoreElements())
  {
      String propertyName = (String)propertyNames.nextElement();

      if (!propertyName.equals("host") && !propertyName.equals("port"))
      {
          if (query == null)
          {
               query = new StringBuffer();
          }
          else
          {
               query.append("&");
          }
           query.append(URLEncoder.encode(propertyName));
           query.append("=");
           query.append(URLEncoder.encode(connectionProperties.getProperty(propertyName)));

      }
  }
```

```
    if (query != null)
    {
        buffer.append("?");
        buffer.append(query.toString());
    }

    connectionString = buffer.toString();


}


   /**************************************************************************
   Instantiate a connection object and supply connection properties to the server.
   **************************************************************************/
   private void getConnected() {
```

[5]

```
      IOMI privateCMA = null;

      try {
```

[6]

```
          // Create an Object Request Broker
          ORB orb2 = new com.sas.net.brg.orb.BrgOrb();
```

[7]

```
          // Instantiate an object and set up a remote reference to it
          org.omg.CORBA.Object obj2 = orb2.string_to_object(connectionString);
```

[8]

```
          //Use helper class to set the interface you want
          privateCMA = IOMIHelper.narrow(obj2);

      }
             catch (Exception e)        {
                privateCMA = null;
                e.printStackTrace();
             }
```

[9]

```
    connection = privateCMA;
   }


   /**************************************************************************
   This releases a connection to a SAS Metadata Server.
   **************************************************************************/
   private void disConnect() {
     connection._release();
   }


   /**************************************************************************
   This sends a request to the GetRepositories method of the SAS Metadata Server.
```

```
   ***************************************************************************/
   private void getRepositories() {
        int returnCodeFromOMI = -999;
        int flags = 0;
        String options = "";
        StringHolder returnInfoFromOMI = new org.omg.CORBA.StringHolder();
     try {
```

[10]

```
        returnCodeFromOMI = connection.GetRepositories(returnInfoFromOMI,flags,options);
        System.out.println("returnCodeFromOMI = " + returnCodeFromOMI);
        System.out.println("returnInfoFromOMI = " + returnInfoFromOMI.value);
     }

     catch (com.sas.iom.SASIOMDefs.GenericError e)
     {
        System.out.println(e);
        e.printStackTrace();
     }
     catch (org.omg.CORBA.SystemException e)
     {
        System.out.println(e);
        e.printStackTrace();
     }
   }

   /**************************************************************************
   Main program
   ***************************************************************************/
   public static void main (String[] args) {

         runToOMI getToOMI = new runToOMI();
         //getToOMI.setConnectionProperties("host","port","username","password");
         getToOMI.setConnectionProperties
         ("login004.unx.sas.com","6713","myuid","mypaswd");
         getToOMI.propertiesToUrl();
         getToOMI.getConnected();
         getToOMI.getRepositories();
         getToOMI.disConnect();

         int n = 0;
         System.exit(n);

   }

}
```

[1] The **com.sas.meta.SASOMI.IOMI** and **com.sas.meta.SASOMI.IOMIHelper** packages specify that the SAS Open Metadata Interface IOMI interface is used. The **org.omg.CORBA.*** package specifies the IOM CORBA interface. (The asterisk at the end of the name indicates that all classes in the CORBA package should be available to the compiler for compilation.)

[2] These statements declare object variables for a connectionProperties object, a connectionString object, and an IOMI connection object. Values are assigned later in the program.

[3]

	These statements create the connectionProperties object. Note the use of name=value pairs to supply all server connection values except the factory number. The factory number must be provided when the server properties are converted to a URL string.
[4]	These statements convert the server connection properties into a URL string. Note the factory number "2887e7d7−4780−11d4−879f−00c04f38f0db". If you omit this value, or if the string has an error in it, the IOM server returns an error.
[5]	This statement declares an object variable called "PrivateCMA".
[6]	This statement creates an object request broker (ORB) for the SAS Open Metadata Interface.
[7]	This statement instantiates an object (ob2) for a generic stub (org.omg.CORBA.object) and sets up a remote reference to it.
[8]	This statement assigns the privateCMA object variable to a helper class that sets the obj2 object as an IOMI interface.
[9]	This statement sets the object variable "connection" to the values in privateCMA.
[10]	This statement uses the connection object in a GetRepositories method call.

Previous Page | Next Page | Top of Page

Sample Visual Basic OMI Client

This section describes how to create a Visual Basic SAS Open Metadata Interface client. The client uses the IOM Bridge for COM to connect to a server in a Windows or other operating environment. Note that in the Visual Basic environment, the *IOMI* class is referred to as the *OMI* class.

Type Libraries

All Visual Basic clients (OMI and IServer) must reference the following type library:

SASOMI: (SAS 9.1) Type Library
contains the IServer and OMI method classes.

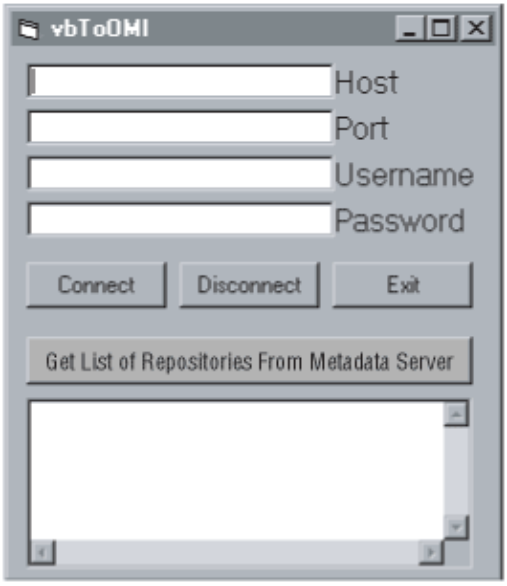
Clients that use the IOM Bridge for COM must additionally select the following type library:

Combridge 1.0 Type Library
contains the definitions for IOM Bridge for COM software.

Visual Basic provides a semi−interactive client development environment. Use the software's References window to find and select the type libraries from a list.

Sample Visual Basic OMI Class Connection Program

The following window was created by the Visual Basic program. The program collects the information necessary to connect to the SAS Metadata Server and uses it to issue a GetRepositories method call. The GetRepositories method is the first method that you will want to issue because it returns the information necessary to begin interrogating a specific repository.



Note: In order to focus on the SAS Open Metadata Interface, the code that created the window has been omitted from the sample. Numbers have been inserted at the beginning of each step. Select a number to see a description of a step.

The program issues the metadata request via the DoRequest method.

[1]

```
Option Explicit
    Dim obCB As New SASCombridge.Combridge
    Dim obOMI As SASOMI.OMI
    Dim returnFromOMI As Long
    Dim wrap          As String
    Dim inputXML      As String
    Dim outputXML     As String
```

[2]

```
 Private Sub connect_Click()
       On Error GoTo connectError
       Set obOMI = Nothing
       Set obOMI = obCB.CreateObject(host.Text, username.Text, password.Text,
         CInt(port.Text), "", EncryptNothing, "", "SASOMI.OMI")
       msgOutputToUser.Text = "Connected to server." + wrap + msgOutputToUser.Text
    Exit Sub

    connectError:
       msgOutputToUser.Text = Err.Description + wrap + msgOutputToUser.Text
    End Sub
```

[3]

```
 Private Sub disconnect_Click()
       On Error GoTo disconnectError
       Set obOMI = Nothing
       msgOutputToUser.Text = "Disconnected from server." + wrap + msgOutputToUser.Text
    Exit Sub

    disconnectError:
       msgOutputToUser.Text = Err.Description + wrap + msgOutputToUser.Text
    End Sub

    Private Sub exit_Click()
       End
    End Sub

 Private Sub Form_Load()
       wrap = Chr(13) + Chr(10)
    End Sub
```

[4]

```
 Private Sub runmethod_Click()
       On Error GoTo runError

       inputXML = "<getrepositories>" + _
                  "<repositories/>" + _
                  "<flags>0</flags>" + _
                  "<options/>" + _
                  "</getrepositories>"
```

[5]

```
 returnFromOMI = obOMI.DoRequest(inputXML, outputXML)
```

```
    msgOutputToUser.Text = "Output from SAS Metadata Server: " _
      + outputXML + Str(returnFromOMI) + wrap + msgOutputToUser.Text
    msgOutputToUser.Text = "Using  DoRequest  method, " _
      + "returnCode = " + Str(returnFromOMI) + wrap + msgOutputToUser.Text
Exit Sub

runError:
   msgOutputToUser.Text = Err.Description + wrap + msgOutputToUser.Text

End Sub
```

[1]

The DIM statements declare object variables for a SASCombridge object, a SASOMI OMI class object, the method return code, an inputXML string, and an output XML string. The latter two object variables are parameters for the DoRequest method.

[2]

These statements create and assign values to a SAS Metadata Server connection object *obOMI* that is invoked each time the **Connect** button is selected. The server connection properties are passed as arguments to the Visual Basic CreateObject function. The host, username, password, and port number are passed as name=value pairs and the *encryptionPolicy* and *serverIdentifier* properties are passed as strings. Null values are provided for the *servicename* and *encryptionAlgorithm* parameters.

[3]

These statements define the actions for the **Disconnect** button. *obOMI* is reset to a null value.

[4]

These statements define the actions for the **Get List of Repositories from OMI Server** button. An XML string containing the GetRepositories method and its parameters is assigned to the *inputXML* object variable.

[5]

This statement issues a DoRequest method call to pass the input XML string to the SAS Metadata Server. Note that the DoRequest call references the connection object *obOMI* that was created earlier in the program.

Previous Page | Next Page | Top of Page

Sample Visual C++ IOMI Client

This section describes how to create a Visual C++ IOMI client. The sample program connects to the SAS Metadata Server using the IOM Bridge for COM.

Visual C++ uses object pointers instead of object variables to represent method parameters. Therefore, you need to convert SAS Open Metadata Interface string parameters to BSTRs before you can issue a method call, and you need to convert output from a BSTR to another format if you intend to use it elsewhere.

Type Libraries

All Visual C++ clients (IOMI and IServer) must reference the following type library:

SASOMI: (SAS 9.1) Type Library
contains the IServer and IOMI method classes.

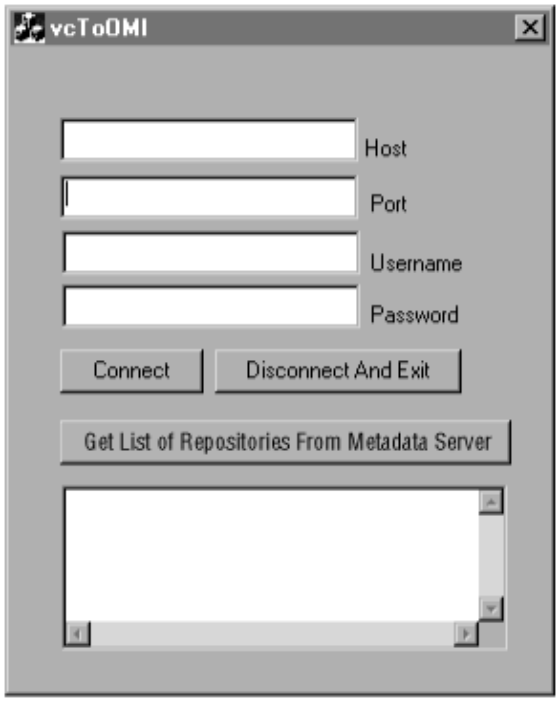
Clients that use the IOM Bridge for COM must additionally select the following:

Combridge 1.0 Type Library
contains the definitions for IOM Bridge for COM software.

Use the software's OLE/COM Object Viewer to reference the type libraries.

Sample Visual C++ IOMI Class Connection Program

The following window was created using the Visual C++ MFC AppWizard. The window collects the information necessary to connect to the SAS Metadata Server and uses it to issue a GetRepositories method call. The GetRepositories method is the first method that you will want to issue because it returns the information necessary to begin interrogating a specific repository. To add SAS Open Metadata Interface functionality to the window, edit the vcToOMIDlg.h and vcToOMIDlg.cpp files as described in the sections that follow.



vcToOMIDlg.h

vcToOMIDlg.h is a header file that defines the functions that are available to the MFCAppWizard. In the vcToOMIDlg.h file, add IMPORT statements for the appropriate IOM and SAS Open Metadata Interface class libraries and insert statements referencing these libraries as follows. To conserve space, only the affected portion of the header file is shown.

```
// vcToOMIDlg.h : header file
//

#if !defined(AFX_VCTOOMIDLG_H__A31C4C7A_E267_11D4_87A6_00C04F2C3599__INCLUDED_)
#define AFX_VCTOOMIDLG_H__A31C4C7A_E267_11D4_87A6_00C04F2C3599__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#import "c:\program files\sas\shared files\integration technologies\omi.tlb"
using namespace SASOMI;

#import "c:\program files\sas\shared files\integration technologies\sascomb.dll"
using namespace SASCombridge;

/////////////////////////////////////////////////////////////////////////////
// CVcToOMIDlg dialog

class CVcToOMIDlg : public CDialog
{
...
```

The IMPORT statements are added after the IF statements. The first IMPORT statement specifies the pathname of the SASOMI type library and assigns it the namespace "SASOMI". The second IMPORT statement specifies the pathname of the Combridge DLL and assigns it the namespace "SASCombridge".

vcToOMIDlg.cpp

The vcToOMIDlg.cpp file is an implementation file. In this file, add the following statements:

- pointers that reference the namespaces defined in the vcToOMIDlg.h file
- a coinitialize statement
- a connection pointer to the SAS Metadata Server
- the necessary BSTRs to issue the GetRepositories method call
- statements releasing the SAS Metadata Server.

Add the first set of pointers beneath the IF statements, in the form *Namespace::DesiredInterface::DiscretionaryName*, as follows. In the code fragment, note that the pointer for the SASOMI namespace is set to the IOMI interface and is assigned the name pIOMI. A second pointer for the SASCombridge DLL is set as ICombridgePtr and assigned the name pICombridge.

```
// vcToOMIDlg.cpp : implementation file
//

#include "stdafx.h"
#include "vcToOMI.h"
#include "vcToOMIDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


CString wrap("\r\n");
SASOMI::IOMIPtr pIOMI;
SASCombridge::ICombridgePtr pICombridge;

...
```

Add the coinitialize statement to the OnInitDialog subroutine, as follows.

```
/////////////////////////////////////////////////////////////////////////////
// CVcToOMIDlg message handlers

BOOL CVcToOMIDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);                // Set small icon
CoInitialize(NULL);


        return TRUE;
}
```

The following code fragment creates the connection pointer, pIOMI. pIOMI is defined as a local pointer to an in−process server that is invoked by the Connect button. The pointer is assigned the values of the IOMI namespace, the SASCombridge DLL, and a local object that supplies server connection properties. Insert this somewhere after the CoInitialize statement.

```
void CVcToOMIDlg::OnConnect()
{
```

```
   try {
          UpdateData(TRUE);
      ICombridgePtr pICombridgea("SAS.Combridge", NULL, CLSCTX_INPROC_SERVER);


      pICombridge = pICombridgea;


      pIOMI = (IOMIPtr)(pICombridge->CreateObject((_bstr_t)m_Host,
                                                  (_bstr_t)m_Username,
                                                  (_bstr_t)m_Password,
                                                  m_Port,
                                                                                  "
                                                                                  E
                                                                                  "
                                                                                  "



      m_MsgOutputToUser = "Connected to server" + wrap + m_MsgOutputToUser;
      UpdateData(FALSE);
        }

        catch (_com_error e) {
      m_MsgOutputToUser = e.ErrorMessage() + wrap + m_MsgOutputToUser;
      UpdateData(FALSE);
        }

}
```

Use the following sample code fragment to issue a GetRepositories call using the DoRequest method. In the code fragment, the CString *inputXML* contains the GetRepositories method and its parameters. It and CString *outputXML* are converted to BSTRs and passed to the SAS Metadata Server via the generic DoRequest method. The DoRequest method call references the pIOMI connection pointer to establish communication with the SAS Metadata Server.

```
void CVcToOMIDlg::OnRunmethod()
{
    HRESULT hr;

        CString inputXML("<getrepositories><repositories/><flags>0</flags><options/>
    </getrepositories>");
    BSTR inXML = inputXML.AllocSysString();

        CString outputXML;
        BSTR outXMLa = outputXML.AllocSysString();

   UpdateData(TRUE);

   try {
          hr = pIOMI->DoRequest(inXML, &outXMLa);
          CString outXML3(outXMLa);
      m_MsgOutputToUser = outXML3 + wrap + m_MsgOutputToUser;
      UpdateData(FALSE);
        }

        catch (_com_error e) {
      m_MsgOutputToUser = e.ErrorMessage() + wrap + m_MsgOutputToUser;
      UpdateData(FALSE);
        }

}
```

Finally, insert the following code to disconnect from the SAS Metadata Server.

```
void CVcToOMIDlg::OnDisconnect()
{

   UpdateData(TRUE);

   try {

      pIOMI->Release();
      pICombridge->Release();
      m_MsgOutputToUser = "Disconnected from server." + wrap + m_MsgOutputToUser;
      UpdateData(FALSE);
      OnOK();
        }

        catch (_com_error e) {
      m_MsgOutputToUser = e.ErrorMessage() + wrap + m_MsgOutputToUser;
      UpdateData(FALSE);
      OnOK();
        }

}
```

Previous Page | Next Page | Top of Page

METADATA Procedure

The METADATA procedure sends an XML request containing a SAS Open Metadata Interface method call to a SAS Metadata Server. Procedure statements specify the information necessary to connect to the server, to pass the input XML string, and to optionally store the output XML string to a file.

Procedure Syntax

```
PROC METADATA

Server Connection Statements
   <SERVER="host_name">
   <PORT=port_number>
   <USERID="userid">
   <PASSWORD="password">
   <PROTOCOL="value">

Input Statement
   IN=fileref | "embedded XML stream";

Output Statements
   HEADER=NONE|SIMPLE|FULL
   OUT=fileref;

Informational Statements
   REPOSITORY="repository_identifier";
   VERBOSE;
```

Server Connection Statements

The server connection statements are optional in the METADATA procedure but are required to connect to the SAS Metadata Server. If you omit these statements, then the values of the system options METASERVER, METAPORT, METAUSER, METAPASS and METAPROTOCOL are used. If these system values are empty or incomplete, a dialog box will be posted to acquire the option values. Once entered, the values are stored as system options for the remainder of the current session. For more information about these system options, see SAS Metadata System Options.

SERVER=
: is the host name or network IP (Internet Protocol) address of the computer hosting the SAS Metadata Server that you want to access, for example, **SERVER="d6292.us.company.com"**. The value **localhost** can be used if the SAS session is connecting to a server on the same computer.

PORT=
: is the TCP port to which the SAS Metadata Server listens for connections. For example, **port=5282**. It should be the same port number used to start the SAS Metadata Server.

USERID=
: is a valid user ID on the computer hosting the server.

PASSWORD=
: is the password corresponding to the user ID.

PROTOCOL=
: is the network protocol for communicating with the SAS Metadata Server. The valid value is:

  BRIDGE
  : specifies that the connection will use the SAS Bridge protocol.

Input Statement

IN=

specifies an XML−formatted SAS Open Metadata Interface method call or a fileref that points to an XML file containing a SAS Open Metadata Interface method call.

PROC METADATA submits method calls to the server via the DoRequest method. The DoRequest method provides a generic way of submitting metadata−related method calls to the metadata server. To learn how to format a metadata−related method call in XML, see DoRequest. To learn more about metadata−related SAS Open Metadata Interface methods, see IOMI Class. To learn about the metadata types used by SAS Open Metadata Interface methods to store metadata, see SAS Namespace Types. To learn how to write a metadata property string, see Constructing a Metadata Property String.

Output Statements

HEADER=

optionally inserts an XML encoding declaration in the output XML file created by the OUT= statement. The declaration specifies the character−set encoding for browsers and parsers to use when processing national language characters in the output XML file. Valid values are:

NONE

omits an encoding declaration. Browsers and parsers might not handle national language characters appropriately.

SIMPLE

inserts the static header "<? xml version=1.0?>", which instructs the browser or parser to detect the encoding.

FULL

creates an XML declaration that specifies the SAS session encoding. When HEADER=FULL, the encoding value is taken from the ENCODING= option specified in the FILENAME statement used to identify the output XML file to the SAS system or from the ENCODING= system option. See Examples for usage information.

OUT=

optionally specifies a fileref in which to store the output XML string returned by the SAS Metadata Server. In most cases, the output will mirror the input, except that the requested values will be filled in. Use the HEADER statement to specify a character−set encoding for the output. If the OUT= statement is omitted, procedure output is written to the SAS LOG.

Informational Statements

The following statements are used to supply or obtain information from the metadata server:

REPOSITORY=

specifies the name of the repository to use when resolving $METAREPOSITORY substitution. PROC METADATA allows you to specify the substitution variable $METAREPOSITORY within your input XML. The substitution variable is resolved to the repository identifier of the named repository specified by REPOSITORY=. If you do not specify REPOSITORY= on the procedure, the value of the METAREPOSITORY system option is used. REPOS is an alias for REPOSITORY.

VERBOSE=

specifies to print the input XML string after it has been through preprocessing.

Examples

The following PROC METADATA example issues a GetTypes method call.

```
PROC METADATA

  IN="<GetTypes>
      <Types/>
      <Ns>SAS</Ns>
      <Flags/>
      <Options/>
      ";
  RUN;
```

This example omits server connection statements, therefore, the software will use the system option values of METASERVER, METAPORT, METAUSER, METAPASS, and METAPROTOCOL to connect to the metadata server. If these system option values are empty or blank, then a dialog box is posted to acquire the connection information.

The following example illustrates the behavior of the VERBOSE statement. PROC METADATA issues a GetMetadataObjects request to list all of the objects of type PhysicalTable that are defined in the active repository. The active repository identifier is substituted where $METAREPOSITORY appears in the XML. The name of the active repository we are using is "My Repository".

```
PROC METADATA

    REPOS="My Repository"

    IN="<GetMetadataObjects>
        <Reposid>$METAREPOSITORY</Reposid>
        <Type>PhysicalTable</Type>
        <Objects/>
        <Ns>SAS</Ns>
        <Flags/>
        <Options/>
        </GetMetadataObjects>"
    VERBOSE;
    RUN;
```

The VERBOSE statement returns the following preprocessed XML, which includes the repository identifier referenced by $METAREPOSITORY.

```
        NOTE: Input XML:

        <GetMetadataObjects>
        <Reposid>A0000001.A5K2EL3N</Reposid>
        <Type>PhysicalTable</Type>
        <Objects/>
        <Ns>SAS</Ns>
        <Flags/>
        <Options/>
        </GetMetadataObjects>
```

The following is an example of a PROC METADATA call that specifies server connection statements with the IN= statement. If the request is issued on the same host computer where the metadata server is running, the value "localhost" can be specified in the SERVER= statement. The procedure issues a GetRepositories method to list all repositories on the metadata server:

```
PROC METADATA

    SERVER="host_name"
    PORT=port_number
    USERID="cubtest"
    PASSWORD="cubtest1"

    IN="<GetRepositories>
        <Repositories/>
        <Flags/>
        <Options/>
        es>";
   RUN;
```

The following example shows how filerefs are used in the IN= and OUT= statements:

```
filename output "output.xml";
filename input  "input.xml";

PROC METADATA

   SERVER="host_name"
   PORT=port_number
   USERID="cubtest"
   PASSWORD="cubtest1"
   IN=input
   OUT=output;

   RUN;
```

The example submits the contents of input.xml to the metadata server and stores the server's response in output.xml.

The following examples show how the HEADER=SIMPLE and HEADER=FULL options are used.

```
filename out "u:\out.xml" encoding=ebcdic;

  PROC METADATA

    header=simple
    out=out
    IN="<GetTypes>
        <Types/>
        <Ns>SAS</Ns>
        <Flags/>
        <Options/>
        </GetTypes>";
    RUN;
```

Inserts the static header "<?xml version="1.0" ?>" in the output XML file identified by the fileref "OUT".

```
  filename out "u:\out.xml" encoding=ebcdic;

  PROC METADATA

    header=full
    out=out
    IN="<GetTypes>
        <Types/>
        <Ns>SAS</Ns>
        <Flags/>
```

```
    <Options/>
    </GetTypes>";
RUN;
```

Inserts the header "<?xml version="1.0" encoding="ebcdic1047"?>" in the output XML file identified by the fileref "OUT". "ebcdic1047" is the encoding for western EBCDIC. For a list of encoding values, see "ENCODING= Values in SAS Language Elements" in the **SAS 9.1 National Language Support (NLS): User's Guide**.

Previous Page | Next Page | Top of Page

METAOPERATE Procedure

The METAOPERATE procedure enables you to perform administrative tasks associated with the SAS Metadata Server and metadata repositories in batch mode. Using PROC METAOPERATE you can

- unregister, delete, purge, or empty a metadata repository
- pause one or more repositories or the repository manager to change their state, and then resume them to their original state, for example, to temporarily set a repository to READONLY
- refresh the server to recover memory or to change certain configuration options, like enabling or disabling Applications Response Measurement (ARM) logging
- stop or get the status of the SAS Metadata Server.

Procedure statements connect to the metadata server and supply the necessary parameters to perform the desired action.

Procedure Syntax

```
PROC METAOPERATE

Server Connection Statements
   <SERVER="host_name">
   <PORT=port_number>
   <USERID="userid">
   <PASSWORD="password">
   <PROTOCOL=value>

Action Statements
   ACTION=PAUSE|RESUME|REFRESH|UNREGISTER|DELETE|PURGE|EMPTY|STATUS|STOP|NOAUTOPAUSE
   OPTIONS="pause_options|resume_options|refresh_options";

Repository Identification Statement
   REPOSITORY="repository_name"
```

Server Connection Statements

The server connection statements establish communication with the SAS Metadata Server. The statements are optional in the METAOPERATE procedure but are required to connect to the SAS Metadata Server. If you omit these statements then the values of the system options METASERVER, METAPORT, METAUSER, METAPASS, and METAPROTOCOL are used. For more information about these system options, see SAS Metadata System Options. If the connection parameters are empty or incomplete, a dialog box will be posted to acquire the parameter values. Once entered, the values are stored as system options for the remainder of the current session. When specifying connection parameters, be sure to enter or omit quotation marks as documented. If you enter quotation marks where they should be omitted, or vice versa, the connection will fail.

SERVER=
: is the host name or network IP (Internet Protocol) address of the computer hosting the SAS Metadata Server that you want to access, for example, **SERVER="d6292.us.company.com"**. The value **localhost** can be used if the SAS session is connecting to a server on the same computer.

PORT=
: is the TCP port to which the SAS Metadata Server listens for connections. For example, **port=5282**. Specify the port number used to start the SAS Metadata Server.

USERID=

is an authenticated user ID on the computer hosting the server. The metadata server supports several authentication providers. For more information, see "Controlling User Access to the Server" in the **SAS 9.1 Metadata Server: Setup Guide**.

PASSWORD=

is the password corresponding to the user ID.

PROTOCOL=

is the network protocol for communicating with the SAS Metadata Server. The valid value is BRIDGE.

BRIDGE

specifies that the connection will use the SAS Bridge protocol.

Action Statements

ACTION=

The ACTION statement specifies the action that you wish to perform. ACTION is a required statement.

Note: A user must have Administrative User status to execute all actions except STATUS. For more information, see "Server Administrative Privileges" in the **SAS 9.1 Metadata Server: Setup Guide**.

PAUSE

suspends client activity in one or all repositories or the repository manager and enables you to temporarily change their state. The repositories to pause and their modified state are identified in <Repository> XML elements that are passed to the metadata server in the OPTIONS statement. When a repository is paused using the <Repository> element, the default Pause state is READONLY. A state value of READONLY allows clients to read metadata in the specified repository or repository manager but prevents them from writing to it. Issuing a PAUSE action without an OPTIONS statement pauses all repositories on the metadata server, except the repository manager, to an OFFLINE state. A state value of OFFLINE disables read and write access to the specified repositories. You can determine a repository's current state by issuing a GetRepositories method call via PROC METADATA and checking its PauseState attribute. For more information, see the documentation for the METADATA Procedure and the GetRepositories method.

Note: A repository that is paused must be resumed to restore client activity to its normal state.

RESUME

restores client activity in a paused repository to its normal state. You can determine a repository's normal state by issuing a GetRepositories method call via PROC METADATA and checking its Access attribute. For more information, see the documentation for the METADATA Procedure and the GetRepositories method. The repository to resume is identified in a <Repository> XML element that is passed to the metadata server in the OPTIONS statement. Passing the RESUME action without the OPTIONS statement restores client activity in all repositories on the metadata server and the repository manager. For more information about the <Repository> element, see the description of the OPTIONS statement.

REFRESH

The REFRESH action has two uses:

- It can quiesce client activity on the SAS Metadata Server long enough to invoke the specified server configuration option(s) and then automatically resume it. Currently, one server configuration option can be changed using the REFRESH action. ARM logging can be enabled or disabled by passing an <ARM> XML element in the

OPTIONS statement.

- It pauses and resumes client activity on one or more repositories in a single step. This is useful for recovering memory on the server and for reloading authorization inheritance rules. The repositories to refresh are identified in <Repository> XML elements that are passed to the metadata server in the OPTIONS statement.

Invoking the REFRESH action without an OPTIONS statement has no effect. For more information about the <ARM> and <Repository> elements, see the description of the OPTIONS statement.

UNREGISTER
removes metadata about how to access the specified repository from the repository manager without disturbing the metadata records in the repository.

DELETE
deletes the specified repository and the metadata necessary to access it from the repository manager.

PURGE
removes logically deleted metadata records from the specified repository without disturbing the current metadata records.

EMPTY
deletes the metadata records in the specified repository without disturbing the repository's registration in the repository manager.

STATUS
returns the metadata server's SAS version number, host operating environment, the user ID that started the server, the SAS Metadata Model version number, and information about the server's current state.

STOP
stops all client activity immediately and terminates the metadata server. Metadata in repositories is unavailable until the metadata server is restarted. In SAS 9.1, a metadata server is started independently of SAS.

NOAUTOPAUSE
Each of the UNREGISTER, DELETE, PURGE, and EMPTY actions issue an implicit PAUSE without options before performing the action, and an implicit RESUME after the action, in order to quiesce client activity on the server before changing repository attributes or dependencies. This was required in 9.0 servers and is optional in 9.1 servers. To turn off this behavior, specify the NOAUTOPAUSE option on the procedure. NOAUTOPAUSE should also be specified for the PAUSE action when a ltRepository> XML element is passed in the OPTIONS statement.

OPTIONS=
Passes an optional quoted string containing one or more XML elements that specify options for the PAUSE, RESUME, or REFRESH actions. The supported XML elements are:

`<REPOSITORY ID="Reposid|REPOSMGR|ALL" STATE="READONLY|OFFLINE"/>`
identifies the repositories on which to invoke the PAUSE, RESUME, or REFRESH actions and for PAUSE, optionally specifies a pause state.

ID=
is required and specifies the unique 8−character or 17−character identifier of a repository, REPOSMGR to indicate the repository manager, or ALL. ALL indicates that all repositories on the server *except* the repository manager should be paused, resumed, or refreshed.

STATE=
optionally specifies a state for the PAUSE action. If the STATE parameter is omitted, the default pause state is READONLY. READONLY allows clients to read metadata in the specified repository or repository manager but prevents them from writing to it. OFFLINE disables read and write access to the specified repositories or the

repository manager. The STATE parameter is ignored by the RESUME and REFRESH actions.

`<ARM ARMSUBSYS="(ARM_NONE|ARM_OMA|ARM_DSIO)" ARMLOC="fileref|filename"/>`
specifies one or more ARM system options to enable or disable ARM_OMA logging. If ARM logging is already enabled, specifying ARMLOC= writes the ARM log to a new location. Note that relative and absolute pathnames are read as different locations. See "Starting the ARM_OMA Subsystem" in the **SAS 9.1 Metadata Server: Setup Guide** for syntax details.

Note: To ensure that the options string is parsed correctly by the metadata server, any double−quotation marks in the XML elements need to be marked in some way to indicate that they should be treated as characters. This can be done by alternating single and double quotation marks or double and double−double quotation marks to distinguish the string literal from a quoted value as follows:

```
'<ARM ARMSUBSYS="(ARM_OMA)" ARMLOC="fileref"/>'
"<ARM ARMSUBSYS=""(ARM_OMA)"" ARMLOC=""fileref""/>"
```

Repository Identification Statement

REPOSITORY=
specifies the repository identifier of an existing repository. The REPOSITORY statement is required when the ACTION is UNREGISTER, DELETE, or EMPTY. REPOS= is an alias for REPOSITORY=.

Examples

The following PROC METAOPERATE example connects to a metadata server that is running on the same host as the SAS session and purges logically deleted metadata records from the specified repository.

```
PROC METAOPERATE
   SERVER="localhost"
   PORT=1111
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=PURGE
   REPOS="MyRepos";

RUN;
```

The following example deletes the metadata records in the specified repository without disturbing the repository's registration in the repository manager. Since the connection statements are omitted, the procedure will use the values of the METASERVER, METAPORT, METAUSER, METAPASS, and METAPROTOCOL system options to establish the connection or present a connection dialog box if none are found. The EMPTY action is useful for clearing a repository that will be repopulated.

```
PROC METAOPERATE

  ACTION=EMPTY
  REPOS="MyRepos";

RUN;
```

The following example connects to a metadata server that is running on another host and issues a PAUSE action to temporarily change client activity on repository A5234567 to an OFFLINE state.

```
PROC METAOPERATE
   SERVER="d6292.us.company.com"
   PORT=2222
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=PAUSE
   OPTIONS="<Repository Id=""A5234567"" State=""OFFLINE""/>"
   NOAUTOPAUSE;

RUN;
```

The following example connects to the server and issues a RESUME action to restore client activity on repository A5234567 to its original access mode:

```
PROC METAOPERATE
   SERVER="d6292.us.company.com"
   PORT=2222
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=RESUME
   OPTIONS="<Repository Id=""A5234567""/>"
   NOAUTOPAUSE;

RUN;
```

The following example connects to the server and issues a PAUSE action to change client activity on all repositories and the repository manager to a READONLY state. The State parameter is omitted from the options string because the default Pause state is READONLY.

```
PROC METAOPERATE
   SERVER="d6292.us.company.com"
   PORT=2222
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=PAUSE
   OPTIONS="<Repository Id=""ALL""/>
            <Repository Id=""REPOSMGR""/>"
   NOAUTOPAUSE;

RUN;
```

The following example connects to a metadata server that is running on the same host as the SAS session and issues a STATUS action:

```
PROC METAOPERATE
   SERVER="localhost"
   PORT=1111
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE
```

```
   ACTION=STATUS;

RUN;
```

The following is an example of the output returned by a STATUS request:

```
  NOTE: Server ctomr1.na.company.com SAS Version is 9.01.01B0P02122003.
  NOTE: Server ctomr1.na.company.com Operating System is WIN_SRV.
  NOTE: Server ctomr1.na.company.com Operating System Family is WIN.
  NOTE: Server ctomr1.na.company.com Operating System Version is .
  NOTE: Server ctomr1.na.company.com started by sasjeb.
  NOTE: Server ctomr1.na.company.com Metadata Model is Version 3.01.
  NOTE: Server ctomr1.na.company.com is RUNNING on February 19, 2003 01:55:30.
```

The following example connects to a metadata server that is running on the same host as the SAS session and issues a STOP action. The STOP action quiesces all client activity and terminates the metadata server.

```
PROC METAOPERATE
   SERVER="localhost"
   PORT=1111
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=STOP;

RUN;
```

Previous Page | Next Page | Top of Page

SAS Metadata DATA Step Functions

Overview

SAS provides a family of metadata DATA step functions to get attributes, associations, and properties from metadata objects. These functions also enable you to set and update attributes, associations, and properties for metadata objects.

For more information about the SAS Metadata Model, see SAS Namespace Types.

Connection Information

Before you can use the metadata DATA step functions, you must establish a connection with the SAS Metadata Server by using the SAS metadata system options. The METASERVER, METAPORT, METAUSER, and METAPASS options individually specify the metadata serve r connection properties for a given user. The METAPROTOCOL option sets the network protocol for communicating with the SAS Metadata Server and the METAREPOSITORY option specifies the name of the default repository to use on the SAS Metadata Server.

The following option statements set the metadata system options to connect to the SAS Metadata Server:

```
options metaserver="a123.us.company.com"
        metaport=9999
        metauser="metaid"
        metapass="metapwd"
        metaprotocol=bridge
        metarepository="myrepos";
```

For more information about these system options, see SAS Metadata System Options.

Referencing Metadata Objects in the DATA Step

The SAS Open Metadata Architecture uses Uniform Resource Identifier (URI) formats to identify metadata objects. A URI is a generic set of all names and addresses, which are short strings that refer to objects.

When you reference a metadata object on the server, typically you use the generated ID for the metadata object. The object ID has the general form of "A3YBDKS4.AH000001".

When you use the SAS metadata DATA step functions, you must reference objects using one of the following URI formats:

- Reference by ID only

  omsobj:`A3YBDKS4.AH000001`

  This format includes the unique instance identifier that is assigned to the metadata object. This format is the least efficient metadata resource reference.

- Reference by type and id

  omsobj:`LogicalServer/A3YBDKS4.AH000001`

This format includes the metadata object type and the unique instance identifier that is assigned to the metadata object. This format is the most efficient metadata resource reference.

- Reference by type and search parameters

```
omsobj:LogicalServer?@Name='My Logical Server'
```

This format includes the metadata object type and an attribute value, such as the name. This format is generally more efficient than just a reference by ID, provided that your type parameter is not Root.

The parameters after the "?" correspond to a valid query with an <XMLSelect search="criteria"> specification. The syntax of the criteria is

```
object[attribute_criteria][association_path]
```

For more information, see <XMLSelect> Element Form and Search Criteria Syntax in the ***SAS Open Metadata Interface: User's Guide***.

Notes:

- The ***omsobj:*** prefix is case insensitive.
- Slashes in the URI can be either a forward slash (/) or a backward slash (\).
- Escape characters are supported with the %nn URL escape syntax. For more information, see the URLENCODE function in the ***SAS Language Reference Dictionary***.

Performance Issues

For performance reasons, metadata objects are cached by URI within a DATA step or SCL program. To refresh the metadata object with the most recent data from the server, purge the URI with the METADATA_PURGE fu nction.

For best performance, always resolve your URI into an ID instance. This will fully exploit the object caching and reduce the number of reads from the server. For example, if you make several function calls on the object "OMSOBJ:LogicalServer?@Name='foo '" , first use the METADATA_RESOLVE or METADATA_GETNOBJ function to convert the object to "OMSOBJ:LogicalServer\A1234567.A1234567". URIs in the ID instance form usually require only one read from the server per DATA step or SCL program.

Summary Table of Metadata DATA Step Functions

The following table lists the SAS metadata DATA step functions in alphabetical order.

Metadata DATA Step Functions

Name	Description
METADATA_DELASSN	Deletes all objects that make up the specified association
METADATA_DELOBJ	Deletes the first object specified by the input URI
METADATA_GETATTR	Returns the named attribute for the object specified by the URI

METADATA_GETNASL	Returns the n^{th} named association for the object URI
METADATA_GETNASN	Returns the n^{th} associated object of the association specified
METADATA_GETNATR	Returns the n^{th} attribute on the object specified by the URI
METADATA_GETNOBJ	Returns the n^{th} object matching the specified URI
METADATA_GETNPRP	Returns the n^{th} property on the object specified by the input URI
METADATA_GETNTYP	Returns the n^{th} object type on the server
METADATA_GETPROP	Returns the named property for the object specified by the input URI
METADATA_NEWOBJ	Creates a new metadata object
METADATA_PAUSED	Determines whether the server is paused
METADATA_PURGE	Purges the specified URI
METADATA_RESOLVE	Resolves a metadata URI into a specific object on the current metadata server
METADATA_SETASSN	Modifies an association list for an object
METADATA_SETATTR	Sets the named attribute for the object specified by the input URI
METADATA_SETPROP	Sets the named property for the object specified by the input URI
METADATA_VERSION	Returns the server model version number

Previous Page | Next Page | Top of Page

SAS Metadata System Options

SAS provides a family of metadata system options to define the default SAS Metadata Server connection, encryption, and resource options that SAS clients will use. Usually these options are set at installation in the CONFIG.SAS file. However, you can use any SAS language interface to change the values at any time.

Connection Options

The connection options are required to establish a connection with the SAS Metadata Server. These options include METASERVER, METAPORT, METAPROTOCOL, METAUSER, METAPASS, METAPROFILE, and METACONNECT.

The METASERVER, METAPORT, METAUSER, and METAPASS options individually specify the metadata server connection properties for a given user. The METAPROFILE and METACONNECT options specify a stored metadata server connection profile.

Using the Individual Options

The value of METAPROTOCOL determines which other options must be specified in order to establish the server connection. METAPROTOCOL supports two values: COM and BRIDGE. COM specifies the native COM protocol and is experimental in SAS 9.1.

The following table summarizes the options required by each protocol.

Options Required by Protocol

System option	COM	BRIDGE
METASERVER	Optional; if an interface is not specified, then the local COM is used.	Required
METAPORT	Optional	Required
METAUSER	Optional	Required*
METAPASS	Optional	Required*

* If METAUSER and METAPASS are not set and you are running interactively, SAS will present a logon dialog box to acquire these option values for the session. If you are not running interactively, you will have to either specify them using the OPTIONS statement or in the SAS client.

CONFIG.SYS Examples

To set the default metadata server to use the COM protocol with an IP address of "10.20.11.112" and port of 9999, you would add the following lines to the config file:

```
-METAPROTOCOL COM
-METASERVER   "10.20.11.112"
-METAPORT     9999
```

To set the default metadata server to use the BRIDGE protocol, an IP address of "10.20.11.112", a port of 9999, the user ID "sasuser" and the password "sasuser1", you would add the following lines to the config file.

```
-METAPROTOCOL BRIDGE
-METASERVER   "10.20.11.112"
-METAPORT     9999
-METAUSER     "sasuser"
-METAPASS     "sasuser1"
```

Notes:

1. In a network environment, METAUSER should specify a fully qualified user ID, for example, SERVERNAME\USERID.
2. Use PROC PWENCODE to encode the password value to be stored in METAPASS.

Using a Stored User Connection Profile

The METAPROFILE and METACONNECT options reference a stored metadata server connection profile that you create using the METACON command. METAPROFILE specifies the physical path of an XML document that contains metadata user profiles. METACONNECT identifies which named connection in the user profiles to use.

The following is a CONFIG.SYS example that invokes a user connection profile named "Mike's profile".

```
-METAPROFILE  "!SASROOT\metauser.xml"
-METACONNECT   "Mike's profile"
```

Encryption Options

The METAENCRYPTLEVEL and METAENCRYPTALG options are used to encrypt the metadata server connection if SAS/SECURE is licensed. See their descriptions in the **SAS 9.1 Language Reference: Dictionary** for details.

Resource Options

There are three resource options: METAREPOSITORY, METAID, and METAAUTORESOURCES.

METAREPOSITORY specifies the name of the default repository to use on the SAS Metadata Server. If METAREPOSITORY is not specified or specifies an invalid value, a warning is written to the SAS Log and the first repository on the server is used. Using $METAREPOSITORY in your XML with PROC METADATA will resolve to the repository identifier corresponding to the repository named by the option.

METAID is an identifier unique to the current installation of SAS. The purpose of this option is to identify metadata objects that are associated with a particular installation of SAS. The installation process sets this option automatically and writes out a representation of what has been installed, identified by the unique METAID it has generated.

METAAUTORESOURCES identifies general system resources to be assigned at SAS startup. The resources are defined in a repository on the SAS Metadata Server. For example, in SAS Management Console, you can define resources describing a SAS OLAP Server or a SAS Stored Process Server. The resources include a list of librefs (library references), which are then stored as a metadata object in a repository. METAAUTORESOURCES= then identifies which predefined list of librefs to assign at startup.

METAAUTORESOURCES= accepts a URI or unique metadata object instance identifier as a resource identifier.

For a detailed description of the metadata family of system options, see the **SAS 9.1 Language Reference: Dictionary**.

Previous Page | Next Page | Top of Page

Understanding Metadata Types

Introduction

The SAS Open Metadata Interface ships metadata types in two namespaces:

- The SAS namespace defines metadata types for the most commonly used SAS application elements. These types comprise the SAS Metadata Model.

  The SAS Metadata Model provides a framework and a common format for sharing metadata between SAS applications. In addition, it provides a foundation for conversion programs to convert SAS common metadata to standard representations like the Object Management Group's Common Metadata Model XML Metadata Interchange format. This is the namespace that you use in most SAS Open Metadata Interface clients.
- The REPOS namespace is a special−purpose namespace that defines metadata types for repositories.

Other applications, like SAS ETL Studio software, may define additional special−purpose namespaces. These namespaces are described in the appropriate documentation.

Regardless of their namespace, the following rules describe a metadata type:

- Each metadata type models the metadata for a particular kind of object. Distinctions among objects are based on the conceptual identities of the objects, e.g., table, column, document, and on their behavior. Further distinctions are based on an object's origin and characteristics.
- The metadata types exist in a hierarchy. Supertypes define common behaviors for subtypes. Subtypes extend behaviors. A subtype can have subtypes of its own.
- A metadata object is uniquely described by the values of its attributes and associations.

  - The attributes describe the characteristics of the metadata object.
  - The associations describe an object's relationships with objects of other metadata types.
- User−defined attributes are supported as extensions; user−defined metadata types are *not* supported.

Documentation Overview

This reference documents metadata types in three ways:

1. Reference documentation describes each metadata type.
2. Hierarchy diagrams depict the inheritance structure for the most common metadata types (from the most abstract types to the most specialized types).
3. Association diagrams illustrate the relationships between the metadata types.

The reference documentation (SAS Namespace Types and REPOS Namespace Types) contains the details necessary to write a metadata property string. Before reading the reference documentation, see Types Documentation for important information about how the information is structured and keys that describe the columns in the attribute and association tables. Then see Understanding Associations and Overview to Hierarchy and Association Diagrams. These topics will help you to better understand the SAS Metadata Model and how the various metadata types relate to one another. For information about the specific metadata types and associations needed to create metadata objects that represent common application elements, see Model Usage Scenarios in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Understanding Associations

An association is a relationship that joins two metadata types. In the SAS Metadata Model, two associations exist between any two related metadata types. That is, an association is defined that describes the relationship from the perspective of each of the metadata types. As an example, consider the relationship between a data table and its columns. The DataTable metadata type has an association named "Columns" defined to describe its relationship to the Column metadata type. Conversely, the Column metadata type has an association named "Table" defined to describe its relationship to the DataTable metadata type. The two names refer to the same relationship; however, each association has different characteristics, including the number of object instances supported in the relationship and a required or optional nature.

An association that supports a relationship to a single object is referred to as a **single association.** The Table association from the example above is an example of a single association: a Column object can be associated with a single DataTable object. An association that supports a relationship to multiple objects is referred to as a **multiple association**. The Columns association is an example of a multiple association: a DataTable object can be associated with many Column objects.

The Table association is also an example of a required association. A Column object cannot be created without this association. However, not all single associations are required associations. An example of a single association that is optional is the PrimaryPropertyGroup association. A Column object can have a PrimaryPropertyGroup association defined to a single PropertyGroup object but it is not required to have an association to a PropertyGroup object, primary or otherwise.

The required/optional nature and number of objects that are supported by an association are expressed in the model by a cardinality figure. The cardinality figure let us know the number of related objects that can or must be associated with a given object through a particular association. Cardinality concepts are described in greater detail in the following section.

Cardinality

A cardinality defines an upper and a lower bound that is expressed with the notation *lower..upper*. In the SAS Metadata Model, the lower bound for an association is 0 or 1. If the lower bound is 0, then the association is optional. The upper bound is 1 or *n*. If the upper bound is 1, the association is a single association. If it is *n*, the association is a multiple association. The following is a summary of the cardinality combinations supported in the SAS Metadata Model:

0..1	Single, non−required association
1..1	Single, required association
0..n	Multiple, non−required association
1..n	Multiple, required association

Objects that have have a 1..1 cardinality in an association are called **dependent objects**. Dependent objects cannot be updated. If a change is needed to the dependent object's attributes, then the object must be deleted and a new object and association created. In addition, if the partner object of a dependent object is deleted, then the dependent object is deleted as well. For example, if a PhysicalTable object (PhysicalTable is a subtype of DataTable) is deleted, then all of the Column objects associated with it are also automatically deleted. A PhysicalTable object is not deleted when an associated Column object is deleted because the Columns association has a 0..*n* cardinality.

To better understand the concept of cardinality, consider the following association diagram.

In the diagram, the labels preceded by a + (plus sign) are association names. The name directly outside a box describes the association defined for the object on the opposite side. That is, PhysicalTable has a 0..n Columns association to Column. Column has a 1..1 Table association to PhysicalTable.

Association diagrams depicting the associations between other commonly used SAS namespace types are provided in Understanding the Diagrams.

Types Documentation

This guide contains reference information for the SAS and REPOS namespace types. The documentation for each metadata type includes

- a description
- the XML element used to represent the metadata type in a metadata property string
- the type's subclass name, if any
- a list of subtypes
- overview
- an Attributes table
- an Associations table.

Legend for Attributes Table

The Attributes table lists all of the attributes defined for a given metadata type, provides information about valid values, and indicates whether the attribute is required and can be updated.

Name	Description	Type	Length	Reqd for Add	Update allowed
Id	Object's repository ID	String	17	No	No
Name	A logical identifier for the object. Used for, but not limited to, display.	String	60	Yes	Yes
Desc	More detailed documentation for this object.	String	200	No	Yes

In the previous table, **Type** indicates the attribute's data type, **Length** is the attribute's allowed length, and *Yes* or *No* in the appropriate column indicates whether the attribute is required when the object is created, and whether the attribute can be updated.

Legend for Associations Table

The Associations table lists the associations that are supported for the metadata type.

Name, Subelements, and Partner Name	Description	Number of subelements	Reqd for add	Object ref	Update allowed
	Unknown	0 to *	No	Yes	Yes

Name: AccessControls Subelements: AccessControl AccessControlEntry AccessControlTemplates Partner Name: Objects					

In the previous table (which illustrates the information provided to describe a PhysicalTable object's association to access control objects):

Name, Subelements, and Partner Name *column*

Name

is the association name that describes the association from the perspective of the named object (PhysicalTable).

Subelements

are the metadata types supported as related objects in the association.

Partner Name

is the association name that describes the association from the perspective of the related objects.

In other words, the PhysicalTable metadata type has an AccessControls association to the AccessControl, AccessControlEntry, and AccessControlTemplate metadata types. In turn, the AccessControl, AccessControlEntry, and AccessControlTemplate metadata types have an Objects association to the PhysicalTable metadata type.

Description

is an optional description of the association. One is not provided here.

Number of subelements

indicates the cardinality of the association. In this example, the association is optional; a PhysicalTable can have zero or multiple access control objects associated with it.

Reqd for add

indicates whether the association needs to be defined when the object is created. *No* means it can be added later with the UpdateMetadata method.

Obj ref

indicates whether the related object is required to exist before the association can be created. In this example, *Yes* indicates that an AccessControl, AccessControlEntry, or AccessControlTemplate object must be defined before this association can be created.

Update Allowed

indicates whether the objects in the association can be updated. *Yes* indicates the objects can be updated.

The information in the previous example indicates that a PhysicalTable object does not require an AccessControls association in order to be created (the object can be updated to include the association later). However, before an association can be created, one or more AccessControl, AccessControlEntry, or AccessControlTemplate objects must have been defined first. The associations can be added to a PhysicalTable object by using the AddMetadata method or the UpdateMetadata method. The attributes of the objects in the association can also be updated with no restrictions.

Conventions

Date, Time, and DateTime Values

Date, time, and datetime information is stored on the metadata server as GMT values in SAS date, time, and datetime encoding. These values are then formatted in the XML according to a specified locale. The metadata server supports a US−English locale. You can use a different locale in the client by setting the OMI_NOFORMAT flag in a GetMetadata request. The OMI_NOFORMAT flag gets date, time, and datetime data as raw floating point values that the client can use as SAS date, time, and datetime values and format however they want.

"V" in Length Attribute

Attributes that have no practical length limitation are represented with a "V" in the Length attribute, for example, "V64". The "V" indicates the property is variable length (arbitrarily large). The documented length (64) is the maximum length of the string that can be stored before an overflow algorithm is invoked. Storing a string that exceeds the documented length causes one or more TextPage objects and corresponding associations that connect them to the original object to be created to store the string. Each TextPage object holds an additional 1,000 characters of text.

Use of the overflow algorithm has performance overhead associated with it. XMLSELECT processing also will not search overflow text in attempts to qualify an object for selection.

Previous Page | Next Page | Top of Page

SAS Namespace Types

Overview

This section describes the metadata types for defining application metadata objects. The SAS namespace types form the basis of the SAS Metadata Model.

The SAS Metadata Model defines approximately 150 metadata types. In order to make the model more understandable, types that are used to support a particular scenario are grouped into submodels. The groupings are for documentation purposes only. For an alphabetical list of types, see Alphabetical Listing of Types. For a hierarchical list of types, see Hierarchical Listing of Types.

All methods in the IOMI class are valid with the SAS namespace types.

For information about these methods, see OMI (IOMI) Class.

Submodels in the SAS Metadata Model

The following submodels have been identified to help you to navigate the model:

Analysis submodel
: includes types that are used to describe statistical transformations, multidimensional data sources, and OLAP information. For a more detailed description, see Analysis Submodel.

Authorization submodel
: includes types that are used to define access controls. Metadata objects based on security types can be associated with metadata objects describing people, repositories, and application elements to control access both to the metadata and the data that the metadata describes.

Business Information submodel
: includes types that are used to describe people, their responsibilities, and information about how to contact them, as well as business documentation and other descriptive information. For a more detailed description, see Business Information Submodel.

Foundation submodel
: includes the basic metadata types of the model, from which all other types are derived, and some utility metadata types. For a more detailed description, see Foundation Submodel.

Grouping submodel
: includes types that are used to group metadata objects together in a particular context or hierarchy. For a more detailed description, see Grouping Submodel.

Mining submodel
: includes types that are used to store analytic information associated with data mining. For a more detailed description, see Mining Submodel.

Property submodel
: includes types that are used to describe prototypes of metadata objects, parameters for processes, and properties or options for SAS libraries, data sets, connections to servers, or software commands. For a more detailed description, see Property Submodel.

Relational submodel
: includes types that are used to describe relational tables and other objects used in a relational database system, such as indexes, columns, keys, and schemas. For a more detailed description, see Relational Submodel.

Resource submodel
: includes types that are used to describe data resources such as files, directories, SAS libraries, SAS catalogs and catalog entries. For a more detailed description, see Resource Submodel.

Software Deployment submodel
: includes types that are used to describe software, servers, and connection information. For a more detailed description, see Software Deployment Submodel.

Transform submodel
: includes types that are used to describe a transformation of data. This can be a logical to physical mapping, or a set of steps that transform input data to a final result. For a more detailed description, see Transform Submodel.

XML submodel
: includes types that are used to describe XML constructs such as SAS XML LIBNAME Engine map definitions and XPath location paths. For a more detailed description, see XML Submodel.

For a list of the types in each submodel, see Types by Submodel.

Previous Page | Next Page | Top of Page

Analysis Submodel

This submodel contains the types used by the SAS OLAP server.

Metadata Types

The following types, relevant to the Analysis submodel, are classified as Container, Aggregation, and Other. Click on each type to display the type attributes and associations.

AggregateAssociation
Aggregation
Cube
Dimension
Hierarchy
Level
Measure
OLAPProperty
OLAPSchema

Container Types

OLAPSchema is a container for Dimension objects and Cube objects that can be accessed by a particular OLAP server.

Aggregation Types

Aggregation and *AggregateAssociation* types are used to represent aggregated data and the physical location of aggregated data.

Other Types

Cube represents multidimensional data. *Dimension* represents a categorization of data, organized into hierarchies. Each hierarchy is represented by the *Hierarchy* type. *Level* represents a grouping of information within a hierarchy. *Measure* represents a calculated value. *OLAPProperty* is an attribute associated with members of a given dimension level.

Usage

These types are used to store metadata about multi−dimensional data structures. This type of information is created by the OLAP server and other analytic applications.

Previous Page | Next Page | Top of Page

Authorization Submodel

Authorization contains types that are used to define access controls.

Metadata Types

The Authorization submodel has the following types. Click on each type to display the type attributes and associations.

AccessControl
AccessControlEntry
AccessControlTemplate
Permission
PermissionCondition
SecurityRule
SecurityRuleScheme
SecurityTypeContainmentRule

Previous Page | Next Page | Top of Page

Business Information Submodel

This submodel contains types used to describe people, documents, and other descriptive information.

Metadata Types

The Business Information submodel has the following types. Click on each type to display the type attributes and associations.

Document
Email
Keyword
Location
Person
Phone
ResponsibleParty
Timestamp
UnitofTime

Keyword is used to add single keyword descriptions to another metadata object. *Document* contains a URI and can be used to associate documentation to a metadata object. *Timestamp* contains a timestamp value and a role that indicates the meaning of the timestamp. *UnitofTime* is used to specify the number of a unit of time measurement such as day, hour, or week.

The *Person* type describes a particular person and can be associated with *ResponsibleParty* objects, *Email* objects, *Location* objects, and *Phone* objects. *ResponsibleParty* describes a particular role for a person, such as author or administrator. *Email* contains an email address. *Location* is used to describe an address. *Phone* contains information about phone numbers.

Usage

The primary usage of these metadata types is to further describe other metadata objects. These types are used to:

- associate documentation to an object
- identify the author, owner, or administrator of an object
- identify important date/time events, such as when the object was created or updated.

Previous Page | Next Page | Top of Page

Foundation Submodel

Foundation contains the basic metadata types from which all other types are derived and some utility metadata types.

Metadata Types

The following types relevant to the Foundation submodel are classified as Basic, Extension, Identity, and Role. Click on each type to display the type attributes and associations.

AbstractExtension
Classifier
Extension
ExternalIdentity
Feature
Identity
IdentityGroup
NumericExtension
Role
Root

Basic Types

Root is the supertype for all types in the model.

Classifier is the supertype for objects that contain data or reports made from transformed data.

Feature is the supertype for objects that contain data or other information and have a dependency on a *Classifier*. For example, Column is a *Feature* of DataTable, which is a *Classifier* ; Measure is a *Feature*, and Dimension is a *Classifier*.

Subtypes of *AbstractExtension* are used to extend a metadata type by adding additional information that is needed for a particular deployment of an application.

ExternalIdentity is associated with some other object and represents an ID for that object obtained from some other context. For example, a GUID or a DistinguishedName for an object can be stored in an ExternalIdentity object

Extension Types

AbstractExtension is the supertype for the extension types. An extension is used to "extend" or add additional attributes to any metadata type. It would be used when an application is deployed at a particular site, and the metadata type does not contain all the information that is required by the deployment. The extension will include the name of the extension and a value. An SQL type of the extension may also be specified. The *Extension* type is used for character values and the *NumericExtension* type is used for numeric values.

Identity Types

Identity and *IdentityGroup* are used for access control.

Role Types

There is only one *Role* type, and it identifies the various "roles" associated with an object. For example, a relational table may have a role of "Source" for Warehouse Administrator. This means that Warehouse Administrator (W A) reads data from that table; it is not a table created by WA. Another role may be "Summary" for the OLAP product. This identifies the way the OLAP product will use the table.

Usage

These types are primarily used as supertypes for other metadata types. The exceptions are the extension types and external identity. Extensions are used when an application is deployed at a particular site, and a metadata type does not contain information that is required by the deployment. ExternalIdentity is used to maintain information about the objects identity from another context.

Previous Page | Next Page | Top of Page

Grouping Submodel

This submodel contains types used to group metadata objects together in a particular context, as well as types to construct a hierarchy of metadata objects.

Metadata Types

The Grouping submodel has the following types. Click on each type to display the type attributes and associations.

Group
Tree

Group provides a simple grouping mechanism and has an association to a set of other metadata objects. *Tree* is like *Group*, except that it can be used to form a hierarchy. A *Tree* object may have a single Parent ree object and any number of SubTree objects.

Usage

Each of these types may be associated to a *SoftwareComponent* or *DeployedComponent* (see the Software Deployment Submodel). This provides a context for grouping metadata objects in an application hierarchy. For example, Enterprise Miner keeps its information organized in hierarchical "projects". Each project is represented as a *Tree* object. The root *Tree* is associated to a *Soft w areComponent* object that represents Enterprise Miner 5.0. When any copy of EM 5.0 is run, it can interrogate the server for the *SoftwareComponent* that represents its type of software, obtain the Tree associated with that *SoftwareComponent* , and locate its "projects".

Previous Page | Next Page | Top of Page

Mining Submodel

This submodel contains the types used to store metadata about mining models as created by SAS Enterprise Miner. It is used in conjunction with the Transform Submodel.

Metadata Types

The Mining submodel has the following types. Click on each type to display the type attributes and associations.

AnalyticColumn
AnalyticTable
FitStatistic
MiningResult
Target

A *MiningResult* is a type of *Transformation* that is used to represent an Enterprise Miner Model.

An *AnalyticColumn* contains analytic attributes to apply to a column. These attributes include information such as upper and lower limits, cost of acquiring a variable, level, and distribution.

An *AnalyticTable* contains analytic attribute to apply to a table. These attributes include information such as the sampling rate and a description of the group of observations in this table.

A *FitStatistic* is usually a measure of accuracy of a predicted value. It may also measure the performance or accuracy of a model.

A *Target* models the event or the value of interest for a mining model.

Usage

These types are used to store metadata about mining models. This type of information is created by SAS Enterprise Miner and other data mining applications.

Previous Page | Next Page | Top of Page

Property Submodel

The Property submodel is used to:

1. provide parameter information for a stored process, including the valid choices for the parameter
2. store preconfigured parameters or options used with an object such as a library, data set, or server (options for connecting to the server), or a stored process
3. create a property sheet used to drive a user interface.

Metadata Types

The Property submodel contains the following types. Click on each type to display the type attributes and associations.

AbstractProperty
AssociationProperty
AttributeProperty
LocalizedResource
LocalizedType
Property
PropertyGroup
PropertySet
PropertyType
Prototype
PrototypeProperty

AbstractProperty, LocalizedType, and *PrototypeProperty* are supertypes that aren't expected to be instantiated; they exist so their subtypes will inherit appropriate attributes and associations. You will not create objects of these types, so for the rest of this discussion, these will be ignored.

Property objects are used to contain name/value pairs, and any object may have *Property* objects. They are used to provide additional information for a particular metadata object or to provide parameter information for a *Transformation* (the metadata type used to describe a process or program). Each *Property* is required to have have an owning *PropertyType.* The *PropertyType* stores the SQL type as an integer. If the *PropertyType* is an array, there will be an association to another *PropertyType* object using the ElementType association. This object will contain the SQL type of the array elements. The *PropertyType* may also have a *StoredConfiguration*. The *StoredConfiguration* will contain additional information such as a list of enumerated values.

LocalizedResource contains information about text meant to be displayed to a user. It contains the text for a particular locale, as well as a locale identifier. A Property may have a set of *LocalizedResource* objects, one for each textual object that can be displayed and its locale.

A *Prototype* object is used to provide a cookie cutter or template for creating a set of metadata objects. For example, some of the metadata objects used to describe a Workspace Server are a *ServerComponent*, a set of *Connection* objects (*COMConnection* and *TCPIPConnection*), and possibly a *Transformation* that is the initialization process for the server. A *Prototype* object is used to describe the metadata objects used in a particular scenario, like defining a Workspace Server. A *Prototype* has *AttributeProperty* and *AssociationProperty* objects that describe the attributes and associations that are needed in the scenario. *Property* objects are used to describe name/value pairs and can be associated with any type of object, including *Prototype*.

There are several ways of grouping *Property* objects. The Properties association is available for any object, and the objects in this association are considered to be the "default" properties. This association should contain a complete set of properties for "default" usage. If there is no default, then this association should not be used. Another grouping is the *PropertySet*, which is a complete set of *Property* objects used in a particular context. The third is *PropertyGroup*, which allows *Property* objects to be grouped in a hierarchy of *PropertyGroup* objects used to drive a UI.

Usage

Case 1:
A stored process is a SAS program that may accept input from the user to select values for parameters. A stored process is described by the metadata type *Transformation*, or one of its subtypes, and will frequently need parameters filled in for correct execution. An example is a program that subsets defects data based on division. There may be a parameter called "Division" that may have one of three values: BIP, DWT, ALL. This parameter should be represented as a *Property*.

There are three ways to associate a *Property* with the object it describes: through the Properties association, through a *PropertySet* object, or through a *PropertyGroup* object.

If the *Property* is going to be displayed in a UI and have its value selected through the UI, then it should be part of a *PropertyGroup*. PropertyGroups may be organized in a hierarchy. For example, there may be a *PropertyGroup* for properties associated with Operating System and a set of subgroups for Windows 2000, OS/390 and AIX. The *PropertyGroup* for Windows 2000 would have the *Property* objects used by Windows 2000, and so on, for the other operating systems. There may be many more *Property* objects available through PropertyGroups that are never copied into a particular Properties or PropertySet grouping. For example, the default group of properties may have no host specific objects. *Property* objects however may be a v ailable through PropertyGroups that provide specific information for a specific environment.

Case 2:
If there are default settings for the stored process, then these defaults should be available through the Properties association. For example, for this stored process the Property Division would have a default setting of "All" and is included in the Properties association.

If the Data Warehousing Technology (DWT) group wants an alternative setting for the Property to be "DWT", then a copy of the *Property* should be made and associated to a *PropertySet*. When the stored process is run for the DWT group, the property set for DWT is used, not the default settings.

It is frequently the case that a *Property* may be duplicated, one copy in the Properties list, and another in the PropertySet list, and yet a third in the PropertyGroup list, because we decided that there is no merging of *Property* objects between the various grouping mechanisms. This was a design trade−off, and merging was viewed as being too complicated.

Properties are for a full set of properties with the default values filled in. No hierarchy is allowed.
PropertySets are for a full set of properties with values filled in for a specific usage. No hierarchy is allowed.
PropertyGroups are used to drive a UI and may be grouped in hierarchies.

Another Case 2:
The second case, using a *SASLibrary* as an example, is similar to the previous one, except that typically a *PropertyGroup* would not be used. The default options for a *SASLibrary* would be available through the Properties association, and alternative settings would be available through PropertySets. Instead of having PropertyGroups associated directly with a *SASLibrary*, a property sheet is created by using the metadata type *Prototype*.

Case 3:
A *Prototype* is a metadata type that describes the settings of another type in a particular environment. For *SASLibrary* objects, there might be a *Prototype* object for DB2 libraries, another for Oracle, and another for base. The *Prototype* would have a top−level *PropertyGroup*, which in turn may have nested PropertyGroups that contain the possible settings for attributes, *Property* objects, and associated objects. A *Prototype* should not use either the Properties or PropertySets associations, because it is used only as a template for creating other metadata types. All *Property* objects used with the Prototype should be organized in PropertyGroups.

Previous Page | Next Page | Top of Page

Relational Submodel

The Relational submodel contains the metadata types used to describe relational tables and other objects used in a relational database system, such as indexes, columns, keys, and schemas.

Metadata Types

The following types, relevant to the Relational submodel, are classified as Container, Table, Column, Key, and Other. Click on each type to display the type attributes and associations.

Column
Column Range
DatabaseSchema
DataTable
ExternalTable
ForeignKey
Index
JoinTable (Deprecated)
Key
KeyAssociation
LogicalColumn
PhysicalTable
QueryTable
RelationalTable
SASPassword
UniqueKey
WorkTable

Container Types

These types are subtyped from *TablePackage* in the Resource Submodel. The only container defined in the Relational submodel is *DatabaseSchema*.

Table Types

DataTable is the supertype and has an association to *Column* that is inherited by all of the table types. *RelationalTable* is used to represent a description of a table that does not have a physical representation. A able described in a data model would be represented as a *RelationalTable*. *PhysicalTable* is a relational table with a fixed physical location, such as in a particular database schema.

Other table types represent tables that do not have a fixed location. *QueryTable* is a transient relational table that is the result set from execution of an SQL statement. *WorkTable* represents a table in the SAS Work library; it is transient and exists only for the lifetime of a SAS session.

ExternalTable represents tables that contain information similar to a relational table, but do not reside in a DBMS. Excel data or comma separated lists are represented using this type.

Column Types

LogicalColumn is a supertype used to describe column–like objects. It is not a part of a relational table. *Column* is the type used to describe columns in a relational table. *ColumnRange* represents a range of columns

that have the same characteristics and which will all be acted upon in the same manner for any transformation or query.

Key Types

Key is the supertype for *UniqueKey*, which represents unique and primary keys, and *ForeignKey*, which represents foreign keys. The *KeyAssociation* object is used to associate columns used in a foreign key with columns in the unique key.

Other Types

Index is used to represent an index associated with a table, and *SASPassword* represents the SAS password that is used for an encrypted table.

Usage

Relational tables should be associated with a data package, for example, a *DatabaseSchema* or *SASLibrary*. The schema or library is associated to the component where the physical data resides. In addition, a *SASLibrary* may be associated to a *Directory* or set of *Directories* which contain the file system path. A table has a set of columns represented by the *Column* type. For more information about defining keys see Creating Metadata for Tables, Columns, and Keys in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Previous Page | Next Page | Top of Page

Resource Submodel

The Resource submodel contains the metadata types used to describe physical objects such as files, directories, SAS libraries, SAS catalogs, and catalog entries.

Metadata Types

The types listed below are categorized as Container, Content, and Other types. Click on each type to display the type attributes and associations.

ArchiveEntry
ArchiveFile
ContentLocation
ContentType
DeployedDataPackage
Device
DeviceType
Directory
File
Memory
SASCatalog
SASCatalogEntry
SASFileRef
SASLibrary
Stream
Text
TextStore

Container Types

Directory, *SASLibrary*, *SASCatalog*, and *TablePackage* describe containers for data and are subtyped from the type *DeployedDataPackage*. *TablePackage* is the supertype for all data containers that contain relational tables. *DatabaseSchema* and *OLAPSchema* are also subtyped from *DeployedDataPackage* but are contained in the Relational Submodel and Analysis Submodel.

Content Types

These are types that describe physical location and are subtyped from *ContentLocation*. These include file types, text types, and other location types.

Text Types

Text types are supertypes for Content types and represent a physical container for data. Unfortunately, Text is a misnomer for this type, because binary as well as textual information might be stored in objects represented by this type. Types th a t represent files, SAS catalogs, and URL's as well as *TextStore*, which contains data stored directly in the metadata repository, are all subtyped from *Text*. *Document*, which is the type that represents a URI, is also subtyped from *Text*.

File Types

File types are *File* and *ArchiveFile*. An *ArchiveFile* may be a TAR or Zip file and acts as both a file and a container for *ArchiveEntry* objects.

Entry Types

These are types that describe objects that are an entry in some other physical object. These types are *SASCatalogEntry*, which is an entry in a *SASCatalog*, and *ArchiveEntry*, which is an entry in an ArchiveFile.

Device Types

These types represent other physical resources. They are *Device*, which is used to represent printers and terminals, *Memory* in a computer, and *Stream*, *Connection*, and *Email* which represent ways of delivering information. *DeviceType* represents information about supported devices including display information.

Other Types

SASFileRef is used specifically with SAS software to identify the physical location used by SAS software. *Report* is a generic *Classifier* (refer to the Foundation Submodel) used to describe the result of transforming data into another representation such as a textual table or a graph.

Usage

Resource submodel is used to define the location and type of data sources and how the data is to be delivered.

A file, for example, has a name and a physical location of a file system. The *File* type is used with an associated *Directory*. In addition, this file may be referenced by a *SASFileref*.

A Zip file is considered an *ArchiveFile*. The *ArchiveFile* contains entries represented by an *ArchiveEntry*. There is also an associated *Directory*.

SAS applications may need to reference a *SASCatalog*. The SASCatalog also has associated *SASCatalogEntry* objects. A *SASCatalog* is located by a SAS application through the *SASLibrary*. For more information about defining a < i >SASLibrary refer to the example Usage Scenario: Creating Metadata for a SASLibrary in the **SAS 9.1 Open Metadata Interface: User's Guide**.

A *Report* may have a a location in a file system. If it does, there would be an associated *File* or *Directory* if the report is actually a collection of files. A *Report* may also be delivered in a variety of ways. For example, the *Report* may be sent directly to a printer, *Device*, or an email account. A *Report* may be a static item that is stored but it could be created by a *Transformation* which is also defined in the metadata. The resulting report would be a *ClassifierTarget* of a *ClassifierMap*. For more information on defining Transformations refer to the Usage Scenario: Creating Metadata for a Stored Process in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Data sources may be requested by a variety of devices. Each device has limitations on the content that may be displayed. *ContentType* should be associated with any type of content to aid applications in determining if the content may be used, a n d if so, then what is the best way to use it.

SAS 9.1 Open Metadata Interface: Reference

Software Deployment Submodel

This submodel contains types used to describe software, servers, and connection information.

Metadata Types

The following types, relevant to the Software Deployment submodel, are classified as Software, Connection, Service, and Other. Click on each type to display the type attributes and associations.

AuthenticationDomain
COMConnection
ConfiguredComponent
Connection
DeployedComponent
LogicalServer
Login
Machine
NamedService
OpenClientConnection
SASClientConnection
ServerComponent
ServerContext
ServiceComponent
ServiceType
SoftwareComponent
TCPIPConnection

Software Types

SoftwareComponent represents a type of software, for example SAS/ACCESS for ORACLE V9.0. *DeployedComponent* describes software that is actually installed. An actual installation of SAS/ACCESS for ORACLE on any machine is a *DeployedComponent*. An installation of software may have various configurations that are defined to run the software. This submodel contains several subtypes of *DeployedComponent* that represent different types of configured software. *ConfiguredComponent* represents software that is configured to run. This includes software such as Java components or customizers. *ServiceComponent* represents software that acts as a service. A *ServerComponent* is used to represent servers and spawners.

LogicalServer represents a grouping of homogeneous servers for the purpose of load balancing or pooling. It is important to note that IOM workspace servers will always use the logical server construct even if it is a single server.

ServerContext groups non−homogeneous servers that all share common resources. The *ServerContext* gives an application context for the grouped servers.

Connection Types

These types contain information about how to communicate with a *DeployedComponent*. *Connection* is the parent class with two primary subtypes, *OpenClientConnection* and *SASClientConnection*. *SASClientConnection* was create d because of specific limitations SAS software has on connection information, such as the eight character length limitation on the name used to refer to the server.

TCPIPConnection and *COMConnection* are the two subtypes of *OpenClientConnection*, and each contains attributes that provide the protocol specific connection information.

Service Types

There are two types in this category: *ServiceType* and *NamedService*. *ServiceType* contains descriptive information about the types of services that are provided by a *DeployedComponent*. DBMS may be a *ServiceType* specifie d for an Oracle or DB2 server. *NamedService* contains the name used by a Naming Service, such as the RMI registry, or Active Directory, to refer to a *DeployedComponent*.

Other Types

The other types in the Software Deployment submodel are *Machine*, *Login*, and *AuthenticationDomain*. *Machine* is used to identify the computer that can run a *DeployedComponent*. *Login* contains a user ID and password . *AuthenticationDomain* is used to identify which *Login* objects can be used with which *Connection* objects. For example, CARYNT might be an *AuthenticationDomain*, and any *Login* associated with that domain can be used with an y *Connection* object associated with that domain.

Usage

The types in the Software Deployment submodel are used to define the software that is found in an enterprise and information about how to initialize and access the software. The types are generic in nature since it is impractical to have a metadata type f or every software system or every type of connection. In most use cases, these types will have associated properties which give more specific information about the deployment of the system or provide additional options that may be used. A prototype will typically be defined for these classes which represent software that is supported by an application. Many of these prototypes are installed by the SAS Management Console during the initialization of a new repository. For more information about using the Software Deployment submodel refer to the Usage Scenario: Creating Objects which Represent a DBMS in the **SAS 9.1 Open Metadata Interface: User's Guide**. The Usage Scenario: Creating Metadata for a Workspace Server also contains information about using a prototype to create the metadata definition.

Previous Page | Next Page | Top of Page

Transform Submodel

The Transform submodel is used to:

1. describe stored processes
2. represent ETL processes
3. define queries
4. schedule processes
5. give initialization information for Software Components.

Metadata Types

The following types, relevant to the Transform submodel, are classified as Abstract, Event, Query, Process, and Scheduling.

AbstractJob
AbstractTransformation
Event
FeatureMap
GroupByClause (Deprecated)
HavingClause (Deprecated)
JFJob
Job
Join (Deprecated)
OnClause (Deprecated)
OrderByClause
QueryClause
RowSelector
Select
StepPrecedence
Transformation
TransformationActivity
TransformationStep
Variable
WhereClause (Deprecated)

Abstract Types

AbstractJob, AbstractTransformation, QueryClause are supertypes that aren't expected to be instantiated; they exist so their subtypes will inherit appropriate attributes and associations.

Event Type

Events are used to describe the conditions that must occur to drive other processes.

Query Types

GroupByClause, *HavingClause*, *OnClause*, *WhereClause*, and *Join* all have been deprecated from the model. They will be removed once all changes have been made by Information Map. For the purposes of this discussion, we w i ll focus on the current types.

Select represents a query process. The query is stored as text in the SourceCode association of the *Select* object. The query may contain strings which should be replaced by a value. The *Variable* type will contain information and associations which help determine which strings should be replaced and which value should be used.

RowSelector and *OrderByClause* are used by *Transformation* types and subtypes to further qualify the transformation.

Process Types

Process types are used to define a process. A process may be a stored process.& In this case, the code for the process is stored and additional associations give information about the inputs and outputs and where the process can be run. The process could also be a process in which an application will be generating the code, based upon the associated inputs and outputs, and the location or *DeployedComponent* which will be running the generated process.

Impact analysis is one of the value propositions to the way in which the metadata is defined. The metadata for a process contains all the information about the sources and the targets. There for if a change is made to any source, it is easy to identify the process and targets which may be impacted by the change.

The *TransformationActivity* is a grouping of *TransformationSteps.*At this level, the TransformationSources and TransformationTargets associations represent the initial inputs to the activity and the final output of the activity. Fore more detailed information about what is happening within the activity the application should drill down first to the *TransformationSteps*, then *Transformations.* A *TransformationStep* is a grouping of *Transformations. Transformations* include *ClassifierMap* and *Select* (discussed under Query Types). A *ClassifierMap* shows the mapping between *Classifiers*. Examples of *Classifier* include *PhysicalTable* or *Report*. A *Classifier* often has features, for example, *PhysicalTable* has *Columns*. These features are mapped by using a *FeatureMap. StepPrecendence* is used to show the order of steps within an activity. If there is no *StepPrecendence* defined, then it is assumed that the steps may run in parallel.

Scheduling Types

Once a process has been defined and tested the process may be scheduled. The *Job* type groups *TransformationActivities* into a runtime unit to be rescheduled. *JFJob* represents a job which is scheduled in the LSF Job Flow.

Usage

Case 1: Describe Stored Processes
The stored process begins with a ClassifierMap and has associations to the SourceCode that is to be run, the component(s) which can run the process (ComputeLocations), the inputs (ClassifierSources), and the outputs (ClassifierTargets) of the stored process. For more detailed information, refer to the Usage Scenario: Creating Metadata for a Stored Process in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Case 2: Represent ETL Processes
The ETL uses a ClassifierMap and the associated FeatureMaps to show the mapping of data through a process flow. The ClassifierMaps are reusable entities that are grouped together in TransformationSteps. TransformationSteps are also reusable and are grouped into TransformationActivities. A TransformationActivity may also be reused and would be grouped by a Job. The Job is the unit which defines the process which is to be run. The Job may be scheduled to run as a batch process but may be triggered by various external or internal events. StepPrecedence is used here to show the order of TransformationSteps. Each level of the ETL process may have different locations where the process should be run. The

ComputeLocations association is used here also to show the components which are capable of performing the process.

Case 3: Define Queries
A query uses the Select, which is a subtype of ClassifierMap, to define the SQL query. The query is stored as SourceCode and may contain substitution strings. The associated Variable objects will contain information about which string to replace and where to get the value that should be used. The Select object will also use the ClassifierSource and ClassifierTarget associations to document the inputs and outputs of this query.

Case 4: Schedule Processes
Any process defined in the metadata may be scheduled to run as a job. It is required that the process be part of a TransformationActivity. TransformationActivities may then be grouped together as Job.

Case 5: Give Initialization Information for Software Components
A Deployed Component (or its subtypes) may need initialization information that is used at startup. The DeployedComponent would have an associated InitProcess. The Transformation type is usually used to represent this process, and startup information that is needed is associated to the Transformation using the TransformationSources association. For an example of an InitProcess, refer to the Usage Scenario: Creating Metadata for a Workspace Server in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Previous Page | Next Page | Top of Page

XML Submodel

XML contains types that are used to describe XML constructs such as SAS XML LIBNAME Engine map definitions and XPath location paths.

Metadata Types

The XML submodel consists of the following types. Click on each type to display the type attributes and associations.

SXLEMap
XPath

SXLEMap is the root node for a SAS XML LIBNAME Engine map definition.

XPath is used to store an XPath location path.

Usage

These types are used by the SAS XML LIBNAME Engine to aid in defining the XML mappings.

Previous Page | Next Page | Top of Page

SAS Namespace Types

Alphabetic listing of types

Class Name	Description
AbstractExtension	Abstract Extension
AbstractJob	AbstractJob
AbstractProperty	AbstractProperty
AbstractTransformation	Abstract Transformation Type
AccessControl	Access Control
AccessControlEntry	Access Control Entry
AccessControlTemplate	Access Control Template
AggregateAssociation	AggregateAssociation
Aggregation	OLAP Aggregation
AnalyticColumn	Analytic Column
AnalyticTable	Analytic Table
ArchiveEntry	ArchiveEntry
ArchiveFile	ArchiveFile
AssociationProperty	AssociationProperty
AttributeProperty	Attribute Property
AuthenticationDomain	Authentication Domain
Change	Change
Classifier	Classifier
ClassifierMap	Classifier Map
Column	Columns
ColumnRange	ColumnRange
COMConnection	COM Connection
ConditionalPrecedence	ConditionalPrecedence

ConfiguredComponent	ConfiguredComponent
Connection	Connection Information
ContentLocation	ContentLocation
ContentType	ContentType
Cube	OLAP Cube
DatabaseCatalog	DatabaseCatalog
DatabaseSchema	Database schema
DataSourceName	DataSourceName
DataTable	All Table Types
DeployedComponent	Deployed Component Abstract Type
DeployedDataPackage	Deployed Data Package
Device	Device
DeviceType	DeviceType
Dimension	OLAP Dimensions
Directory	Operating System Directories
Document	Web Documentation Items
Email	E−Mail Address
EMModel	EMModel
EMRules	EMRules
Event	Transformation Events
Extension	Object Extensions
ExternalIdentity	ExternalIdentity
ExternalTable	External Data Table
Feature	Feature
FeatureMap	Feature map
File	File
FitStatistic	Fit Statistic

ForeignKey	Foreign Keys
Group	Metadata Groupings
GroupByClause	Group By Clause
HavingClause	SQL Having Clauses
Hierarchy	OLAP Hierarchy
Identity	Identity
IdentityGroup	IdentityGroup
Index	Index
ITChannel	ITChannel
ITContentSubscriber	ITContentSubscriber
ITEventSubscriber	ITEventSubscriber
ITFilter	ITFilter
ITMap	ITMap
ITModel	ITModel
ITMsmqModel	MS Message Queue Model
ITQueueAlias	Queue Alias
ITRendModel	ITRendModel
ITSubscriber	ITSubscriber
ITTransportAlias	Transport Alias
JFJob	JFJob
Job	Job
Join	SQL Join
JoinTable	Table Joins
Key	Key Abstract Type
KeyAssociation	Key Associations
Keyword	Keyword
Level	Level

LocalizedResource	Localized Resource
LocalizedType	Localized Type
Location	Location
LogicalColumn	Logical Column: supertype for column types
LogicalServer	LogicalServer
Login	Login Definitions
Machine	Machines
Measure	OLAP measure
Memory	Memory
MiningResult	Data Mining Result
NamedService	NamedService
NumericExtension	Numeric Extension
OLAPProperty	OLAP Property
OLAPSchema	OLAP Schema
OnClause	SQL ON Clauses
OpenClientConnection	Open Client Connection Abstract Type
OrderByClause	Order By Clause
Permission	Permission
PermissionCondition	Permission Condition
Person	People
Phone	Phone
PhysicalTable	Physical Table
Property	Property
PropertyGroup	Property Group
PropertySet	PropertySet
PropertyType	Property Type
Prototype	Prototype

PrototypeProperty	PrototypeProperty
PSColumnLayoutComponent	PSColumnLayoutComponent
PSGridLayoutComponent	PSGridLayoutComponent
PSLayoutComponent	PSLayoutComponent
PSPortalPage	PSPortalPage
PSPortalProfile	Portal Profile
PSPortlet	PSPortlet
QueryClause	Query Segment
QueryTable	Result of a query
RelationalSchema	Table Package
RelationalTable	Table
Report	Report
ResponsibleParty	Responsible Party
Role	Role
Root	Metadata Root Abstract Type
RowSelector	Row Selector
SASCatalog	SAS Catalog
SASCatalogEntry	SAS Catalog Entry
SASClientConnection	SAS Client Connection
SASFileRef	SASFileRef
SASLibrary	SAS Library
SASLicense	SASLicense
SASPassword	SAS Passwords
SecurityRule	Rule that affects the way authorization decisions are made.
SecurityRuleScheme	Scheme groups related SecurityRules
SecurityTypeContainmentRule	SecurityTypeContainmentRule
Select	SQL Select

ServerComponent	ServerComponent
ServerContext	ServerContext
ServiceComponent	ServiceComponent
ServiceType	Service Type
SoftwareComponent	Software Component
StepPrecedence	Step Precedence
Stream	Stream
SummaryStats	Summary Statistics
SXLEMap	SXLEMap
SyncStep	SyncStep
TableCollection	TableCollection
Target	Target
TCPIPConnection	TCP Connection Abstract Type
Text	Text
TextStore	Text Store
Timestamp	Timestamp
Transformation	Transformation
TransformationActivity	Transformation Activity
TransformationStep	Transformation Step
Tree	Metadata Trees
UniqueKey	Unique Keys
UnitofTime	UnitofTime
Variable	Variable
WhereClause	SQL Where Clauses
WorkTable	Work Table
XPath	XPath

SAS 9.1 Open Metadata Interface: Reference

SAS Namespace Types

Type Hierarchy

- **Root** (This is an abstract type that is the supertype for all of the other metadata types.)
    - **Hierarchy** (This type represents an OLAP hierarchy, ie. an ordered list of categorical information.)
    - **SummaryStats** (DEPRECATED: SummaryStats)
    - **Aggregation** (This type represents aggregations stored in a multidimensional object, or mapped to other data objects.)
    - **AggregateAssociation** (DEPRECATED: This type associates an Aggregation to either a Level or a Measure and to the physical object (Column or some other persistence object) that contains the data.)
    - **AbstractTransformation** (The parent type for transformation types. Transformation types are used to document transformation of data. This includes extraction, transformation, and loading of data, stored procedures, and logical to physical mappings.)
        - ♦ **TransformationStep** (TransformationStep contains mapping objects. They may be ordered using an association to StepPrecedence.)
            - ◊ **SyncStep** (The SyncStep is used to help define a workflow. This step does not represent a process step but instead a point if synchronization to help define complex flow algorithms. Valid roles for this type are: AND, OR, EVENTWAIT, CASE An AND SyncStep would have multiple PredecessorDependencies which all must occur before the SuccessorDependencies are activated. An OR SyncStep would have multiple PredecessorDependencies. Any one of these would have to occur before the SuccessorDependencies are activated. An EVENTWAIT SyncStep is activated by TriggeringEvents. A CASE SyncStep signals the start of case logic.)
        - ♦ **TransformationActivity** (TransformationActivity contains TransformationSteps.)
        - ♦ **FeatureMap** (FeatureMap is used to map any number of Feature objects to any number of Feature objects.)
        - ♦ **Transformation** (This type is used for generic transformations such as stored procedures.)
            - ◊ **ClassifierMap** (ClassifierMap is used to Map any number of Classifier objects to any number of Classifier objects.)
                - · **Select** (Used to document an SQL select statement.)
                - · **Join** (DEPRECATED: The output of a Join is limited to one JoinTable. A Join may have at most two inputs. A Join may not exist without a Select. For nested joins a Join should be documented for each pair of joined tables. For example: Table1 is joined to Table2 which results in JoinTable1. JoinTable1 is then joined with Table3 to create JoinTable2. JoinTable2 now acts as in input into a Select which documents the rows and columns that are to be included in the final output table.)
            - ◊ **AbstractJob** ( )
                - · **Job** (Groups transformation activities to be an actual runtime unit.)
                - · **JFJob** (Represents a job that has been or will be scheduled in the LSF JobFlow.)

◊ **MiningResult** (SAS Enterprise Miner Models)

- **QueryClause** (This abstract type represents types that are used to define the transformations that are performed. These include types that subset, group and order data.)
  - ♦ **WhereClause** (DEPRECATED: This type represents a Where SQL Clause.)
  - ♦ **RowSelector** (Used to document a SAS datastep select statement.)
  - ♦ **GroupByClause** (Has an association to the columns used to group this data.)
  - ♦ **HavingClause** (DEPRECATED: This type represents a Having clause in an SQL statement.)
  - ♦ **OrderByClause** (Has an association to columns used to order the data.)
  - ♦ **OnClause** (DEPRECATED: This type represents an SQL ON clause.)

- **Event** (An event is something that describes a condition that occurs that drives other things, ie. jobs, to be processed.)

- **Variable** (Used to help define substitution strings and the replacement values.)

- **Feature** (A feature is a property which is encapsulated within a Classifier. In the metamodel, a Feature declares a structural or behavioral characteristic of an instance of a Classifier or of the Classifier itself. Feature is an abstract metaclass.)
  - ♦ **Level** (A grouping of information that can be arranged in a hierarchy. For example, levels of the time dimension could be year, quarter, month, week, day, hour, minute and second.)
  - ♦ **OLAPProperty** (OLAPProperty is an attribute associated with members of a given dimension level. It is a feature of a Dimension, and can associated with a Level, and to a Column through a FeatureMap.)
  - ♦ **StepPrecedence** (The preceding and succeeding steps.)
    - ◊ **ConditionalPrecedence** ( )
  - ♦ **LogicalColumn** (This type is the supertype for Column which is part of a relational table, and Measure, which is used with OLAP types.)
    - ◊ **Measure** (Measure for OLAP)
    - ◊ **Column** (This type represents a column in a table, view, etc.)
      - · **ColumnRange** (Represents a range of columns.)

- **Classifier** (A classifier is an element that describes structural and behavioral features. In the metamodel, a Classifier may declare a collection of Features. Classifier is an abstract metaclass.)
  - ♦ **Cube** (A logical, multidimensional object comprised of a set of cube regions. Each Cube is associated with a set of dimensions, hierarchies, and measures.)
  - ♦ **Dimension** (A categorization of information. Examples are time, geography, product.)
  - ♦ **DataTable** (This is an abstract type that represents an object that contains a set of columns.)
    - ◊ **ExternalTable** (This type represents an table in an external data file.)
    - ◊ **RelationalTable** (This type represents a relational table, one that does not have a physical representation. If the table resides in a DBMS or file system, it should be represented by a PhysicalTable. If it is a transient table, one that has a physical representation but is temporary, it should be represented as a WorkTable.)
      - · **PhysicalTable** (A PhysicalTable is the "materialized" table that resides in a database or a file system.)
        - **WorkTable** (This type represents a table that is created as part of running SAS code that is not intended to be persisted. Usually, these types of tables are in the WORK library. These are a separate type from WTable and PhysicalTable because they have a transient physical existence and are not persisted.)

        - **QueryTable** (Result set of a query.)
            - **JoinTable** (DEPRECATED: This type represents an instance of a join between two tables. This is used in the process submodel.)
        - **TableCollection** (A classifier that defines a set of tables that all share the same characteristics but that should be treated as one classifier for display purposes. It is often not known until the table definition is used how many physical representations will exist for a given definition.)
    - **Index** (An index represents an index on a physical structure. It is tied to the columns that make up the index.)
    - **Report** (This type represents the result of data being transformed into a report. The report has an association to a physical location, such as a SAS catalog entry or file)

- **AbstractExtension** (The parent type for Extension and NumericExtension)
    - **Extension** ( )
    - **NumericExtension** (This allows an application to create a numeric extension to any object.)

- **Role** (This object describes a role of of an object, and the context of the role. Defined contexts are: ContentType WA (abbreviation for Warehouse Administrator) EM (abbreviation for EnterpriseMiner) For ContentType, valid roles are BASE, NWAY, AGGREGATE, STARFACT, STARTDIM For WA, valid roles are Source, Target For EM, valid roles are Training The list of roles will be expanded, along with the list of contexts)

- **Identity** (Identity – abstract class for objects which identify an entity.)
    - **IdentityGroup** (Grouping mechanism for identity objects. A IdentityGroup may also be a member of another IdentityGroup)
    - **Person** (This type represents the information kept about a person.)

- **ExternalIdentity** (DRAFT: This is an ID associated with the object in some other context, such as a GUID or DN for LDAP .)

- **SASPassword** (This type represents the storage of SAS passwords for SAS tables or Connections. There will be one of these for each type of password associated with a file.)

- **Key** (This type represents a keys on tables.)
    - **UniqueKey** (This type is a representation of the unique and primary keys of a table. It is also associated with the columns in the table that comprise the key.)
    - **ForeignKey** (This type represents the fact that a table has a foreign key in it to another table.)

- **KeyAssociation** (This type represents the relationship between 2 columns in a foreign key, unique key relationship. For example, if table a has a unique key of columns a1 and a2, and table b has a foreign key, of columns b1 and b2, into table a, there is a keyassociation between the columns that relate to each other, for ex. b1 to a1 and b2 to a2.)

- **Tree** (This type represents a tree which is the root node for a hierarchy of groups of metadata.)

- **Group** (This type represents the ability to group metadata together.)
    - **SXLEMap** (DRAFT: Root node for SXLE map definition.)

- **DeployedDataPackage** (A Deployed Data Package is a container of data files (files that contain data) or other Deployed Data Packages. This type has associations with DeployedComponents that

can access the package. This type could be a directory, a database catalog or schema or a SAS library. For example, this data package allows you to have different library definitions that all access the same set of files. These different library definitions would be associated to the SAS DeployedComponent that can use those library definitions. Database schemas and catalogs should be associated with the DBMS that provides access to those packages.)

- **OLAPSchema** (This is a grouping of Cubes accessible from an OLAP server.)
- **DatabaseCatalog** ( )
- **SASCatalog** (SASCatalog)
- **RelationalSchema** (This is an abstract type that acts as the supertype for any type that can contain table objects.)
  - **DatabaseSchema** (This type represents a schema in a relational database. Schemas can contain database tables and views.)
  - **DataSourceName** ( )
  - **SASLibrary** (This type represents a SAS library statement.)
- **ContentLocation** (This is the supertype for all types that provide location information.)
  - **Text** (This is a superclass for files, SAS catalog entries and text stored in the repository itself.)
    - **TextStore** (This class represents storing text in the repository.)
    - **SASCatalogEntry** (This type represents a SAS catalog entry.)
    - **File** (The type represents a file in the file system)
      - **ArchiveFile** (This type represents an archive, such as a tar file or a zip file, which may contain entries.)
    - **ArchiveEntry** (This is an entry in an archive file.)
    - **Document** (A Document is a web page that contains documentation pertinent to the object to which this document is related.)
  - **Directory** (A directory represents a physical operating system path.)
  - **Device** (This type is used to describe location information for SAS Filerefs using the following engines: PIPE PLOTTER PRINTER TAPE TERMINAL XPRINTER)
  - **Stream** (This type represents a stream of data. The protocol attribute describes the format of the data.)
  - **Memory** (This represents a result of a transformation that resides in memory on the server.)
  - **Connection** (This type represents the information required to connect to a server. It is associated with the deployed component, i.e. server, that can be accessed using the connection information. It may also be associated with the deployed components that can use the connection information to access the server..)
    - **SASClientConnection** (This type represents the information needed by SAS software to connect to other servers.)
    - **OpenClientConnection** (This abstract type represents the connection information needed by an open client to access a deployed component.)
      - **TCPIPConnection** (This type contains information for making a TCP/IP connection to a server.)
      - **COMConnection** (This type represents a COM connection. The communication and application protocol attributes should be set to "COM".)
  - **Email** (An e−mail address.)
- **SASFileRef** (This type represents a SAS fileref. The associated ContentLocations (and this should be a single object except for concatenated files) represent the location used by the fileref.)

- **ContentType** (A content type for each supported type will be created in the metadata. Content

creators will associate content with a ContentType. This will allow application that deliver content to determine if conte nt may be displayed and/or the best way to display the content.)

- **DeviceType** (This type is used to define a prototype for a specific supported device.)

- **AuthenticationDomain** (This type represents an authentication domain. A set of credentials (such as userid/password) can be used to access a set of servers that share an authentication domain.)

- **SoftwareComponent** (A component is the representation an application. This type would be used to define an application hierarchy. An application would use the SoftwareComponent which represents the application as a starting place to navigate through the hierarchy which is made of Trees and Groups and their members.)
    - ♦ **DeployedComponent** (A deployed component is the representation of installed software. This software may be or may not be licensed as indicated by the IsLicensed attribute.)
        - ◊ **ConfiguredComponent** (Installed software that has configuration information in addition to the licensing information.)
            - · **ServerComponent** (Represents server components that are configured to run.)
                - **LogicalServer** (Used to define a homogeneous group of servers.)
                - **ServerContext** (Used to define an application context for non–homogeneous group servers.)
            - · **ServiceComponent** ( )

- **Login** (This type represents login information for a given user in a given authentication domain.)

- **Machine** (This type represents a physical machine.)

- **ServiceType** (The types of services available from a DeployedComponent. Examples of types are DBMS, FileSystem, OperatingSystem.)

- **NamedService** (This type describes the name of the service and how the name is used (as rebind, bind, or lookup). This type is associated to one DeployedComponent which is being named, and to (possibly multiple) DeployedComponents which provide a naming service.)

- **SASLicense** (This type is used to document the setinit information for SAS products.)

- **ResponsibleParty** (This type is used to associate a set of Person objects with a particular role or responsibility. This object may be associated with other objects. For example, the ResponsibleParty may have the role of "Owner", and have an association with a Person object that represents the owner, and associations with the objects that are owned.)

- **Location** (This type represents a location that includes a type (is it a street address, or an office number), as well as address information (street, city, country).)

- **Phone** (A phone number and the type of phone number (home, office, mobile, fax).)

- **Keyword** (The name attribute of this object is a single word that can be used to identify this object, or a set of objects.)

- **Timestamp** (This object contains a timestamp and the role of the timestamp.)

- **UnitofTime** ( )

- **LocalizedResource** (This type is associated with a particular locale and contains a name/value pair used for displaying information about a Property. The 'Name' attribute of this type indicates what is being loc alized, ie Label, Description, Icon, Help,)

- **LocalizedType** (This is the supertype for all of the types that have localized resources. This is an abstract type.)
    - ♦ **PropertyGroup** (This represents a group of properties. The properties that are part of this group are associated using the GroupedProperties association.)
    - ♦ **PropertyType** (This contains a property type and its validators, customizers, and editors. There will be additional types that are associated with this type that define the validators, customizers and editors.)
    - ♦ **Prototype** (This type represents a prototype or template for a type of metadata.)
    - ♦ **AbstractProperty** (This is the supertype for all of the property types.)
        - ◊ **Property** (A parameter, option or other type of information that is used to tailor a metadata object.)
        - ◊ **PrototypeProperty** (This is the super type for types that define properties that are for prototype objects only. These are either AttributeProperty or AssociationProperty objects.)
            - · **AttributeProperty** (This type is used to describe an Attribute of a particular metadata type.)
            - · **AssociationProperty** (This property defines an association between Prototype objects.)

- **PropertySet** (This contains a set of properties to be used in a particular context. The properties that are part of this set are associated using the SetProperties association.)

- **AnalyticColumn** (Analytic attributes to apply to a column.)

- **AnalyticTable** (Analytic attributes to apply to a table.)

- **FitStatistic** (Usually a measure of the accuracy of a predicted value. Maybe a measure of the accuracy/performance of the model.)

- **Target** ( )

- **XPath** (DRAFT: Used to describe an XPath location path.)

- **Change** (DRAFT. A change made to the repository.)

- **Permission** (Permission)

- **PermissionCondition** (PermissionCondition)

- **AccessControl** ( )
    - ♦ **AccessControlEntry** (AccessControlEntry)
    - ♦ **AccessControlTemplate** (AccessControlTemplate)

- **SecurityRuleScheme** (This class is used to group SecurityRules into a unit. The rules in a scheme were generally created together and will most likely need to be deleted together. The SecurityScheme provides t he common link needed to facilitate this functionality.)

- **SecurityRule** (This is an abstract class from which other "SecurityRules" will derive. Security

rules are used to tell the authorization facility the rules it should use when making authorization decisions.)
    - **SecurityTypeContainmentRule** (A SecurityTypeContainmentRule tells the authorization facility how find parent objects for a particular metadata type. There may be more than one rule for any class.)

- **EMModel** (SAS Enterprise Miner Models)

- **EMRules** (This type represents Enterprise Miner Rules)

- **PSPortalProfile** (A Portal user's Portal information, which defines the basic functionality of the Portal)

- **PSPortalPage** (A PortalPage gathers and displays information in the Portal, and can be customized.)

- **PSPortlet** (A component that performs a specific task within a Portal.)

- **PSLayoutComponent** (A UI element that designates how Portlets should be laid out on a PortalPage.)
    - **PSColumnLayoutComponent** (Provides the information needed to lay a PortalPage out in columns.)
    - **PSGridLayoutComponent** (Object for creating a grid UI layout.)

- **ITTransportAlias** (Alias for a queue service transport.)

- **ITQueueAlias** (Provides an alternate name for a queue.)

- **ITMsmqModel** (A model for Microsoft Message Queuing queues.)

- **ITChannel** (A content distribution center. A channel is sort of like a mailing list. A list of subscriber entries is maintained, and new content can be published to that list. It's also a content access point that aggregates published content with a common subject or intended for a common audience.)

- **ITSubscriber** (A subscriber profile is used to control how published content is delivered to a user. Typical delivery mechanisms are mail, message queue or none. None is used by Portal users who will check the chann el for new archived packages and view them on demand, rather than having them delivered at publication time.)
    - **ITContentSubscriber** ( )
    - **ITEventSubscriber** ( )

- **ITFilter** (A filter string is used to include or exclude content going to a specific subscriber. It can be used to filter entries in a package, or the entire package.)

- **ITModel** (A queue prototype. Defines attributes for defining new queues.)

- **ITMap** (Maps the parameters in a message for architectural translation.)

- **ITRendModel** (Model for defining a Rendezvous queue.)

SAS 9.1 Open Metadata Interface: Reference

SAS Namespace Types

Types by Submodel

- **Analysis**
- **Authorization**
- **Business Information**
- **Foundation**
- **Grouping**
- **Mining**
- **Property**
- **Relational**
- **Resource**
- **Software Deployment**
- **Transform**
- **XML**

Analysis

AggregateAssociation
Aggregation
Cube
Dimension
Hierarchy
Level
Measure
OLAPProperty
OLAPSchema
SummaryStats

Authorization

AccessControl
AccessControlEntry
AccessControlTemplate
Permission
PermissionCondition
SecurityRule
SecurityRuleScheme
SecurityTypeContainmentRule

Business Information

Document
Email
Keyword

Location
Person
Phone
ResponsibleParty
Timestamp
UnitofTime

Foundation

AbstractExtension
Classifier
Extension
ExternalIdentity
Feature
Identity
IdentityGroup
NumericExtension
Role
Root

Grouping

Group
Tree

Mining

AnalyticColumn
AnalyticTable
FitStatistic
MiningResult
Target

Property

AbstractProperty
AssociationProperty
AttributeProperty
LocalizedResource
LocalizedType
Property
PropertyGroup
PropertySet
PropertyType

Prototype
PrototypeProperty

Relational

Column
ColumnRange
DatabaseCatalog
DatabaseSchema
DataSourceName
DataTable
ExternalTable
ForeignKey
Index
JoinTable
Key
KeyAssociation
LogicalColumn
PhysicalTable
QueryTable
RelationalTable
SASPassword
UniqueKey
WorkTable

Resource

ArchiveEntry
ArchiveFile
ContentLocation
ContentType
DeployedDataPackage
Device
DeviceType
Directory
File
Memory
RelationalSchema
Report
SASCatalog
SASCatalogEntry
SASLibrary
Stream
Text

TextStore

Software Deployment

AuthenticationDomain
COMConnection
Connection
DeployedComponent
LogicalServer
Login
Machine
NamedService
OpenClientConnection
SASClientConnection
SASLicense
ServerComponent
ServerContext
ServiceComponent
ServiceType
SoftwareComponent
TCPIPConnection

Transform

AbstractJob
AbstractTransformation
ClassifierMap
ConditionalPrecedence
Event
FeatureMap
GroupByClause
HavingClause
JFJob
Job
Join
OnClause
OrderByClause
QueryClause
RowSelector
Select
StepPrecedence
SyncStep
Transformation
TransformationActivity

TransformationStep
Variable
WhereClause

XML

SXLEMap
XPath

Previous Page | Next Page | Top of Page

REPOS Namespace Types

Overview

This section describes the metadata types that you use when you define repository objects. The SAS Open Metadata Interface provides these metadata types to enable you to register repositories in the repository manager, to define optional relationships between repositories, and to invoke features such as repository auditing. After you create repository objects, you can query their attributes and associations using IOMI class methods, just like you can those of any application−related metadata object. You can also issue IServer class methods to temporarily change a repository's state, for example, in preparation for a backup. For more information, see the documentation for the IOMI Class and IServer Class.

Defining relationships between repositories enables users of the repositories to create cross−repository references between the objects in those repositories. This capability is required for features such as the change management facility. The change management facility ensures multiuser concurrency by implementing a check−out/check−in mechanism for updating metadata in SAS metadata repositories. For more information about repository relationships and cross−repository references, see Creating Relationships Between Repositories in the **SAS 9.1 Open Metadata Interface: User's Guide**. For more information about the change management facility, see Using the Change Management Facility, also in the user's guide. For more information about repository auditing, see Invoking a Repository Audit Trail in the user's guide.

When using IOMI class methods, be sure to note usage considerations described in the metadata type documentation.

Type Hierarchy

The SAS Metadata Model uses classes and objects to define different types of metadata, and to model associations between individual metadata objects. It uses inheritance of attributes and associations to effect common behaviors, and it uses subclassing to extend behaviors.

The hierarchy of the repository types is

- Repository (Abstract Class for Metadata Repositories)
- RepositoryBase (Metadata Type for SAS Repositories)

Previous Page | Next Page | Top of Page

Repository

Abstract class for metadata repositories

Element tag: Repository

Overview

Repository is an abstract class for repositories that store metadata objects. The subtypes represent each repository engine that is supported. For SAS 9.1, the only subtype is RepositoryBase.

Attributes

AttributeName	Description	Type	Length
Desc	Description of object	String	200
Id	Object's repository identifier	String	15
MetadataCreated	Date and time metadata object was created	Double	
MetadataUpdated	Date and time metadata object was last updated	Double	
Name	Object's name	String	15

Usage

The following section includes some special usage notes for the Repository type.

Using GetMetadataObjects

When you use GetMetadataObjects to return a list of repositories, you can list the repositories that match any of the subtypes in the Repository superclass, or you can simply list the repositories that match one of the subtypes.

To list instances of all subtypes in the Repository superclass:

- specify the Repository supertype in the *Type* parameter
- set the OMI_INCLUDE_SUBTYPES (16) flag.

Example:

```
<GetMetadataObjects>
   <Reposid>A0000001.A0000001</Reposid>
   <Type>Repository</Type>
   <Objects/>
   <NS>REPOS</NS>
   <Flags>16</Flags>  /* OMI_INCLUDE_SUBTYPES flag */
   <Options/>
</GetMetadataObjects>
```

To list instances of one subtype in the Repository superclass, specify the appropriate repository subtype in the *Type* parameter.

Example:

```
<GetMetadataObjects>
   <Reposid>A0000001.A0000001</Reposid>
   <Type>RepositoryBase</Type>
   <Objects/>
   <NS>REPOS</NS>
   <Flags/>
   <Options/>
</GetMetadataObjects>
```

Previous Page | Next Page | Top of Page

RepositoryBase

Metadata type for SAS metadata repositories

Element tag:	RepositoryBase
Subclass of:	Repository

Overview

The RepositoryBase class models the metadata for SAS metadata repositories.

Attributes

Attribute Name	Description	Type	Length	Reqd for Add	Update Allowed
Id	The repository's object identifier.	String	17	No	No
Name	The unique object name.	String	60	Yes	Yes
Desc	A description of the repository.	String	200	No	Yes
Access	The repository object's access mode. The default value, CMR_FULL (0), indicates the repository is available for read, write, and update access. CMR_READONLY (1) indicates the repository can only be read.	Integer	1	No	Yes
Path	The directory where the repository is located.	String	200	Yes	No
Engine	The engine for this repository. Valid values are base (the default), ORACLE, or DB2.	String	10	No	Yes
Options	SAS library options for this repository.	String	200	No	No
RepositoryType	The type of repository. Valid values are Foundation, Project, or Custom.	String	40	No	Yes
PauseState	The Pause method override state. This attribute is set by	String	15	No	Yes

	the Pause method and cleared by the Resume method. Valid values are an empty string, READONLY, or OFFLINE. An empty string indicates the repository is not paused.				
AuditType	Turns auditing on and off and specifies the type of records that will be logged. An empty string indicates auditing has never been enabled; 1 enables auditing for added metadata records; 2 enables auditing for deleted metadata records; 4 enables auditing for updated metadata records; 7 enables auditing for all record types (add, delete, update); 0 turns off auditing and indicates it was disabled.	Integer	200	No	Yes
AuditPath	The directory where the audit trail is to be created.	String	200	No	Yes
AuditEngine	The engine for the audit trail. This option is reserved for future use.	String	10	No	Yes
AuditOptions	SAS library options for the audit trail. This option is reserved for future use.	String	200	No	No
MetadataCreated	Date and time the metadata object was created.	Double		No	No
MetadataUpdated	Date and time the metadata object was last updated.	Double		No	No

Association elements

Name, Subelements, and Partner Name	Description	Number of Subelements	Reqd for Add	Obj Ref	Update Allowed
Name: DependencyUsedBy Subelements: Repository	The repositories that depend on this repository.	0 to *	No	No	Yes

Partner Name: RepositoryBase					
Name: DependencyUses Subelements: Repository Partner Name: RepositoryBase	The repositories on which this repository depends.	0 to *	No	No	Yes

Usage

The following section includes usage notes for the RepositoryBase type.

All Methods

A method call that adds, updates, or deletes a RepositoryBase object instance or any of its properties must specify REPOS in the *Namespace* parameter. It must also specify the OMI_TRUSTED_CLIENT flag (268435456) in the *Flags* parameter.

The metadata server must add, update, and delete objects in the REPOS namespace when no other activity is taking place in the server, so it automatically delays other client requests until the REPOS namespace changes are complete. This may have a small effect on server performance.

Using AddMetadata

For remote repositories, specify the Engine and Options attributes.

To make a repository read−only, set Access = 1. To grant specific users ReadMetadata and WriteMetadata permission to a repository or to specific metadata objects in the repository, use the SAS Open Metadata Architecture authorization facility. For more information about the authorization facility, see "Understanding the Authorization Facility" in the **SAS 9.1 Metadata Server: Setup Guide**.

Using DeleteMetadata

You can delete the metadata in a repository or the metadata and the repository's registration. For more information, see DeleteMetadata. Also see Clearing or Deleting a Repository in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Using Pause

The Pause method temporarily overrides the access mode in a repository's Access attribute by specifying a value for the PauseState attribute. The PauseState remains in effect until client activity on the repository is resumed using the Resume method, or until the server is reinitialized. The Resume method returns the repository to the access mode specified in the Access attribute. PauseState is a transient value that is not retained across instances of the server. Use the GetRepositories method to retrieve a repository's Access and PauseState attribute values. For more information, see Pause and Resume.

OMI (IOMI) Class

Overview

The SAS Open Metadata Interface defines a set of methods that read and write metadata types (the IOMI class) and a set of methods for administering SAS metadata repositories and the SAS Metadata Server (the IServer class). This section describes the methods in the IOMI class.

The IOMI class is surfaced as a Component Object Model (COM) class in the Windows development environment and as a Java class in the Java programming environment. Besides the initial invocation, the methods are identical in all programming environments.

Using the IOMI Class

Using the IOMI class is a matter of using its methods. To access these methods, you must connect to the SAS Metadata Server and instantiate objects as described in "Introduction to IOMI Methods."

Introduction to IOMI Methods

Each IOMI method has a set of parameters that communicate the details of the metadata request to the SAS Metadata Server. For example, parameters are used to identify the namespace to use as the context for the request, the repository in which to process the request, and the metadata type to reference. In addition, parameters are provided for specifying flags and additional options to use when processing the request.

Methods that read and write metadata objects additionally require you to pass a metadata property string that describes the object to the server. This metadata property string must be formatted in XML (Extensible Markup Language). For instructions about how to define a metadata property string, see Constructing a Metadata Property String.

Each IOMI method also has two output parameters: a return code and a holder for information received from the server.

Return Code

The return code is a boolean operator that indicates whether the method communicated with the server. A 0 indicates that communication was established. A 1 indicates that communication did not occur. The return code does not indicate the success or failure of the method call itself. It is the responsibility of SAS Open Metadata Interface clients to provide error codes.

Other Method Output

All other output received from the server is in the form of a formatted XML string. The output typically mirrors the input, except that the requested values are filled in.

Constructing a Metadata Property String

To read or write a metadata object, you must pass a string of properties that describe that object to the SAS Metadata Server. This property string is passed to the server in the *inMetadata* parameter of the method call.

A metadata object is described by

- its metadata type
- attributes that are specific to the metadata object, such as its ID, name, description, and other characteristics
- its associations with other metadata objects.

The SAS Open Metadata Interface supports the following XML elements for defining a metadata property string:

Metadata type
identifies the metadata type that you want to read or write, within angle brackets. This example shows the XML element representing the PhysicalTable metadata type.

```
<PhysicalTable></PhysicalTable>
```

A shorthand method of specifying this tag set is

```
<PhysicalTable/>
```

Metadata type attributes
specify attributes of the metadata type as XML attributes (within the angle brackets of the metadata type). This example specifies the PhysicalTable metadata type that has "NE Sales" in the Name attribute.

```
<PhysicalTable Name="NE Sales"/>
```

Metadata type association name and association subelement elements
describe the relationship between the metadata type named in the main XML element and another metadata type as nested XML elements.

```
<PhysicalTable Name="NE Sales"/>
   <Columns>
      <Column/>
   </Columns>
</PhysicalTable>
```

The first nested element, Columns, is the **association name**. The association name is a label that describes the relationship between the main element and the subelement. The second nested element, Column, is the **association subelement**. The association subelement identifies the partner metadata object in the relationship. The nested elements in the example specify that the main metadata object, PhysicalTable, has a Columns association to an object of metadata type Column.

CAUTION:

In order to meet XML parsing rules, the metadata type, attribute, and association element and subelement names that you specify in the metadata property string **must** exactly match those published in the metadata type documentation.

Quotation Marks and Special Characters

The metadata property string is passed as a string literal (a quoted string) in most programming environments. To ensure that the string is parsed correctly, it is recommended that any additional double−quotation marks, such as those used to denote XML attribute values in the metadata property string, be marked to indicate that they should be treated as characters. Below are examples of using escape characters to accomplish the task:

Java

"<PhysicalTable Id=\"123\" Name=\"TestTable\" />"

Visual Basic

"<PhysicalTable Id=""123"" Name=""TestTable"" />"

Visual C++

"<PhysicalTable Id=\"123\" Name=\"TestTable\" />"

SAS

"<PhysicalTable Id=""123"" Name=""TestTable"" />"

'<PhysicalTable Id="123" Name="TestTable" />'

Special characters that are used in XML syntax are specified as:

'<' = <

'>' = >

'&' = &

Previous Page | Next Page | Top of Page

Identifying Metadata

The documentation for many of the metadata types refers to "general, identifying information." This phrase refers to the Id and Name attributes.

Each metadata object in a repository, such as the metadata for a particular column in a SAS table, has a unique identifier assigned to it when the object is created. Each object also has a name. For example, here is the Id and Name for a SAS table column, as returned by the GetMetadata method.

```
<Column Id="A2345678.A3000001" Name="New Column"/>
```

Name
: refers to the name of the metadata object. In the previous example, the Name of the table column is "New Column".

Id
: refers to the unique identifier for a metadata object. It has the form *reposid.instanceid*. For example, in the previous example, the Id for the Column object is A2345678.A3000001.

The *reposid* is assigned to a particular metadata repository when it is created. A *reposid* is a unique character string that identifies the metadata repository that stores the object. Each application using the SAS Open Metadata Architecture will have one or more repositories.

The *instanceid* is assigned to a particular object when it is created. An *instanceid* is a unique character string that distinguishes one metadata object from all other objects of the same type.

Note: No assumptions should be made about the internal format of the Id attribute. Different repository systems use different Id formats.

CAUTION:
: **It is strongly recommended that you avoid coding the identifier of a particular metadata object in a client application.** Instead, use the GetMetadataObjects method or other SAS Open Metadata Interface method to return a list of the unique object identifiers, names, and descriptions for objects of a particular type.

Previous Page | Next Page | Top of Page

Functional Index to IOMI Methods

In the dictionary, IOMI methods are listed in alphabetical order. This topic lists IOMI methods by category.

Functional Index to IOMI Methods

Category	Method	Description
Change Management Methods	CheckinMetadata	Copies metadata objects from a project repository to a primary repository and unlocks them
	CheckoutMetadata	Locks and copies metadata objects from a primary repository to a project repository
	FetchMetadata	Copies metadata objects from a primary repository to a project repository without locking them
	UndoCheckoutMetadata	Deletes the specified object from the project repository and unlocks its corresponding primary object
Management Methods	GetNamespaces	Lists the namespaces defined for a repository
	GetSubtypes	Returns all possible subtypes for a specified metadata type
	GetTypes	Lists all metadata types defined in the SAS Metadata Model
	GetTypeProperties	Returns all possible properties for a metadata type
	IsSubtypeOf	Determines whether one metadata type is a subtype of another
Read Methods	GetMetadata	Reads specified metadata from a repository
	GetMetadataObjects	Lists metadata objects when passed the repository and type
Repository Methods	GetRepositories	Lists all metadata repositories available through this server
Write Methods	AddMetadata	Adds specified metadata to a repository
	DeleteMetadata	Deletes specified metadata from a repository
	UpdateMetadata	Updates specified metadata in a repository
Messaging Method	DoRequest	Executes XML−encoded method calls

SAS 9.1 Open Metadata Interface: Reference

Using IOMI Flags

Various IOMI methods support flags. The write methods (AddMetadata, DeleteMetadata, and UpdateMetadata) require that an OMI_TRUSTED_CLIENT flag be set to authenticate write operations. Other methods support flags to expand or to filter metadata retrieval requests. See Summary Table of IOMI Flags for a list of the available flags and the methods for which they are supported.

Specifying a Flag

IOMI flags are specified as numeric constants in the *Flags* parameter of a method call. For example, to specify the OMI_ALL flag in a GetMetadata call, supply the number 1 in the *Flags* parameter. To specify more than one flag, add their numeric values together and supply the sum in the *Flags* parameter. For example, OMI_ALL (1) + OMI_SUCCINCT (2048)=2049. This flag combination will retrieve all properties for the named object, excluding those for which a value has not been defined.

Corresponding XML Elements

Most of the flags do not require additional input, but those that do supply it in a special XML element in the *Options* parameter. For example, the OMI_XMLSELECT flag, which invokes search criteria to filter the objects retrieved by the GetMetadataObjects method, uses an <XMLSelect> element to pass a search string to the server. The GetMetadata method OMI_TEMPLATES flag is used in conjunction with a <Templates> element to pass additional metadata property strings to the server. Several methods also support an OMI_LOCK_TEMPLATES flag that, when used in conjunction with a <LockTemplates> element, enables you to specify associated objects to include in the operation. See Summary Table of IOMI Options for a list of these special XML elements.

Flag Behavior When Multiple Flags are Used

Some methods, like GetMetadata and GetMetadataObjects, support many flags. GetMetadata flags can also be used in a GetMetadataObjects call when the OMI_GET_METADATA flag is set. When more than one flag is set, each is applied unless a filtering option is also used. For example, GetMetadata flags specified in a GetMetadataObjects request will retrieve properties only for objects retrieved after any <XMLSelect> criteria have been applied. When search criteria are specified in the *inMetadata* parameter of a GetMetadata call to filter the associated objects that are retrieved and OMI_ALL is set, GetMetadata retrieves information only about associated objects that meet the search criteria.

When a template is passed, the properties and any search criteria specified in the template are applied *in addition to* any properties requested by other GetMetadata parameters.

Previous Page | Next Page | Top of Page

Summary Table of IOMI Flags

The following table summarizes the flags that are supported for the metadata−related methods.

Flags Supported in IOMI Methods

Flag name	Numeric Indicator	Method	Description
OMI_ALL	1	GetMetadata GetRepositories	Gets all of the properties of the requested object and general, identifying information about any objects that are associated with the requested object.
OMI_ALL_DESCENDANTS	64	GetSubtypes	Gets the descendants of the returned subtypes in addition to the subtypes.
OMI_ALL_SIMPLE	8	GetMetadata	Gets all of the attributes of the requested object.
OMI_CI_DELETE_ALL	33	CheckinMetadata	Deletes fetched as well as checked and new metadata objects from a project repository upon check−in to a primary repository.
OMI_CI_NODELETE	524288	CheckinMetadata	Keeps copies of checked, fetched, and new metadata objects in a project repository after check−in to a primary repository.
OMI_DELETE	32	DeleteMetadata	Deletes the contents of a repository in addition to the repository's registration.
OMI_DEPENDENCY_USED_BY	16384	GetMetadata GetMetadataObjects	In GetMetadata, specifies to look for cross−repository references in repositories that use the current repository. In GetMetadataObjects, specifies to retrieve objects of the passed type from all repositories that have a DependencyUsedBy dependency in the repository chain. For more information about repository dependencies and cross−repository references, see Creating Relationships Between Repositories in the **SAS 9.1 Open Metadata Interface: User's Guide**.
OMI_DEPENDENCY_USES	8192	GetMetadataObjects	Specifies to retrieve objects of the passed type from all repositories that have a DependencyUses

			dependency in the repository chain. For more information about repository dependencies, see Creating Relationships Between Repositories in the **SAS 9.1 Open Metadata Interface: User's Guide**.
OMI_GET_METADATA	256	GetMetadataObjects	Executes a GetMetadata call for each object that is returned by the GetMetadataObjects method.
OMI_IGNORE_NOTFOUND	134217728	DeleteMetadata UpdateMetadata	Prevents a delete or update operation from being aborted when a request specifies to delete or update an object that does not exist.
OMI_INCLUDE_SUBTYPES	16	GetMetadata GetMetadataObjects	Gets the properties of metadata objects that are subtypes of the passed metadata type in addition to the metadata for the passed type. In GetMetadata, this flag must be used in conjunction with the OMI_TEMPLATES flag.
OMI_LOCK	32768	GetMetadata	Locks the specified object and any associated objects selected by GetMetadata flags and options from update by other users.
OMI_LOCK_TEMPLATE	65536	CheckoutMetadata CopyMetadata FetchMetadata GetMetadata UndoCheckoutMetadata	In GetMetadata, specifies to lock the associated objects specified in a user−defined lock template instead of associated objects requested by the method. In the other methods, overrides the default OMA lock template with a user−supplied template. In all cases, the user−defined lock template is submitted in a <LockTemplates> element in the *Options* parameter.
OMI_MATCH_CASE	512	GetMetadataObjects	Performs a case−sensitive search based on criteria specified in the <XMLSelect> option. The OMI_MATCH_CASE flag must be used in conjunction with the OMI_XMLSELECT flag.
OMI_NO_DEPENDENCY_CHAIN	16777216	GetMetadataObjects	When used with OMI_DEPENDENCY_USES, retrieves objects only from repositories on which the repository directly depends. When used with

			OMI_DEPENDENCY_USED_BY, retrieves objects only from repositories that directly depend upon the repository.
OMI_NOFORMAT	67108864	GetMetadata	Causes date, time, and datetime values in the output XML stream to be returned as raw SAS Date, SAS Time, and SAS Datetime floating point values. Without the flag, the default US−English locale is used to format the values into recognizable character strings.
OMI_PURGE	1048576	DeleteMetadata	Removes previously deleted metadata from a repository without disturbing the current metadata objects.
OMI_REINIT	2097152	DeleteMetadata	Deletes the contents of a repository but does not remove the repository's registration from the repository manager.
OMI_RETURN_LIST	1024	CheckinMetadata CheckoutMetadata CopyMetadata DeleteMetadata FetchMetadata UndoCheckoutMetadata UpdateMetadata	In the change management methods, returns the object identifiers of any associated objects that were copied in the *outMetadata* parameter. In DeleteMetadata, returns the identifiers of any cascading objects that were deleted in the *outMetadata* parameter. In UpdateMetadata, returns the identifiers of any dependent objects that were deleted as a result of the update operation in the *outMetadata* parameter.
OMI_SUCCINCT	2048	GetMetadata GetTypes	In GetMetadata, omits all properties that do not contain values or that contain a null value. In GetTypes, checks the *Options* parameter for a <Reposid> element, and lists the metadata types for which objects exist in the specified repository. For more information, see Listing the Metadata Types in a Repository in the **SAS 9.1 Open Metadata Interface: User's Guide**.
OMI_TEMPLATE	4	GetMetadata	Checks the *Options* parameter for one or more user−defined templates that define which metadata properties to return. The templates

			are passed as a metadata property string in a <Templates> element. For more information, see Using Templates in the **SAS 9.1 Open Metadata Interface: User's Guide**.
OMI_TRUNCATE	4194304	DeleteMetadata	Deletes all metadata objects but does not remove the metadata object containers from a repository or change the repository's registration.
OMI_TRUSTED_CLIENT	268435456	AddMetadata UpdateMetadata DeleteMetadata	Determines whether the client can call this method.
OMI_UNLOCK	131072	UpdateMetadata	Unlocks an object lock that is held by the calling user.
OMI_UNLOCK_FORCE	262144	UpdateMetadata	Unlocks an object lock that is held by another user.
OMI_XMLSELECT	128	GetMetadataObjects	Checks the *Options* parameter for search criteria that qualifies the objects to return. The criteria are passed as a search string in an <XMLSelect> element. For more information, see Filtering a GetMetadataObjects Request in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Previous Page | Next Page | Top of Page

Summary Table of IOMI Options

The following table lists the optional XML elements that are used in conjunction with IOMI flags.

Options Supported for IOMI Methods

Option	Flag	Method	Description
<DOAS>	None	AddMetadata DeleteMetadata GetMetadata GetMetadataObjects GetSubTypes GetTypeProperties IsSubtypeOf UpdateMetadata	Specifies an alternate calling identity for a metadata request. For more information, see <DOAS> Option.
<LockTemplates>	OMI_LOCK_TEMPLATE (65536)	CheckoutMetadata FetchMetadata CopyMetadata GetMetadata UndoCheckoutMetadata	For GetMetadata, specifies associated objects to be locked instead of those requested by method flags and options. For other methods, overrides the default OMA lock template with a user−supplied template.
<Reposid>	OMI_SUCCINCT (2048)	GetTypes	Specifies a repository ID for which to list metadata types. See Listing the Types in a Repository in the **SAS 9.1 Open Metadata Interface: User's Guide**.
<Templates>	OMI_TEMPLATE (4)	GetMetadata	Specifies properties to retrieve for an object, in addition to those specified in the inMetadata parameter. See Using Templates in the **SAS 9.1 Open Metadata Interface: User's Guide**.
<XMLSelect>	OMI_XMLSELECT (128) and optionally OMI_MATCH_CASE (512)	GetMetadataObjects	Specifies a search string to filter the objects that are retrieved. See Filtering a GetMetadataObjects Request in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Previous Page | Next Page | Top of Page

<DOAS> Option

IOMI class methods support a <DOAS> option that enables SAS Open Metadata Interface clients to make a metadata request on behalf of another user. Typically when a metadata request is made, the authorization facility checks the user ID and credentials of the connecting user to determine whether the request is allowed. The <DOAS> option causes the request to be issued on behalf of another user ID and authorized using the credentials of this other user.

Credentials refer to the set of metadata identities associated with a user who is registered in the SAS Metadata Server. The set begins with a principal identity represented by the Person (or IdentityGroup) object that is mapped directly to an authenticated user ID. The credentials set also contains references to any IdentityGroup objects in which the principal identity is either directly or indirectly identified as a member.

The <DOAS> option is provided to enable middleware servers to assert the identity of their own clients during requests for metadata. This way, the request is authorized based on the credentials of the client rather than those of the connecting server. That is, when <DOAS> is found, metadata is created, returned, and updated based on the credentials of the specified client rather than those of the connecting server. It is the responsibility of the connecting server to authenticate the client.

Specifying the <DOAS> Option

The <DOAS> option is supported in the AddMetadata, DeleteMetadata, GetMetadata, GetMetadataObjects, GetSubTypes, GetTypeProperties, IsSubtypeOf, and UpdateMetadata methods.

It is passed in the *Options* parameter in the form

```
<DOAS Credential="credHandle"/>
```

where *credHandle* is a handle returned by the ISecurity getCredentials method that represents the surrogate user's credentials. **A user must have trusted user status in order to issue the ISecurity getCredentials method.** The getCredentials method queries the metadata server to build a set of credentials for an authenticated user ID and returns a handle based on the metadata identities found. A trusted user is a special user whose user ID is defined in the trustedUsers.txt file. For more information about the support provided for middleware servers, see "Trusted Users" in "Understanding the Authorization Facility" in the **SAS 9.1 Metadata Server: Setup Guide**.

Example 1: Standard Interface

The following is an example of a GetMetadataObjects method that specifies the <DOAS> option. The method call is formatted for the standard interface.

```
<!-- set repository Id and type -->
reposid="A0000001.A4345678";
type="PhysicalTable";
ns="SAS";
flags=0;
options="<DOAS Credential="0000000000235462"/>";

rc = GetMetadataObjects(reposid, type, objects, ns, flags, options);
```

This request will return only PhysicalTable objects which the user identified in the credential handle is authorized to read.

Example 2: DoRequest Method

The following is an example of an AddMetadata method that specifies the <DOAS> option. The method call is formatted for the *inMetadata*= parameter of DoRequest method.

```
<AddMetadata>
   <Metadata>
     <PhysicalTable Name="NECust" Desc="All customers in the northeast region"/>
   </Metadata>
   <Reposid>A0000001.A2345678</Reposid>
   <NS>SAS</NS>
   <Flags>268435456</Flags>
   <Options>
    <DOAS Credential="0000000000235462"/>
   </Options>
</AddMetadata>
```

The requested object will be created only if the user identified in the credential handle has WriteMetadata permission to the specified repository.

Previous Page | Next Page | Top of Page

AddMetadata

Adds new metadata objects to a repository

Category: Write methods

Syntax

rc = AddMetadata(inMetadata, reposid, outMetadata, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Passed metadata property string defining the properties to be added for an object.
reposid	C	in	Passed repository ID of the repository to which the metadata is to be added.
outMetadata	C	out	Returned metadata properties expanded from the addition of the object.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_TRUSTED_CLIENT=268435456*** Determines if the client can call this method. This flag is currently required.
options	C	in	Passed indicator for options. ***<DOAS Credential=****"credHandle"****/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option.

Details

The AddMetadata method is used to create new metadata objects in a repository. To update an existing metadata object, use the UpdateMetadata method.

The *inMetadata* parameter specifies an XML string that defines the properties to be added for the object. Not all metadata types or their properties can be added; refer to the documentation for each metadata type. AddMetadata returns an error for any type that cannot be added.

The *outMetadata* parameter mirrors the content of the *inMetadata* parameter and additionally returns identifiers for the requested objects. Any invalid properties in the *inMetadata* string remain in the *outMetadata* string. See Constructing a Metadata Property String for information about the structure of the metadata property string.

The AddMetadata method can be used to create an object, to create an object and an association to an existing object, or to create an object, an association, and the associated object. Associations can be made to objects in the same or another repository. The attributes used to identify the objects indicate the type of operation that is to be performed. For more information, see Adding Metadata in the **SAS 9.1 Open Metadata Interface: User's Guide**.

The metadata server assigns object identifiers at the successful completion of an AddMetadata call.

Be sure to check the return code of a write method call. A nonzero return code indicates that a failure occurred while trying to write the metadata. When a nonzero return code is returned, none of the changes indicated by the method call are made.

The following examples show how to issue a simple AddMetadata method call without regard to the programming environment. For examples, see Program–Specific AddMetadata Examples.

Example 1: Standard Interface

The following AddMetadata request adds a new PhysicalTable object and fills in its Name and Desc properties.

```
<!-- Create a metadata list to be passed to AddMetadata routine -->
inMetadata = "<PhysicalTable Name="NECust"
               Desc="All customers in the northeast region"/>";
reposid= "A0000001.A2345678";
ns= "SAS";
flags= 268435456;  <!-- OMI_TRUSTED_CLIENT flag -->
options= "";

rc = AddMetadata(inMetadata, reposid, outMetadata, ns, flags, options);

<!-- outMetadata XML string returned -->
<PhysicalTable Id="A2345678.A2000001" Name="NECust"
  Desc="All customers in the northeast region"/>
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method. Note that <Metadata> tags, rather than <inMetadata> tags, identify the passed property string.

```
<AddMetadata>
   <Metadata>
     <PhysicalTable Name="NECust" Desc="All customers in the northeast region"/>
   </Metadata>
   <Reposid>A0000001.A2345678</Reposid>
   <NS>SAS</NS>
   <Flags>268435456</Flags>
   <Options/>
</AddMetadata>
```

Related Methods

- CopyMetadata
- UpdateMetadata
- GetRepositories

Previous Page | Next Page | Top of Page

CheckinMetadata

Copies metadata objects from a project repository to a primary repository and unlocks them

Category: Change management methods

Syntax

rc = CheckinMetadata(inMetadata, outMetadata, projectReposid, changeName, changeDesc, changeId, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata string identifying objects to check in. Reserved for future use.
outMetadata	C	out	Metadata string returned from the server.
projectReposid	C	in	Passed identifier of the source project repository.
changeName	C	in	Name for generated Change object.
changeDesc	C	in	Description for generated Change object.
changeId	C	in	Identifier of existing Change object. Reserved for future use.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_CI_DELETE_ALL=33*** Deletes fetched as well as checked and new metadata objects from a project repository upon check–in to a primary repository. ***OMI_CI_NODELETE=524288*** Keeps copies of checked, fetched, and new metadata objects in a project repository after

			check–in to a primary repository. ***OMI_RETURN_LIST=1024*** Includes the object identifiers of any associated objects that were copied to a primary repository in the *outMetadata* parameter.
options	C	in	Reserved for future use.

Details

The CheckinMetadata method is used with the CheckoutMetadata method to provide an organized method of controlling updates to metadata objects by multiple users. CheckinMetadata copies to and unlocks in the appropriate primary repository metadata objects that were locked and copied to the specified project repository by CheckoutMetadata.

Note: For 9.1, CheckinMetadata does not support check–in of individual objects. A CheckinMetadata request copies all of the objects in the specified project repository to their respective primary repositories.

The source project identifier must include both the repository manager ID and the repository ID in the form *Reposmgrid.Reposid*.

The *changeName* and *changeDesc* parameters allow you to supply a description of object modifications for reporting purposes. These values are stored as the Name= and Desc= attributes of a Change object that is generated by the check–in process.

The default behavior of the CheckinMetadata method is to delete checked and new metadata objects from the project repository upon check–in. Fetched objects are not deleted. Set OMI_CI_DELETE_ALL to additionally delete fetched objects. Set the OMI_CI_NODELETE flag to keep copies of the objects in the project repository.

For more information about primary repositories, project repositories, and Change objects, see Using the Change Management Facility in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Example 1: Standard Interface

The following CheckinMetadata request copies all the metadata objects in project repository A0000001.A5WW3LXC into their respective primary repositories and unlocks them.

```
projectReposid="A0000001.A5WW3LXC";
changeName="Test1";
changeDesc="Changes for CM testing";
ns= "SAS";
flags="0";
options= "";

rc = CheckinMetadata(projectReposid, changeName, changeDesc, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<CheckinMetadata>
   <ProjectReposid>A0000001.A5WW3LXC</ProjectReposid>
   <ChangeName>CM Testing</ChangeName>
   <ChangeDesc>Changes for CM testing</ChangeDesc>
   <Ns>SAS</Ns>
   <Flags>0</Flags>
   <Options/>
</CheckinMetadata>
```

Related Methods

- CheckoutMetadata
- UndoCheckoutMetadata
- FetchMetadata

Previous Page | Next Page | Top of Page

CheckoutMetadata

Locks and copies metadata objects from a primary repository to a project repository

Category: Change management methods

Syntax

rc = CheckoutMetadata(inMetadata, outMetadata, projectReposid, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata string identifying an object to be locked and copied.
outMetadata	C	out	Metadata string returned by the server.
projectReposid	C	in	Passed object ID of the target project repository.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_LOCK_TEMPLATE=65536*** Overrides the default lock template with a user−defined template submitted in the *Options* parameter. ***OMI_RETURN_LIST=1024*** Includes the object identifiers of any associated objects that were checked out in the *outMetadata* parameter.
options	C	in	An XML element that specifies method options. Currently, the only supported option is: ***<LockTemplates>*** Specifies a user−defined lock template. The <LockTemplates> element must be specified in

			conjunction with the OMI_LOCK_TEMPLATE flag.

Details

The CheckoutMetadata method provides an organized method of controlling updates to metadata objects by multiple users. The method locks and copies the specified metadata object from its primary repository to a project repository where it can be safely updated. Any associated objects identified in a default lock template are automatically locked and copied as well.

The metadata object to copy is identified in the *inMetadata* parameter by its metadata type and its object instance identifier. The object instance identifier is specified in the form *Reposid.Objectid*. The target project repository is identified in the *projectReposid* parameter and is specified in the form *Reposmgrid.Reposid*. Multiple metadata objects can be checked out at the same time by stacking their metadata property strings in the *inMetadata* parameter.

When a metadata object is checked out, any associated objects defined for it in a default lock template are checked out as well. To view the default lock template defined for a metadata type, see Default Lock Templates in the **SAS 9.1 Open Metadata Interface: User's Guide**. To override the default lock template and check out a different set of associated objects, set the OMI_LOCK_TEMPLATE (65536) flag and supply an alternate template in a <LockTemplates> element in the *Options* parameter. For more information about lock templates, see Lock Templates in Using the Change Management Facility in the user's guide.

By default, the *outMetadata* parameter lists only the object(s) specified in the *inMetadata* parameter. To include associated objects that were checked out by the lock template, set the OMI_RETURN_LIST (1024) flag.

Example 1: Standard Interface

The following CheckoutMetadata request locks a PhysicalTable object in primary repository A5WW3LXC and copies it to project repository A5NZA58I.

```
inMetadata="<PhysicalTable ID="A5WW3LXC.AA000002"/>";
projectReposid="A0000001.A5NZA58I";
outMetadata="";
ns= "SAS";
flags=0;
options= "";

rc = CheckoutMetadata(inMetadata, outMetadata, projectReposid, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->
<CheckoutMetadata>
   <Metadata>
      <PhysicalTable ID="A5WW3LXC.AA000002"/>
   </Metadata>
```

```
   <ProjectReposid>A0000001.A5NZA58I</ProjectReposid>
   <ns>SAS</ns>
   <flags>0</flags>
   <Options/>
</CheckoutMetadata>
```

Related Methods

- UndoCheckoutMetadata
- CheckinMetadata
- FetchMetadata

Previous Page | Next Page | Top of Page

CopyMetadata

Copies metadata objects from one metadata repository to another

Category: Change management methods

Syntax

rc = CopyMetadata(inMetadata, outMetadata, targetReposid, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata objects to be copied.
outMetadata	C	out	Returned metadata string that contains the results of the copy operation.
targetReposid	C	in	Passed object ID of the target repository.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_LOCK_TEMPLATE=65536*** Overrides the default lock template with a user−supplied template submitted in the <LockTemplates> element in the *Options* parameter. ***OMI_RETURN_LIST=1024*** Returns the object identifiers of any associated objects that were copied in the *outMetadata* parameter.
options	C	in	An XML element that specifies method options. Currently, the only supported option is: ***<LockTemplates>***

			Specifies associated objects to copy. The <LockTemplates> element must be specified in conjunction with the OMI_LOCK_TEMPLATE flag.

Details

CopyMetadata enables you to copy individual metadata objects in a repository to the same or another repository. To copy or promote a repository, use SAS Management Console. It is categorized as a change management method, but it can be used independently of the change management facility.

When a metadata object is copied, any associated objects defined for it in a default lock template are copied as well. To view the default lock template defined for a metadata type, see Default Lock Templates in the **SAS 9.1 Open Metadata Interface: User's Guide**. To override the default lock template and copy a different set of associated objects, set the OMI_LOCK_TEMPLATE (65536) flag and supply an alternate lock template in a <LockTemplates> element in the *Options* parameter. For more information about lock templates, see Lock Templates in Using the Change Management Facility, also in the user's guide.

Set the OMI_RETURN_LIST (1024) flag to write the associated objects that were copied to the *outMetadata* parameter.

Copied objects are considered new objects and are assigned unique identifiers. Associations between copied objects are updated to reference each other instead of the original objects. If a copied object has an association to a non−copied object and the non−copied object exists in a repository other than the target repository, the association will be broken unless a dependency exists between the repositories and a cross−repository reference can be created between the copied and non−copied objects.

Example 1: Standard Interface

The following CopyMetadata request duplicates a PhysicalTable object in the same repository.

```
inMetadata="<PhysicalTable Id="A5WW3LXC.AA000002"/>";
ToReposid="A0000001.A5WW3LXC";
outMetadata="";
ns= "SAS";
flags=1024;
options= "";

rc = CopyMetadata(inMetadata, outMetadata, ToReposid, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<CopyMetadata>
   <Metadata>
      <PhysicalTable Id="A5WW3LXC.AA000002"/>
   </Metadata>
   <!--Metadata and ToReposid parameters specify the same repository-->
   <ToReposid>A0000001.A5WW3LXC</ToReposid>
```

```
   <Ns>SAS</Ns>
   <!--OMI_RETURN_LIST flag -->
   <Flags>1024</Flags>
   <Options/>
</CopyMetadata>
```

Related Methods

- AddMetadata

Previous Page | Next Page | Top of Page

DeleteMetadata

Deletes metadata objects from a repository

Category: Write Methods

Syntax

rc = DeleteMetadata(inMetadata, outMetadata, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata properties to be deleted for a specific object.
outMetadata	C	out	Returned metadata string that contains the results of the delete operation. The outMetadata parameter is returned only if OMI_RETURN_LIST is specified.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_DELETE=32*** Deletes the contents of a repository in addition to the repository's registration. ***OMI_IGNORE_NOTFOUND=134217728*** Prevents a delete operation from being aborted when a request specifies to delete an object that does not exist. ***OMI_PURGE=1048576*** Removes previously deleted metadata from a repository without disturbing the current metadata objects. ***OMI_REINIT=2097152*** Deletes the contents of a repository but does not

			remove the repository's registration from the repository manager. ***OMI_RETURN_LIST=1024*** Returns a list of deleted object IDs, as well as any cascading object IDs that were deleted. ***OMI_TRUNCATE=4194304*** Deletes all metadata objects but does not remove the metadata object containers from a repository or change a repository's registration. ***OMI_TRUSTED_CLIENT=268435456*** Determines if the client can call this method. This flag is required.
options	C	in	Passed indicator for options. ***<DOAS Credential="credHandle"/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option.

Details

The DeleteMetadata method handles requests to remove objects from a repository. To delete specific properties of a metadata object, use the UpdateMetadata method.

Besides deleting specific metadata objects, you can use the DeleteMetadata method to delete all of the metadata objects in a repository, to unregister a repository, and to destroy a repository. You unregister a repository by issuing the DeleteMetadata method on the RepositoryBase object in the REPOS namespace without flags. To delete objects without unregistering a repository, or to destroy a repository, you must specify a flag. Flags that operate on the repository level (OMI_DELETE, OMI_PURGE, OMI_REINIT, and OMI_TRUNCATE) are supported only in the REPOS namespace on the RepositoryBase type.

Caution: You should not combine the REPOS namespace flags. Each one should be run exclusively. Combining flags will potentially yield undesirable results. For additional information about using REPOS namespace flags, see Clearing or Deleting a Repository in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Delete requests for individual metadata objects in the SAS namespace are handled as follows: When a delete is requested on an object that has dependent objects, the dependent objects are automatically deleted as well. There is no need to specify the dependent objects in the deletion request. This is called a cascading delete. If dependent objects are specified, the metadata server attempts to locate objects that have already been deleted, and aborts the delete operation when they are not found. You can prevent this from happening by setting the OMI_IGNORE_NOTFOUND flag, but it is recommended that you avoid specifying dependent objects.

Dependent objects that exist in other repositories are deleted if the delete request is issued on the repository that has a DependencyUses association in the repository relationship. The DeleteMetadata method will not

delete dependent objects when the delete request is issued on the repository that has a DependencyUsedBy association. To learn more about repository dependencies, see Creating Relationships Between Repositories in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Be sure to check the return code of a delete method call. A non−zero return code indicates a failure in the method call. When a non−zero return code is returned, none of the changes indicated by the method call will be made.

For additional information, including usage examples, see Deleting Metadata Objects, also in the user's guide.

The following examples show how to issue a DeleteMetadata method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For examples, see Program−Specific DeleteMetadata Examples.

Example 1: Standard Interface

The following DeleteMetadata request deletes a SASLibrary object. When a SASLibrary object is deleted, any object in the library will be deleted as well. The OMI_RETURN_LIST flag is specified (268435456 + 1024 =268436480) so the *outMetadata* parameter will return a list of all deleted object IDs.

```
inMetadata="<SASLibrary Id='A2345678.A2000001'/>";
outMetadata="";
ns= "SAS";
flags= 268436480;
options= "";

rc = DeleteMetadata(inMetadata, outMetadata, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<DeleteMetadata>
  <Metadata>
    <SASLibrary Id="A2345678.A2000001"/>
  </Metadata>
  <NS>SAS</NS>
  <Flags>268436480</Flags>
  <Options/>
</DeleteMetadata>
```

Related Methods

- AddMetadata
- UndoCheckoutMetadata
- UpdateMetadata

SAS 9.1 Open Metadata Interface: Reference

DoRequest

Executes XML encoded method calls

Category: Message Methods

Syntax

rc = DoRequest(inMetadata, outMetadata);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Passed metadata string that contains the metadata method to execute and any parameters appropriate to that method. For more information about the format of this input string, see the documentation for the method that you want to execute.
outMetadata	C	out	Returned metadata string that contains the results of the executed method. For more information about the format of this string, see the documentation for the method that you executed.

Details

The DoRequest method enables you to submit other IOMI methods and all of their parameters to the SAS Metadata Server in an input XML string. The XML string has the form:

```
<MethodName>
   <Parameter1>Value</Parameter1>
   <Parameter2>Value</Parameter2>
   <Parameter n>Value</Parameter n>
...
</MethodName>
```

where <MethodName> is an XML element containing the name of an IOMI method and <Parameter> is an XML element containing the name of a method parameter.

Multiple methods can be submitted by placing them within a <Multiple_Requests> XML element. For example:

```
<Multiple_Requests>
<MethodName1>
   <Parameter1>Value</Parameter1>
   <Parameter2>Value</Parameter2>
   <Parameter n>Value</Parameter n>
</MethodName1>
<MethodName2>
   <Parameter1>Value</Parameter1>
   <Parameter2>Value</Parameter2>
   <Parameter n>Value</Parameter n>
</MethodName2>
</Multiple_Requests>
```

The published parameter names can, but are not required to be, used for all parameters except *inMetadata*. A <Metadata> XML element must be used to represent the *inMetadata* parameter. When names other than those published are used, the parameters must be passed in the order published in the method documentation.

There is no need to declare an *outMetadata* parameter in the input XML string.

The input XML string is passed to the metadata server in the *inMetadata* parameter of the DoRequest method. The following is an example of an AddMetadata method call that is formatted for the DoRequest method:

```
<AddMetadata>
  <Metadata><PhysicalTable Name="TestTable" Desc="Sample table"/></Metadata>
  <Reposid>A0000001.A2345678</Reposid>
  <NS>SAS</NS>
  <Flags>0</Flags>
  <Options/>
</AddMetadata>
```

The input XML string is submitted to the server as a string literal (a quoted string). To ensure that the string is parsed correctly, it is recommended that any additional double quotation marks in the string, such as those used to denote XML attribute values in the metadata property string, be marked in some way to indicate that they should be treated as characters. Below are examples of using escape characters to accomplish this task:

Java

```
"<PhysicalTable Name=\"TestTable\" Desc=\"Sample table\" />"
```

Visual Basic

```
"<PhysicalTable Name=""TestTable"" Desc=""Sample table"" />"
```

Visual C++

```
"<PhysicalTable Name=\"TestTable\" Desc=\"Sample table\" />"
```

SAS

```
"<PhysicalTable Name=""TestTable"" Desc=""Sample table"" />"

'<PhysicalTable Name="TestTable" Desc="Sample table" />'
```

Any metadata–related (IOMI class) method can be passed to the server using the DoRequest method.

Example

The DoRequest method is issued in the standard interface. The following example shows how to issue a DoRequest method call without regard for the programming environment.

```
outMetadata=program-specific value
inMetadata =
"<GetMetadataObjects>
   <Reposid>A0000001.A2345678</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadataObjects>";

rc=DoRequest(inMetadata,outMetadata);
```

Previous Page | Next Page | Top of Page

FetchMetadata

Copies metadata objects from a primary repository to a project repository without locking them

Category: Change management methods

Syntax

rc = FetchMetadata(inMetadata, outMetadata, projectReposid, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata string identifying an object to be copied.
outMetadata	C	out	Metadata string returned by the server.
projectReposid	C	in	Passed object ID of the target project repository.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_LOCK_TEMPLATE= 65536*** Overrides the default lock template with a user−defined template submitted in the *Options* parameter. ***OMI_RETURN_LIST=1024*** Returns the object identifiers of any associated objects that were fetched in the *outMetadata* parameter.
options	C	in	An XML element that specifies method options. Currently, the only supported option is: ***<LockTemplates>*** Specifies a user−defined lock template. The <LockTemplates> element must be specified in conjunction with the

			OMI_LOCK_TEMPLATE flag.

Details

The FetchMetadata method creates object instances in the project repository that cannot be copied back to the corresponding primary repository.

When a metadata object is fetched, associated objects defined for it in a default lock template are fetched as well. To view the default lock template defined for a metadata type, see Default Lock Templates in the **SAS 9.1 Open Metadata Interface: User's Guide**. To override the default lock template and fetch a different set of associated objects, set the OMI_LOCK_TEMPLATE (65536) flag and supply an alternate lock template in a <LockTemplates> element in the *Options* parameter. For more information about lock templates, see Lock Templates in Using the Change Management Facility, also in the user's guide.

By default, the *outMetadata* parameter lists only the object(s) specified in the *inMetadata* parameter. To include associated objects that were fetched by the lock template, set the OMI_RETURN_LIST (1024) flag.

A fetched object is deleted from a project repository by using the UndoCheckoutMetadata method.

Example 1: Standard Interface

The following FetchMetadata request copies a PhysicalTable object and its associated objects to project repository A0000001.A5NZA58I using the default lock template.

```
inMetadata="<PhysicalTable Id="A5WW3LXC.AA000002"/>";
outMetadata="";
projectReposid="A0000001.A5NZA58I";
ns= "SAS";
flags=0;
options= "";

rc = FetchMetadata(inMetadata, outMetadata, projectReposid, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<FetchMetadata>
   <Metadata>
      <PhysicalTable ID="A5WW3LXC.AA000002"/>
   </Metadata>
   <ProjectReposid>A0000001.A5NZA58I</ProjectReposid>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</FetchMetadata>
```

Related Methods

- CheckoutMetadata
- CheckinMetadata
- UndoCheckoutMetadata

Previous Page | Next Page | Top of Page

GetMetadata

Reads specified metadata from a repository

Category: Read Methods

Syntax

rc = GetMetadata(inMetadata, outMetadata, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata properties to be read from an object
outMetadata	C	out	Metadata property values returned from the object by the read operation
ns	C	in	Namespace to use as the context for the request.
flags	L	in	***OMI_ALL=1*** Specifies that the method get all of the properties of the requested object. If the XML stream returns a reference to any associated objects, then the method will return only general, identifying information for the associated objects. ***OMI_ALL_SIMPLE=8*** Specifies that the method get all of the attributes of the requested object. ***OMI_DEPENDENCY_USED_BY=16384*** Specifies to look for cross−repository references in repositories that use the current repository. For more information, see "Retrieving Information about Cross−Repository

			References" in Creating Relationships Between Repositories in the **SAS 9.1 Open Metadata Interface: User's Guide**. ***OMI_INCLUDE_SUBTYPES=16*** Specifies to include subtypes of the requested type when a template is used. The OMI_INCLUDE_SUBTYPES flag cannot be used without the OMI_TEMPLATES flag. ***OMI_LOCK=32768*** Locks the requested object and any associated objects requested in the method call from update by other users. ***OMI_LOCK_TEMPLATE=65536*** Specifies to lock the associated objects requested in a user−defined lock template rather than those requested by the method. You submit the user−defined lock template in a <LockTemplates> XML element in the *Options* parameter. ***OMI_NOFORMAT=67108864*** Causes date, time, and datetime values in the output XML stream to be returned as raw SAS Date, SAS Time, and SAS Datetime floating point values. Without the flag, the default US−English locale is used to format the values into recognizable character strings. ***OMI_SUCCINCT=2048*** Specifies to omit properties that do not contain values or that contain null values. ***OMI_TEMPLATE=4*** Checks the Options parameter for one or more user−defined templates that specify the properties to return for a given metadata type. The templates are passed in a <Templates> element in the Options parameter.
options	C	in	An XML string that specifies metadata options. Currently, the only supported options are: ***<DOAS Credential="credHandle"/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option. ***<LockTemplates>***

			Specifies associated objects to lock. The <LockTemplates> element must be specified in conjunction with the OMI_LOCK_TEMPLATE flag. ***<Templates>*** Specifies properties to return for an object, by metadata type. The <Templates> element must be specified in conjunction with the OMI_TEMPLATE flag.

Details

The GetMetadata method provides three ways of identifying the metadata that you want to retrieve.

- You can supply a formatted metadata property string and pass it to the GetMetadata method as input in the *inMetadata* parameter. In this case, only the properties whose names have been passed in the property string will be returned. This allows for selective retrieval of pieces of metadata about an object.
- You can pass an *inMetadata* string with just the Type and Id attributes filled in and specify one or more flags in the *Flags* parameter. In this case, all of a particular category of property (for example, all attributes, all associations, or both) that describe the requested object will be retrieved.
- You can pass one or more user−defined templates to expand or to filter the properties requested in the *inMetadata* and *Flags* parameters. For detailed information about templates, see Using Templates in the **SAS 9.1 Open Metadata Interface: User's Guide**.

The method supports optional search criteria on the association name subelements that are passed to the server in the *inMetadata* parameter and in the <Templates> element to filter the associated objects that are retrieved. For more information see Filtering the Associated Objects Returned by a GetMetadata Request in the **SAS 9.1 Open Metadata Interface: User's Guide**.

The GetMetadata method uses the US−English locale to format date, time, and datetime values. Set the OMI_NOFORMAT (67108864) flag to get these values as SAS floating point values that you can format as you want.

You must have a registered identity on the host metadata server in order to set the OMI_LOCK (32768) flag. OMI_LOCK enables you to lock the objects retrieved by the GetMetadata method from use by other users in preparation for making an update. By default, the flag locks the objects specified in the metadata property string and by GetMetadata flags and options. To specify a different set of associated objects to lock, additionally set the OMI_LOCK_TEMPLATE (65536) flag and supply one or more user−defined templates in a XML element in the *Options* parameter. When OMI_LOCK_TEMPLATE is set, the software will lock only the specified object and any associated objects identified in the user−defined lock template. (Specifying OMI_LOCK_TEMPLATE without also specifying OMI_LOCK does not lock any objects.) For information about creating a lock template, see Lock Templates in Using the Change Management Facility in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Metadata objects that are locked by OMI_LOCK are unlocked by issuing an UpdateMetadata method call that sets the OMI_UNLOCK or OMI_UNLOCK_FORCE flag. See Locking Metadata Objects in the **SAS 9.1 Open Metadata Interface: User's Guide** for an overview of available concurrency controls.

Some GetMetadata flags have interdependencies that can affect the metadata that is returned when more than one flag is set. For more information, see Using IOMI Flags.

The following examples issue a GetMetadata method call with no flags and without regard to the programming environment. The first example instantiates objects for method parameters and issues the call using the standard interface. The second formats the same request for the *inMetadata* parameter of the DoRequest method. See also Program−Specific GetMetadata Examples.

Example 1: Standard Interface

The following GetMetadata method call retrieves the name, description, and columns of the PhysicalTable with an Id of A2345678.A2000001.

```
<!-- Create a metadata list to be passed to GetMetadata routine -->
inMetadata= "<PhysicalTable Id="A2345678.A2000001" Name="" Desc="">
               <Columns/>
             </PhysicalTable>";
ns="SAS";
flags=0;
options="";

rc= GetMetadata(inMetadata, outMetadata, ns, flags, options);

<!-- outMetadata XML string returned -->
<PhysicalTable Id="A2345678.A2000001" Name="New Table" Desc="New Table added
through API">
   <Columns>
     <Column Id="A2345678.A3000001" Name="New Column" Desc="New Column added
through API"/>
     <Column Id="A2345678.A3000002" Name="New Column2" Desc="New Column2 added
through API"/>
   </Columns>
</PhysicalTable>
```

Example 2: DoRequest Method

The following XML string formats the request in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<GetMetadata>
   <Metadata>
     <PhysicalTable Id="A2345678.A200001" Name="" Desc="">
        <Columns/>
     </PhysicalTable>
   </Metadata>
   <MS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadata>
```

Related Methods

- GetMetadataObjects

SAS 9.1 Open Metadata Interface: Reference

GetMetadataObjects

Lists metadata objects of the specified type in the specified repository

Category: Read Methods

Syntax

rc = GetMetadataObjects(reposid, type, objects, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
reposid	C	in	Passed repository ID.
type	C	in	Passed type name.
objects	C	out	Returned list of metadata objects.
ns	C	in	Namespace to use as the context for the request.
flags	L	in	***OMI_DEPENDENCY_USED_BY=16384*** Specifies to retrieve objects of the passed type from all repositories that have a DependencyUsedBy dependency in the repository chain. ***OMI_DEPENDENCY_USES=8192*** Specifies to retrieve objects of the passed type from all repositories that have a DependencyUses dependency in the repository chain. ***OMI_NO_DEPENDENCY_CHAIN=16777216*** When used with OMI_DEPENDENCY_USES, retrieves objects only from repositories on which the repository directly depends. When used with OMI_DEPENDENCY_USED_BY, retrieves objects only from repositories that directly depend upon the repository.

			OMI_GET_METADATA=256 Specifies to execute a GetMetadata call for each object that is returned by the GetMetadataObjects request. ***OMI_INCLUDE_SUBTYPES=16*** Specifies to retrieve metadata objects that are subtypes of the passed metadata type in addition to objects of the passed type. If OMI_XMLSELECT is also specified, it affects the metadata and subtypes that are retrieved. ***OMI_MATCH_CASE=512*** Specifies to perform a case−sensitive search of the criteria specified in the <XMLSELECT> option. The OMI_MATCH_CASE flag cannot be used without the OMI_XMLSELECT flag. ***OMI_XMLSELECT=128*** Checks the Options parameter for criteria to qualify the objects that are returned by the server. The criteria are specified in a search string that is passed in the <XMLSELECT> option. See Filtering a GetMetadataObjects Request in the **SAS 9.1 Open Metadata Interface: User's Guide** for more information.
options	C	in	***<DOAS Credential="credHandle"/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option. ***<XMLSELECT>*** Specifies an XML search string.

Details

The GetMetadataObjects method retrieves a list of all objects of the passed type in the passed repository. The default behavior is to return general, identifying information for each object.

You can set IOMI flags to expand or to filter the object request:

- OMI_INCLUDE_SUBTYPES expands the retrieval request to include subtypes.
- OMI_GET_METADATA enables you to issue a GetMetadata call from within the GetMetadataObjects call and specify additional metadata to retrieve for each object.
- OMI_DEPENDENCY_USES and OMI_DEPENDENCY_USED_BY expand the retrieval request to include additional repositories.

- OMI_XMLSELECT enables you to filter the objects that are retrieved by passing an XML search string.

For additional information, including examples of using GetMetadataObjects flags, see the following topics in the **SAS 9.1 Open Metadata Interface: User's Guide**:

- Expanding a GetMetadataObjects Request to Retrieve Additional Attributes
- Expanding a GetMetadataObjects Request to Include Subtypes
- Expanding a GetMetadataObjects Request to Search Additional Repositories
- Filtering a GetMetadataObjects Request

General information about flags, including interdependencies between them, is provided in Using IOMI Flags.

The following examples show how to issue a GetMetadataObjects call with no flags and without regard to the programming environment. See also Program−Specific GetMetadataObjects Examples.

Example 1: Standard Interface

The following GetMetadataObjects request returns all objects defined for metadata type PhysicalTable in repository A0000001.A2345678.

```
<!-- set repository Id and type -->
reposid="A0000001.A2345678";
type="PhysicalTable";
ns="SAS";
flags=0;
options="";

rc = GetMetadataObjects(reposid, type, objects, ns, flags, options);

<!-- XML string returned in objects parameter -->
<Objects>
   <PhysicalTable Id="A2345678.A2000001" Name="New Table"/>
   <PhysicalTable Id="A2345678.A2000002" Name="New Table2"/>
</Objects>
```

Example 2: DoRequest Method

The following XML string formats the request in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->
<GetMetadataObjects>
   <Reposid>A0000001.A2345678</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadataObjects>
```

Related Methods

- GetMetadata

SAS 9.1 Open Metadata Interface: Reference

GetNamespaces

Lists namespaces

Category: Management Methods

Syntax

rc = GetNamespaces(ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
ns	C	out	Returned list of all namespaces
flags	L	in	Passed indicator for flags. For no flags, a 0 should be passed. * Reserved for future use.
options	C	in	Passed indicator for options. Reserved for future use.

Details

A namespace refers to a metadata model that can be accessed by the SAS Open Metadata Interface. The **`ns`** parameter returns the Id (a character string used to represent the Namespace) and Desc properties for all of the namespaces defined in the current repository manager.

The SAS Open Metadata Interface is shipped with two namespaces: the "REPOS" namespace, which contains the repository metadata types, and the "SAS" namespace, which contains metadata types describing application elements.

The following examples show how to issue a GetNamespaces method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For program−specific examples, see Program−Specific GetNamespaces Examples.

Example 1: Standard Interface

The following GetNamespaces request returns the namespaces in the current repository manager using the method interface.

```
flags=0;
options="";

rc = GetNamespaces(ns,flags,options);

<!-- XML string returned in ns parameter -->
<Namespaces>
   <Ns Name="SAS"/>
   <Ns Name="REPOS"/>
</Namespaces>
```

Example 2: DoRequest Method

The following XML string shows how to format the same request for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->
<GetNamespaces>
   <NameSpaces/>
   <Flags>0</Flags>
   <Options/>
</GetNamespaces>
```

Related Methods

- GetTypes
- GetSubtypes

Previous Page | Next Page | Top of Page

GetRepositories

Lists the metadata repositories available through this server

Category: Repository Methods

Syntax

rc = GetRepositories(repositories, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
repositories	C	out	Returned list of all repositories that are available to the current server.
flags	L	in	***OMI_ALL=1*** Gets the Access and PauseState attributes of all repositories and the repository manager.
options	C	in	Passed indicator for options. Reserved for future use.

Details

A repository is a collection of related metadata objects. Each repository is registered in a repository manager. The SAS Metadata Server can access only those repositories that are registered in its repository manager. There is one repository manager for a SAS Metadata Server.

Issued without flags, the GetRepositories method returns a list of the repositories that are registered in the repository manager, as well as the default namespace and general, identifying information for each repository.

Setting OMI_ALL additionally returns the Access and PauseState attributes of the repository manager and all repositories. These attributes indicate the availability of the repository manager and individual repositories. For a description of the attributes, see the RepositoryBase type.

A GetRepositories method call issued on a repository manager that is paused to an offline state will return an error.

Examples

The following examples show how to issue a GetRepositories method call without regard to the programming environment. For examples, see Program–Specific GetRepositories Examples.

Standard Interface

The following is an example of a GetRepositories method that is issued without flags.

```
flags=0;
options= "";

rc = GetRepositories(repositories,flags,options);

<!-- XML string returned in repositories parameter -->
<Repositories>
   <Repository Id="A0000001.A5345678" Name="Sample API Repository"
     Desc="Repository with Sample Metadata"  DefaultNS="SAS"/>
   <Repository Id="A0000001.A5934023" Name="Sales Repository"
     Desc="Repository representing all sales for the first quarter"
     DefaultNS="SAS"/>
</Repositories>
```

DoRequest Method

The following is an example of a GetRepositories method that is issued with the OMI_ALL flag set.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<GetRepositories>
   <Repositories/>
   <Flags>1</Flags>
   <Options/>
</GetRepositories>

<!-- XML string returned in repositories parameter -->
<Repositories>
   <Repository Id="A0000001.A5345678" Name="Sample API Repository"
     Desc="Repository with Sample Metadata"  DefaultNS="SAS"
     Access="OMS_FULL" PauseState="" />
   <Repository Id="A0000001.A5934023" Name="Sales Repository"
     Desc="Repository representing all sales for the first quarter"
     DefaultNS="SAS" Access="OMS_FULL" PauseState="READONLY" />
</Repositories>
```

GetSubtypes

Returns the subtypes for a specified metadata type

Category: Management Methods

Syntax

rc = GetSubtypes(supertype, subtypes, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
supertype	C	in	Name of the metadata type for which you want a list of subtypes.
subtypes	C	out	Returned XML list of all subtypes for the specified type.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_ALL_DESCENDANTS=64*** Returns all subtypes along with all of their descendants.
options	C	in	Passed indicator for options. ***<DOAS Credential="credHandle"/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option.

Details

Subtypes are metadata types that take on the characteristics of a specific metadata supertype. In addition, a subtype can have subtypes of its own.

The *subtypes* parameter returns an XML string that consists of the Id, Desc, and a HasSubtypes attribute for each subtype. The HasSubtypes attribute indicates whether a subtype has any subtypes of its own. If this attribute has a value of 0, then the subtype does not have subtypes. If it has a value of 1, then the subtype does have subtypes.

The GetSubtypes method will not return metadata about descendants unless the OMI_ALL_DESCENDANTS flag is set.

The following examples show how to issue a GetSubtypes method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For examples, see Program−Specific GetSubtypes Examples.

Example 1: Standard Interface

The following GetSubtypes request lists the subtypes defined for supertype DataTable.

```
supertype= "DataTable";
ns= "SAS";
flags= 0;
options= "";

rc = GetSubtypes(supertype, subtypes, ns, flags, options);

<!-- XML string returned in the Subtypes parameter -->
<subtypes>
   <Type Id="PhysicalTable" Desc="Physical Storage Abstract Type" HasSubtypes="0"/>
   <Type Id="WorkTable" Desc="Work Tables" HasSubtypes="1"/>
   <Type Id="Join" Desc="Table Joins" HasSubtypes="0"/>
</subtypes>
```

Example 2: DoRequest Method

The following XML string shows how to format the request in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<GetSubtypes>
   <Supertype>DataTable</Supertype>
   <Subtypes/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetSubtypes>
```

Related Methods

- GetTypes
- IsSubtypeOf

SAS 9.1 Open Metadata Interface: Reference

GetTypeProperties

Returns the properties for a metadata type

Category: Management Methods

Syntax

rc = GetTypeProperties(type, properties, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
type	C	in	Name of the metadata type for which you want a list of properties.
properties	C	out	Returned XML list of all properties for the specified metadata type.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	Passed indicator for flags. Reserved for future use.
options	C	in	Passed indicator for options. ***<DOAS Credential=****"credHandle"****/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option.

Details

Properties consists of an XML string that contains all of the properties for a specific metadata type.

The following examples show how to issue a GetTypeProperties method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method.

For examples, see Program−Specific GetTypeProperties Examples.

Example 1: Standard Interface

The following GetTypeProperties method call requests the properties of the Column metadata type.

```
type="Column";
ns="SAS";
flags=0;
options="";

rc = GetTypeProperties(type, properties, ns, flags, options);

<!-- sample XML string returned in Properties parameter -->
<Properties Cardinality="" ColType="" Cvalue=""   Desc="" Format="" Id=""
Informat="" IsNullable="" Length="" Max="" MetadataCreated="" MetadataUpdated=""
Min="" Moment1=""  Moment2="" Name="" Nmiss="" Nvalue="" SortOrder=""
Statistic="" Sum="" SummaryRole="" UserType="">
   <Administrator/>
   <Extensions/>
   <Key/>
   <KeyAssoc/>
   <MemberGroups/>
   <Note/>
   <Owner/>
   <Packages/>
   <Table/>
</Properties>
```

Example 2: DoRequest Method

The following XML string shows how to format the request in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<GetTypeProperties>
   <Type>Column</Type>
   <Properties/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
<GetTypeProperties>
```

Related Methods

- GetTypes
- GetSubtypes

GetTypes

Lists the metadata types in a namespace

Category: Management Methods

Syntax

rc = GetTypes(types, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
types	C	out	Returned XML list of metadata types.
ns	C	in	Passed namespace to use as the context for the request. Use a " (double quotation mark) to indicate the default namespace of the repository.
flags	L	in	***OMI_SUCCINCT=2048*** Specifies to list metadata types for which objects exist in a given repository. The repository is identified in a <Reposid> element in the *Options* parameter.
options	C	in	An XML string that specifies metadata options. Currently, the only supported option is: ***<Reposid>*** Specifies a unique repository identifier, in the form `A0000001.RepositoryId` where A0000001 is the repository manager identifier. The <Reposid> element must be specified in conjunction with the

			OMI_SUCCINCT flag.

Details

The GetTypes method has two behaviors, depending on whether the OMI_SUCCINCT flag and its corresponding <Reposid> element are passed.

- Used without the flag, the method returns an XML string that lists all of the metadata types defined in the passed namespace.
- Used with the flag, the method returns information only for metadata types for which objects exist in the specified repository.

In either case, the XML string is returned in the *Types* parameter and has a HasSubTypes attribute that indicates whether a type has any subtypes. If this attribute has a value of 0, then the type does not have any subtypes. If it has a value of 1, then the type does have subtypes.

The following examples show how to issue a GetTypes method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For examples, see Program−Specific GetTypes Examples. For an example of a call that uses the OMI_SUCCINCT flag, see Listing the Types in a Repository in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Example 1: Standard Interface

The following GetTypes request lists the metadata types defined in the SAS namespace.

```
ns= "SAS";
flags= 0;
options= "";

rc = GetTypes(types, ns, flags, options);
```

This is a partial example of the output received from the server.

```
<!-- XML string returned -->
<Types>
   <Type Id="AbstractColumn" Desc="Abstract Column" HasSubTypes="1"/>
   <Type Id="AbstractGroup" Desc="Abstract Group" HasSubTypes="1"/>
   <Type Id="ApplicationTree" Desc="The root object for an application." HasSubTypes="0"/>
...
</Types>
```

Example 2: DoRequest Method

This is an example of the same request formatted for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<GetTypes>
   <Types/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
```

```
</GetTypes>
```

Related Methods

- GetNamespaces
- GetSubtypes

Previous Page | Next Page | Top of Page

IsSubtypeOf

Determines if one metadata type is a subtype of another

Category: Management Methods

Syntax

rc = IsSubTypeOf(type, supertype, result, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
type	C	in	Name of the metadata type that may be a subtype of *supertype*.
supertype	C	in	Name of the metadata type that may be a supertype of *type*.
result	N	out	Returned indicator. 0 indicates that *type* is not a subtype of *supertype*. 1 indicates that it is a subtype.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	Passed indicator for flags. Reserved for future use.
options	B	in	Passed indicator for options. ***<DOAS Credential=****"credHandle"****/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option.

The following examples show how to issue an IsSubType method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For examples, see Program−Specific IsSubtypeOf Examples.

Example 1: Standard Interface

The following IsSubtypeOf request determines whether WorkTable is a subtype of DataTable.

```
ns="SAS";
flags=0;
options="";

rc = IsSubtypeOf(WorkTable, DataTable, result, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<IsSubtypeOf>
   <Type>WorkTable</Type>
   <Supertype>DataTable</Supertype>
   <Result/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
<IsSubtypeOf>
```

Related Methods

- GetSubtypes

Previous Page | Next Page | Top of Page

UndoCheckoutMetadata

Deletes the specified object from the project repository and unlocks its corresponding primary object

Category: Change management methods

Syntax

rc = UndoCheckoutMetadata(inMetadata, outMetadata, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Metadata string identifying the objects to be deleted and unlocked.
outMetadata	C	out	Metadata string returned by the server.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_LOCK_TEMPLATE= 65536*** Overrides the default lock template with a user−defined template submitted in the *Options* parameter. ***OMI_RETURN_LIST=1024*** Includes the object identifiers of any associated objects that were unchecked in the method output.
options	C	in	An XML element that specifies method options. Currently, the only supported option is: ***<LockTemplates>*** Specifies a user−defined lock template. The <LockTemplates> element must be specified in conjunction with the

			OMI_LOCK_TEMPLATE flag.

Details

The UndoCheckoutMetadata method enables you to remove from the project repository objects that may have been checked out by mistake. It also enables you to delete metadata objects copied to the project repository by FetchMetadata.

To undo checkout of an object, specify its metadata type and project repository object instance identifier in the *inMetadata* parameter. The corresponding primary object identified in the metadata object's ChangeState attribute is immediately unlocked and the specified object is deleted from the project repository.

When a metadata object is unchecked, any associated objects identified in the object's default lock template are unchecked as well. **An object must be unchecked using the same lock template that was used to check out or fetch it.** That is, if a user−defined lock template was used to check out or fetch the object, then this same template must be passed in the UndoCheckoutMetadata method. For more information about lock templates, see Lock Templates in Using the Change Management Facility in the **SAS 9.1 Open Metadata Interface: User's Guide**.

By default, the *outMetadata* parameter lists only the object(s) specified in the *inMetadata* parameter. Set the OMI_RETURN_LIST flag to include associated objects that were unchecked in the output.

Example 1: Standard Interface

The following UndoCheckoutMetadata request deletes PhysicalTable object B700005N from project repository A3VTX83H and unlocks the corresponding primary object identified in the ChangeState attribute. The OMI_RETURN_LIST flag lists any associated objects that are also deleted.

```
inMetadata="<PhysicalTable Id="A3VTX83H.B700005N"/>";
outMetadata="";
ns= "SAS";
flags=1024;
options= "";

rc = UndoCheckoutMetadata(inMetadata, outMetadata, ns, flags, options);
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<UndoCheckoutMetadata>
  <Metadata>
     <PhysicalTable Id="A3VTX83H.B700005N"/>
  </Metadata>
  <Ns>SAS</Ns>
<!--OMI_RETURN_LIST flag-->
  <Flags>1024</Flags>
  <Options/>
 </UndoCheckoutMetadata>
```

Related Methods

- CheckoutMetadata
- CheckinMetadata
- FetchMetadata

Previous Page | Next Page | Top of Page

UpdateMetadata

Updates specified metadata in a repository

Category: Write Methods

Syntax

rc = UpdateMetadata(inMetadata, outMetadata, ns, flags, options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see Return Code.
inMetadata	C	in	Passed metadata property string for the object to be updated. For the general format of this string, see Constructing a Metadata Property String.
outMetadata	C	out	Returned metadata property string with the requested values filled in.
ns	C	in	The namespace to use as the context for the request.
flags	L	in	***OMI_IGNORE_NOTFOUND = 134217728*** Prevents an update operation from being aborted when a request specifies to update an object that does not exist. ***OMI_RETURN_LIST = 1024*** Returns the object identifiers of any dependent objects that were deleted as a result of the update operation in the *outMetadata* parameter. ***OMI_TRUSTED_CLIENT = 268435456*** Determines if the client can call this method. This flag is required. ***OMI_UNLOCK=131072***

			Unlocks a lock that is held by the calling user. ***OMI_UNLOCK_FORCE=262144*** Unlocks a lock that is held by another user.
options	C	in	Passed indicator for options. ***<DOAS Credential=****"credHandle"****/>*** Specifies an alternate calling identity for the request. For more information, see <DOAS> Option.

Details

The UpdateMetadata method enables you to update the properties of existing objects. It will issue an error if the object to be updated does not exist, unless the OMI_IGNORE_NOTFOUND flag is set.

You can modify both an object's attributes and associations, unless the association is designated as "required for add" in the metadata type documentation.

You specify directives on the association name to indicate whether the association is being appended, modified, removed, or replaced. Different directives are supported for single and multiple associations. For information about these directives and general UpdateMetadata usage, see Updating Metadata Objects in the **SAS 9.1 Open Metadata Interface: User's Guide**.

You must have a registered identity on the host metadata server in order to set the OMI_UNLOCK (131072) and OMI_UNLOCK_FORCE (262144) flags. These flags unlock objects that were previously locked by the OMI_LOCK flag or that were locked by the change management facility. The OMI_LOCK flag is set in a GetMetadata method call to provide basic concurrency controls in preparation for an update. The change management facility implements a checkout/checkin scheme for making metadata updates. A metadata object locked by OMI_LOCK is unlocked after the specified object is updated. A metadata object locked by the change management facility is simply unlocked. Any associated objects that were locked will be unlocked as well.

The following examples show how to issue a simple UpdateMetadata method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For examples, see Program−Specific UpdateMetadata Examples.

Example 1: Standard Interface

The following is an example of an UpdateMetadata method call that modifies an object's attributes. The specified attributes and values will replace those stored for the object of the specified metadata type and repository and object instance identifier.

```
<!-- Create a metadata list to be passed to UpdateMetadata -->

inMetadata= "<PhysicalTable Id="A2345678.A2000001" Name="Sales Table" DataName="Sales"
             Desc="Sales for first quarter"/>";
ns= "SAS";
flags= 268435456;  <!- OMI_TRUSTED_CLIENT flag ->
```

```
options= "";

rc = UpdateMetadata(inMetadata, outMetadata, ns, flags, options);

<!-- outMetadata XML string returned -->
<PhysicalTable Id="A2345678.A2000001" Name="Sales Table" DataName="Sales"
    Desc="Sales for first quarter"/>
```

Example 2: DoRequest Method

The following XML string shows how to format the method call in Example 1 for the *inMetadata* parameter of the DoRequest method.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->

<UpdateMetadata>
   <Metadata>
      <PhysicalTable Id="A2345678.A2000001" Name="Sales Table" DataName="Sales" Desc="Sales for
   </Metadata>
   <NS>SAS</NS>
   <!- OMI_TRUSTED_CLIENT flag ->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

Related Methods

- DeleteMetadata
- GetMetadata

Previous Page | Next Page | Top of Page

ISecurity Class

Overview

The methods described in this section are provided in the ISecurity method class and can be used in a Java, Visual Basic, or C++ thin client that you create to restrict access to metadata resources. The methods can be used to authorize access both to metadata and to the data that is represented by the metadata.

ISecurity methods are available only through the standard interface.

The following information applies to all of the ISecurity methods:

- The variable *rc* returns void. Errors are surfaced through the exception handling in the IOM mechanism.
- *resource* is a Uniform Resource Name (URN) in the form:

  `OMSOBJ:`*`MetadataType/objectId`*
- For 9.1, the methods assume the calling user and any user IDs specified by the calling program have been authenticated prior to calling the SAS Metadata Server and that the caller is a "trusted user" of the metadata server.
- The methods base authorization decisions on user and access control metadata stored in SAS metadata repositories. User and access control metadata is defined using the User Manager and Authorization Manager plug−ins to SAS Management Console.

Using the ISecurity Class

The ISecurity class enables you to

- get and free a credential handle for an authenticated user
- determine whether an authenticated user is authorized to access a resource with a specific permission
- determine which identity object represents the authenticated user in the metadata server
- issue special calls, for example, to determine a user's authorizations on all parts of an OLAP cube.

For more information, see the descriptions of the individual methods.

Previous Page | Next Page | Top of Page

FreeCredentials

Frees the credential handle obtained by GetCredentials

Category: Authorization Methods

Syntax

rc = FreeCredentials(credHandle);

Parameters

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle to free.

Details

The FreeCredentials method frees the metadata server−side credentials associated with a credential handle returned by the GetCredentials method. Each credential handle returned by the GetCredentials method should be freed.

Example

Note: The FreeCredentials method is supported only in the standard interface.

The following is a Java example of a FreeCredentials call.

```
FreeCredentials(credHandle_value);
```

Related Methods

- GetCredentials

Previous Page | Next Page | Top of Page

SAS 9.1 Open Metadata Interface: Reference

GetAuthorizations

Gets a variety of authorization information, depending on the type of authorization requested

Category: Authorization Methods

Syntax

rc = GetAuthorizations(authType, credHandle, resource, permission, authorizations);

Parameters

Parameter	Type	Direction	Description
authType	string	in	The type of authorization to perform.
credHandle	string	in	Credential handle identifying a user identity or an empty string.
resource	string	in	Passed resource identifier.
permission	string	in	Mnemonic representation of the permission for which authorization is being sought. This parameter can be an empty string for some authType values.
authorizations	string	out	Returned two−dimensional string. The content and structure of this array will vary depending on the authorization type specified in the *authType* parameter.

Details

The GetAuthorizations method performs special authorization−related queries. The input required for processing the query and the format and content of the information returned is determined by the *authType* parameter. Currently, the only supported authType value is "Cube".

"Cube" returns an array of strings[*][4]. The number of rows depends on the structure of the cube and each row has four columns:

Type
indicates the metadata type in the row. This will be either "Hierarchy", "Dimension", "Measure", or "Level".

Name
returns the Name attribute of the type instance.

Authorized
returns a "Y" or "N" indicating whether the permission being sought has been granted to the user in this component of the cube structure.

PermissionCondition
returns a condition that must be enforced on this particular cube component to allow access. For more information about permission conditions, see the description of the IsAuthorized method *permissionCondition* parameter.

If the *credHandle* parameter is an empty string, a credential handle is returned for the user making the request.

Examples

Note: The GetAuthorizations method is supported only in the standard interface.

The following are Java examples of a GetAuthorizations call.

The following GetAuthorizations call returns a matrix containing resource authorization decisions on cube A5J52Z80.AT000005 for the user identified by *credHandle_value*. Specifying the ReadMetadata and Read permissions in the *Permissions* parameter returns only authorizations with these permissions.

```
String type       = "Cube";
String resource   = "OMSOBJ:Cube/A5J52Z80.AT000005";
String permission = "ReadMetadata,Read";

StringHolder  permissionCondition = new org.omg.CORBA.StringHolder();
com.sas.iom.SASIOMDefs.VariableArray2dOfStringHolder returnAuthorizations =
       new com.sas.iom.SASIOMDefs.VariableArray2dOfStringHolder();

GetAuthorizations(type,credHandle_value,resource,permission,returnAuthorizations);
```

The following is an example of a GetAuthorizations call that passes an empty string in the *credHandle* parameter to return authorizations for the requesting user.

```
GetAuthorizations(type,"",resource,permission,returnAuthorizations);
```

Related Methods

- IsAuthorized

GetCredentials

Returns a handle to a provider−specific credential

Category: Authorization Methods

Syntax

rc = GetCredentials(credHandle,subject);

Parameters

Parameter	Type	Direction	Description
credHandle	string	out	Returned credential handle representing the subject.
subject	string	in	Passed user ID of the authenticated user for whom a credential is sought or an empty string.

Details

The GetCredentials method returns a credential handle for the user identified in the *subject* parameter. If the *subject* parameter contains an empty string, a credential handle is returned for the user making the request.

A credential handle is a token representing a user's authorizations on the SAS Metadata Server that can be stored on an interim server to reduce the number of authorization requests that are made to the server on behalf of a given user. A credential describes the privilege attributes (identity and group memberships) of the subject, as registered in the metadata server. Every credential handle that is returned by the GetCredentials method should be freed using the FreeCredentials method.

Example

Note: The GetCredentials method is supported only in the standard interface.

The following is a Java example of a GetCredentials call that returns a handle for the user identified by MYDOMAIN\MYUSERID.

```
String subject = new String("myDomain\myUserID");
StringHolder credHandle = new org.omg.CORBA.StringHolder();
```

```
GetCredentials(subject,credHandle);
```

Related Methods

- FreeCredentials
- GetIdentity

Previous Page | Next Page | Top of Page

GetIdentity

Gets identity metadata for the specified user

Category: Authorization Methods

Syntax

rc = GetIdentity(credHandle,identity);

Parameters

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle identifying a user identity, an empty string, or a user ID in the form "LOGINID:*userid*".
identity	string	out	Metadata object identifier of the identity represented by the credential handle.

Details

Given a credential handle, the GetIdentity method will return a URN−like string representing the metadata object identifier of the identity that corresponds to the credential handle. An "identity" refers to the Person or IdentityGroup metadata object describing a user in a SAS metadata repository.

The URN−like string is in the form

`OMSOBJ: MetadataType/objectId`

where

- *MetadataType* is the IdentityGroup or Person metadata type
- *objectId* is a unique object instance identifier in the form *reposid.objectId*.

If the *credHandle* parameter is an empty string, the returned identifier represents the requesting user.

If the user ID of the user on whose behalf the call is made is known, specify it in the form "LOGINID:userid" to eliminate the need to issue GetCredentials and FreeCredentials calls prior to GetIdentities. In the "LOGINID:*userid*" string

- LOGINID specifies to search for Login objects
- *userid* is the value of a Login object's userID attribute.

Examples

Note: The GetIdentity method is supported only in the standard interface.

The following are Java examples of a GetIdentity call.

```
// GetIdentity() returns a URN-like string representing the metadata object
// identifier of an identity specified in the first parameter.  The first
// parameter can be specified in one of three ways:

 StringHolder identityValue = new org.omg.CORBA.StringHolder();

//   1) The first parameter is an empty string:
//      GetIdentity() returns the Identity associated with
//      the current connection to the SAS Metadata Server.

 GetIdentity("",identityValue);

//   2) The first parameter is a valid credential handle:
//      Here the returned Identity corresponds to the credential handle
//      obtained in the previous call to GetCredentials().

 GetIdentity(credHandle.value,identityValue);

//   3) The first parameter is a user ID with the prefix: 'LOGINID:'
//      Here the returned Identity corresponds to specified user ID.

 String loginId = new String("LOGINID:myUserID");
 GetIdentity(credHandle.value,identityValue);
```

Related Methods

- GetCredentials

Previous Page | Next Page | Top of Page

IsAuthorized

Determines whether an authenticated user is authorized to access a resource with a specific permission

Category: Authorization Methods

Syntax

rc = IsAuthorized(credHandle,resource,permission,permissionCondition,authorized);

Parameters

Parameter	Type	Direction	Description
credHandle	string	in	Credentials handle identifying a user identity or an empty string.
resource	string	in	Passed resource identifier.
permission	string	in	Passed user access permission(s).
permissionCondition	string	out	Returned permission condition(s) associated with the data access.
authorized	boolean	out	A boolean value that indicates whether access to a resource is granted or denied.

Details

If the *credHandle* parameter is an empty string, a credential handle is returned for the user making the request.

The *resource* parameter identifies the application element to which access is requested. It accepts a Uniform Resource Name (URN) in the form

`OMSOBJ: metadataType/objectId`

The *Permission* parameter accepts the name of a Permission metadata object that is defined in a SAS metadata repository. A Permission object is just another metadata object in a SAS metadata repository. As new applications are added or if users need additional permissions, then additional Permission objects can be added to the repository. The standard Permission objects are:

ReadMetadata
indicates a user can read a metadata object.
WriteMetadata
indicates a user can write a metadata object.
Read
indicates a user can read the data represented by a metadata object.
Write
indicates a user can write the data represented by a metadata object.

The *Permissions* parameter accepts multiple, comma−delimited permissions (for example, "ReadMetadata, WriteMetadata, Read, Write, Administer"). When multiple permissions are passed, the isAuthorized method parses them and permits access only if the specified user has access based on the sum of the specified permissions.

The *permissionCondition* parameter is used in association with "data" permissions such as "Read" and "Write." It indicates that a permission is granted, but only if the specific condition is met. The syntax of the permission condition is not defined; it is specific to the resource being protected and the technology responsible for enforcing the security of the resource. For example, a permissionCondition for a table would likely be a SQL Where clause, but for an OLAP Dimension it would be an MDX expression identifying the level members that can be accessed in the dimension.

Note: In SAS 9.1, PermissionCondition objects are only supported for OLAP Dimensions.

It is possible for a user to have multiple conditions associated with their access to a resource. In this case, the *permissionCondition* parameter will be returned with multiple strings embedded. Each embedded condition will be separated from the preceding condition by the string "<!−−CONDITION−−>". If you receive a permissionCondition, you must check to see if it contains multiple conditions by searching for <!−−CONDITION−−> in the returned string. If multiple conditions are found, then they should be used to filter data such that the result is a union of the data returned for each condition individually. In other words, the conditions would be ORed together.

Examples

Note: The IsAuthorized method is supported only in the standard interface.

The following are Java examples of an IsAuthorized call.

This example determines if the user represented by *credHandle_value* has ReadMetadata permission to the PhysicalTable object identified as 'A5J52Z80.AK000001'.

```
String resource   = "OMSOBJ:PhysicalTable/A5J52Z80.AK000001";
String permission = "ReadMetadata";

StringHolder  permissionCondition = new org.omg.CORBA.StringHolder();
BooleanHolder authorized = new org.omg.CORBA.BooleanHolder();

IsAuthorized(credHandle_value,resource,permission,permissionCondition,authorized);
```

The following example passes an empty string to return an authorization decision for the requesting identity.

```
IsAuthorized("",resource,permission,permissionCondition,authorized);
```

Related Methods

- GetAuthorizations

Previous Page | Next Page | Top of Page

IServer Class

Overview

The methods described in this section are provided in the IServer method class and can be used in a Java, Visual Basic, or C++ thin client that you create to perform repository and server administrative tasks. For example, you can pause client activity on all or specific repositories, including the repository manager, in preparation for performing a backup. You can also refresh a running server to change its configuration options.

The Stop method is the recommended method for stopping the server.

Warning: Using operating system commands to stop the server process can corrupt your repositories.

In the examples, note the following:

- serverObject is an instantiation of the IServer method class.
- repositoryObject is an instantiation of the RepositoryBase metadata type.
- The variable rc captures the return code of the method.

Note: In SAS 9.1, only the user ID that started the SAS Metadata Server and user IDs that are granted administrative user status can pause, refresh, and resume repositories and the repository manager, and stop the server. Anyone can issue the Status method. For information about administrative user status, see "Server Administrative Privileges" in the **SAS 9.1 Metadata Server: Setup Guide**.

Previous Page | Next Page | Top of Page

Pause

Limits the availability of a repository, the repository manager, or all repositories

Category: Repository Control

Syntax

rc=Pause(options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1).
options	C	in	***<REPOSITORY>*** specifies an XML property string identifying a repository to pause.

Details

The Pause method is issued on a running server to temporarily limit the access mode specified in a repository's Access attribute. The values supported in the Access attribute are described in RepositoryBase. To determine the access mode in force for a given repository, issue a GetRepositories method, setting the OMI_ALL flag. The GetRepositories method will return a list of all repositories registered on the server, as well as the values in their Access and PauseState attributes. A value in the PauseState attribute indicates a repository has been paused.

When passed without options, the Pause method quiesces client activity in all repositories on the server, except the repository manager, and changes them to an OFFLINE state. This closes all repository files on the server and de−assigns the librefs to the repositories.

The <Repository> XML element enables you to optionally identify a specific repository to pause, including the repository manager, and a pause state of READONLY instead of OFFLINE. The <Repository> XML element is passed in the *Options* parameter and has the form

```
<REPOSITORY Id="Reposid|REPOSMGR|ALL" State="OFFLINE|READONLY">
```

where

- Id= is required and specifies the unique 8−character or 17−character identifier of a repository, or the keywords REPOSMGR or ALL. The REPOSMGR value pauses the repository manager. The ALL value pauses all repositories on the server except the repository manager. If Id= is omitted or specified

without a value, it has the same effect as specifying Id="ALL".

- State= specifies OFFLINE or READONLY. READONLY permits clients to read metadata in the specified repository or repository manager, but prevents them from writing to it. OFFLINE disables read and write access to a repository or the repository manager. If a State value is omitted from the <Repository> element, the default state is READONLY.
- Multiple specific repositories can be paused at once by stacking their <Repository> elements in the *Options* parameter.

The values REPOSMGR, ALL, OFFLINE, and READONLY must be specified in uppercase letters.

When Pause is issued with one or more <Repository> tags, the metadata server processes only the indicated repositories and/or repository manager by closing all repository files and de−assigning the repository libref. Those paused for READONLY are re−assigned readonly librefs.

A repository (or repository manager) that is paused must be resumed using the Resume method. The repository files cannot be re−opened until the repositories are resumed.

The metadata server must pause, resume, and refresh repositories when no other activity is taking place in the server, so it automatically delays other client requests until these services are complete. This may have a small effect on server performance.

Because of the pervasive nature of the method, a foundation repository should not be paused to an OFFLINE state without also pausing all repositories that depend on it. When a foundation repository is paused for OFFLINE, all permissions, inheritance rules, and user registrations it contains will no longer be available to dependent repositories. A request for a permission that is not found will be denied. When identity metadata cannot be found, the requesting user is treated as PUBLIC. When inheritance rules cannot be found, the only access controls evaluated when making an authorization decision are those directly on an object and the Default ACT.

The Pause method is supported only in the standard interface.

Example

The following example pauses a repository for READONLY.

```
<!--Pause repository A5H9YT45 for READONLY-->
options='<Repository Id="A5H9YT45" State="READONLY"/>';

rc=Pause(options);
```

Previous Page | Next Page | Top of Page

Refresh

Changes server configuration and invocation options on a running metadata server. Also pauses and resumes a repository in a single step.

Category: Server Control/Repository Control

Syntax

rc=Refresh(options);

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1).
options	C	in	***<ARM>*** an XML element that specifies one or more ARM system options. ***<REPOSITORY>*** an XML element that identifies a repository to pause and resume.

Details

Refresh is issued on a running server. It has two uses:

- It can quiesce client activity on the server long enough to invoke the specified server configuration option(s) and then automatically resume the server. Currently, one configuration option is supported. The <ARM> XML element specifies one or more ARM system options as follows:

  ```
  <ARM ARMSUBSYS="(ARM_NONE|ARM_OMA)" ARMLOC="fileref|filename">
  ```

  The ARMSUBSYS= option permits you to enable or disable ARM logging. If ARM logging is already enabled, specifying ARMLOC= writes the ARM log to a new location. Note that absolute and relative pathnames are read as different locations. See "Starting the ARM_OMA Subsystem" in the **SAS 9.1 Metadata Server: Setup Guide** for syntax details.
- It can pause and resume a repository in a single step. The memory footprint of the server can be reduced by pausing and resuming one or more repositories because the files associated with the repositories are closed. Subsequent client requests will reopen the files as they are needed. The repository to pause is identified in the <Repository> XML element in the form:

```
<REPOSITORY Id="Reposid|REPOSMGR|ALL" State="READONLY|OFFLINE"/>
```

For syntax details, see the Pause method.

When used with the <Repository> element, the Refresh method clears a repository's PauseState attribute.

Issuing the Refresh method without options has no effect.

The metadata server must pause, resume, and refresh repositories when no other activity is taking place in the server, so it automatically delays other client requests until these services are complete. This may have a small effect on server performance.

The Refresh method is supported only in the standard interface.

Examples

The following example enables ARM_OMA logging.

```
options='<ARM armsubsys="(ARM_OMA)" armloc="myARM.log"/>';
rc=serverObject.Refresh(options);
```

The following example disables ARM_OMA logging.

```
options='<ARM armsubsys="(ARM_NONE)"/>';
rc=serverObject.Refresh(options);
```

The following example pauses and resumes a repository to temporarily take it offline, for example, to reduce its memory footprint.

```
options='<Repository Id="A5H9YT45" State="offline"/>';
rc=serverObject.Refresh(options);
```

Previous Page | Next Page | Top of Page

Resume

Restores client activity in one or more repositories

Category: Repository Control

Syntax

```
rc=Resume(options);
```

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1).
options	C	in	***<REPOSITORY>*** specifies an XML property string identifying a repository to resume.

Details

The Resume method restores client activity in a repository that has been paused by the Pause method. Passed without options, the Resume method restores client activity in all repositories, except the repository manager. A <Repository> XML element enables you to identify a specific repository in which to restore activity or to restore activity on the repository manager.

The <Repository> XML element is passed in the *options* parameter and has the form

```
<REPOSITORY ID="Reposid|REPOSMGR|ALL">
```

where ID= is the unique 8−character or 17−character identifier of a repository, REPOSMGR, or ALL. The default, ALL, resumes activity in all repositories, except the repository manager. The values REPOSMGR and ALL must be specified in uppercase letters.

The Resume method restores a repository to the access mode specified in its Access attribute and clears the value in the repository's PauseState attribute. See the RepositoryBase type for more information about these attributes.

Note: In order for security rules to be reloaded correctly after a Pause, a foundation repository and its dependent repositories should be resumed at the same time.

The Resume method is supported only in the standard interface.

Standard Interface Example

The following example resumes a paused repository.

```
<!--Resume repository A5H9YT45 to its normal access mode-->
options='<Repository Id="A5H9YT45"/>';

rc=Resume(options);
```

Previous Page | Next Page | Top of Page

Status

Polls the server to determine its availability

Category: Server Control

Syntax

```
rc=Status(inmeta, options, outmeta);
```

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1).
inmeta	string	in	*<State/>* requests the server's availability. *<Version/>* requests the SAS Metadata Model version number.
outmeta	string	out	Mirrors the inmeta parameter except it has return values filled in.
options	C	in	No options are supported at this time.

Details

The *inmeta* parameter is a string containing an XML element that specifies the requested information.

The *outmeta* parameter mirrors the *inmeta* parameter and returns the following values:

<State/>
: a value of "0" indicates the metadata server is running; a value of "1" indicates that all repositories, and therefore the server, are paused to an OFFLINE state; no response means the server is down.

<Version/>
: returns the SAS Metadata Model version number, for example, **4.01**.

To get information about repositories that are paused to a READONLY state, use the GetRepositories method. For Status and GetRepositories usage information, see "Checking Repository and Server Status" in the **SAS 9.1 Metadata Server: Setup Guide**.

Standard Interface Example

```
<!--Determine availability and version number of the server-->
inmeta='<State/> <Version/>'
outmeta=''
options=''

rc=Status(inmeta,outmeta,options);
```

DoRequest Example

```
<!--Determine availability and version number of the server-->
<Status>
 <Metadata>
   <State/>
   <Version/>
 </Metadata>
 <Options/>
</Status>
```

Previous Page | Next Page | Top of Page

Stop

Shuts down the SAS Metadata Server

Category: Server Control

Syntax

```
serverObject.Stop(options);
```

Parameters

Parameter	Type	Direction	Description
options	C	in	No options are supported at this time.

Details

A return code of 0 indicates that the metadata server was successfully stopped. A return code other than 0 indicates that the metadata server failed to stop.

The Stop method is supported only in the standard interface.

Administrative user status is required to stop the metadata server. For more information, see "Server Administrative Privileges" in the **SAS 9.1 Metadata Server: Setup Guide**.

Example

```
<!--Stops the metadata server-->
options=''

rc=Stop(options);
```

Previous Page | Next Page | Top of Page

Overview to Hierarchy and Association Diagrams

This section provides graphical representations of the metadata types described in SAS Namespace Types. Hierarchy diagrams illustrate the relationship between subtypes and supertypes. Association diagrams describe the associations between related metadata types. The purpose of the diagrams is to help you to understand the relationships between SAS namespace types. By understanding these relationships, you can

- identify which metadata types you need to describe common application elements
- assess the data structures that will be affected when a change is made to an object in a particular hierarchy
- access metadata types by using their associated properties
- identify which metadata types can be created independently and which ones must be created in association with other types.

The diagrams are grouped according to submodel. See also "Usage Scenarios" in the **SAS 9.1 Open Metadata Interface: User's Guide**.

Understanding the Diagrams

The diagrams in this section are UML diagrams. The acronym "UML" stands for Unified Modeling Language, which is an object–oriented analysis and design language developed by the Object Management Group (OMG).

Each diagram shows only a part of the total structure. The following notes apply to all diagrams:

- Each square in the diagram represents a metadata type; the text in the square is the name of the metadata type.
- Each line (connection) indicates that two metadata types have an association. Arrows indicate the direction of the data flow.
- Hollow arrows indicate a subtype/supertype relationship in which the metadata type being pointed to is the supertype.

The following notes apply only to association diagrams:

- The text preceded by a + (plus sign) outside of each square describes the nature of the connected square's association to the square.
- The *n..n* describes the cardinality of the associations.

For more information about associations and cardinality, see "Understanding Associations" in Understanding Metadata Types.

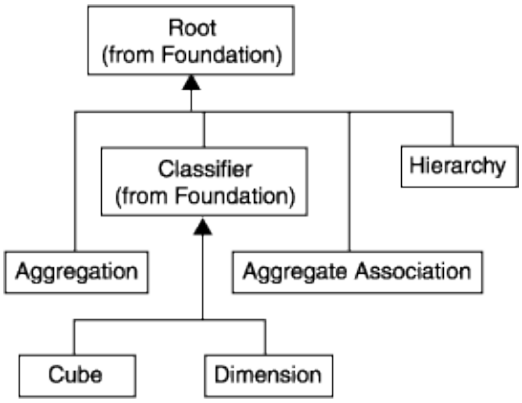
Previous Page | Next Page | Top of Page

Diagrams for Analysis Metadata Types

Analysis Hierarchy

The Analysis Hierarchy Diagram depicts the hierarchy of the metadata types that represent statistical transformations, multidimensional data sources, and OLAP information.

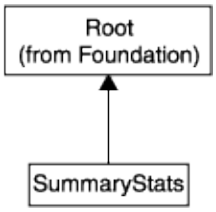
Analysis Hierarchy Diagram



Statistics Hierarchy

The Statistics Hierarchy Diagram depicts the hierarchy of the metadata types that represent summary statistics.

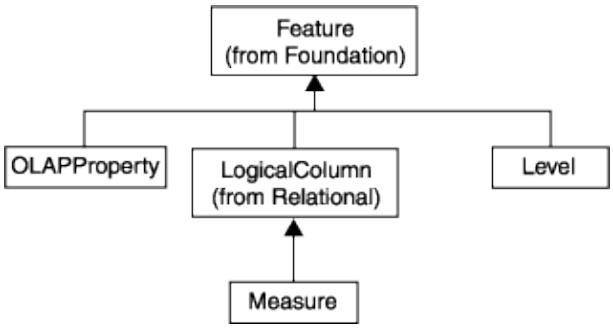
Statistics Hierarchy Diagram



Level, Measure Hierarchy

The Level and Measure Hierarchy Diagram depicts the hierarchy of the metadata types that represent OLAP information.

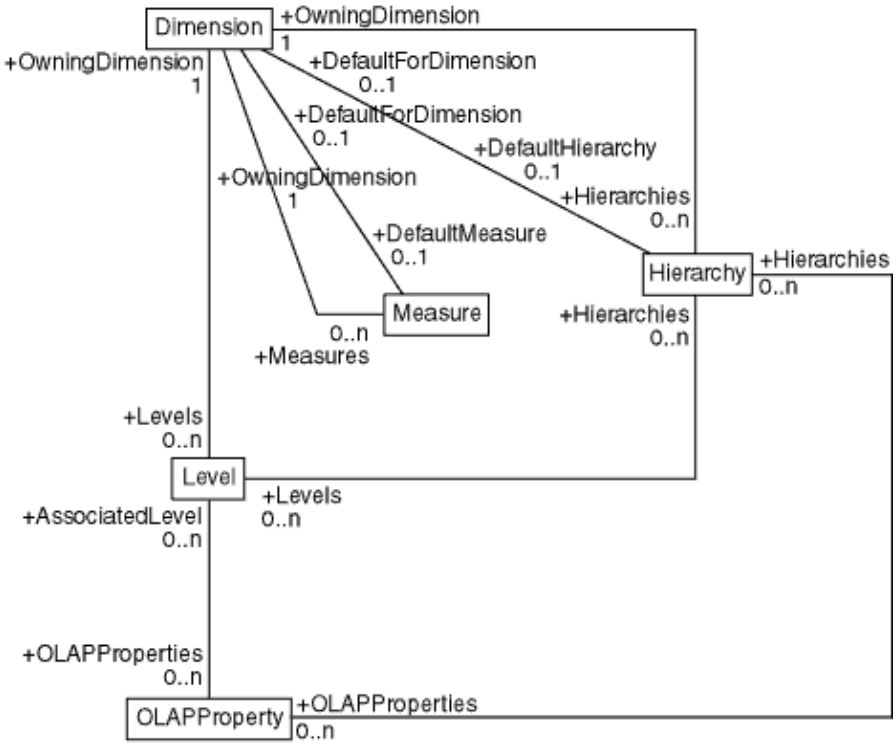
Level and Measure Hierarchy Diagram



Dimension Associations

The Dimension Associations Diagram illustrates the associations between the Dimension, Level, OLAPProperty, Measure, and Hierarchy metadata types.

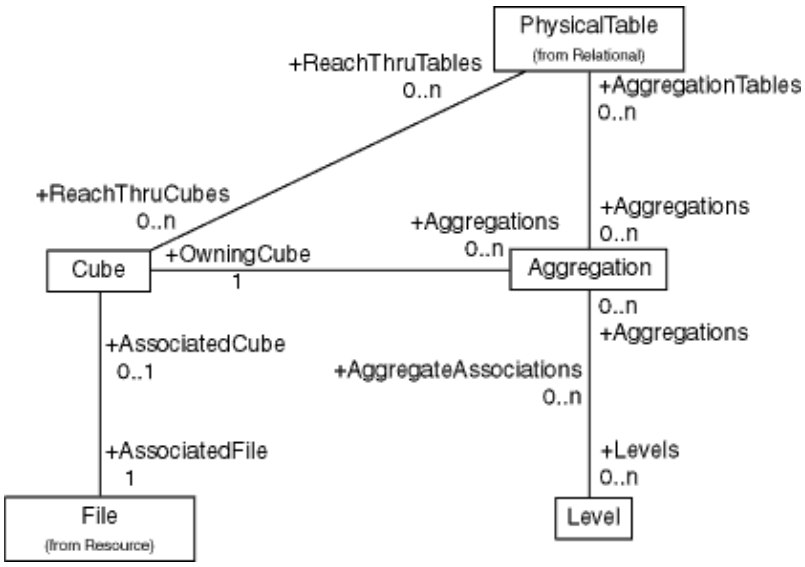
Dimension Associations Diagram



Physical Associations

The Physical Associations Diagram illustrates the associations between the Cube, File, PhysicalTable, Aggregation, and Level metadata types.

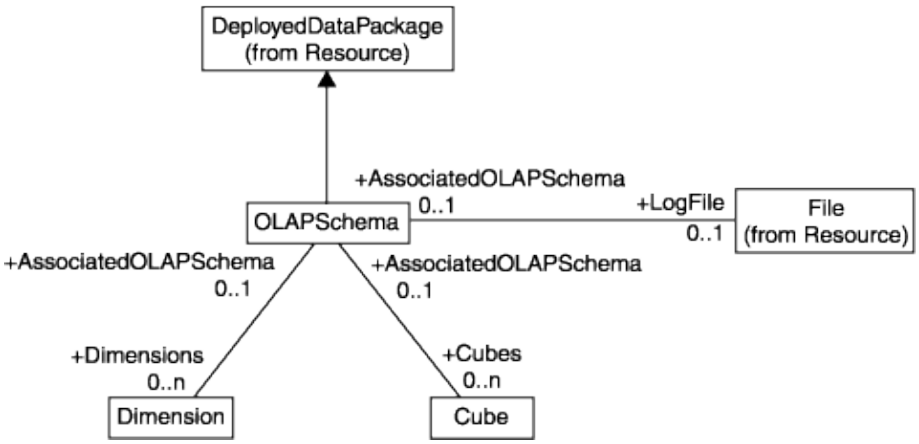
Physical Associations Diagram



OLAP Schema

The OLAP Schema Diagram illustrates the associations between the DeployedDataPackage, OLAPSchema, Dimension, File, and Cube metadata types.

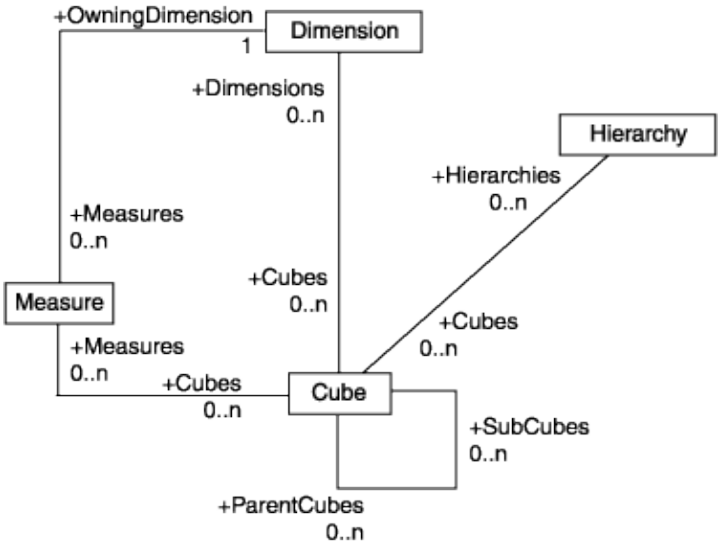
OLAP Schema Diagram



Cube Associations

The Cube Associations Diagram illustrates the associations between the Dimension, Measure, Cube, and Hierarchy metadata types.

Cube Associations Diagram



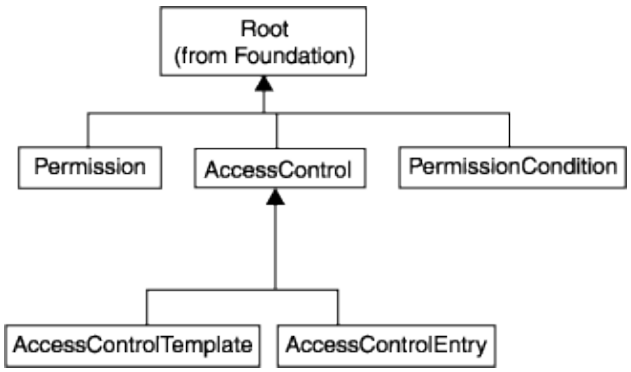
Previous Page | Next Page | Top of Page

Diagrams for Authorization Metadata Types

Authorization Hierarchy

The Authorization Hierarchy Diagram depicts the hierarchy of the metadata types defined to enable you to define access controls for metadata.

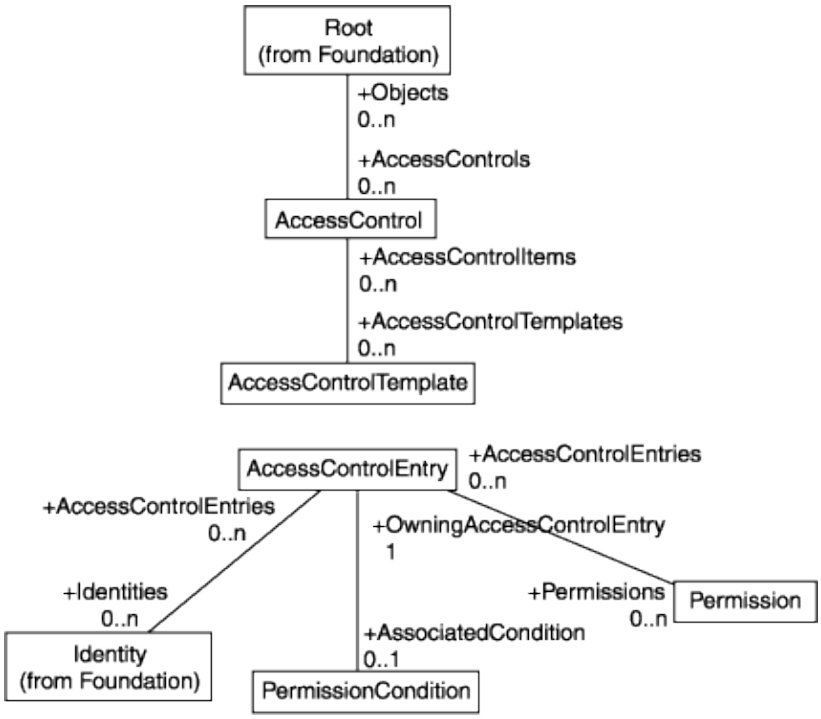
Authorization Hierarchy Diagram



Authorization Associations

The Authorization Associations Diagram illustrates the associations between the Root, AccessControl, and AccessControlTemplate metadata types, and the AccessControlEntry, Identity, Permission, and PermissionCondition metadata types.

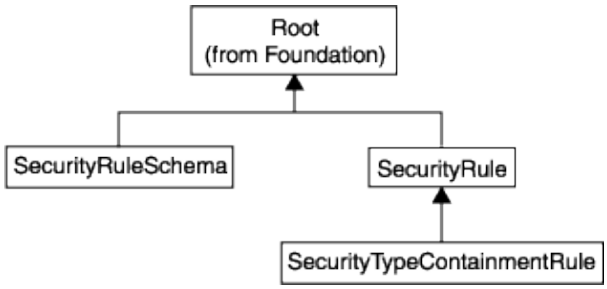
Authorization Associations Diagram



Security Rules Hierarchy

The Security Rules Hierarchy Diagram depicts the hierarchy of metadata types defined to represent security rules.

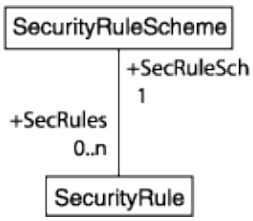
Security Rules Hierarchy Diagram



Security Rules Associations

The Security Rules Associations Diagram illustrates the associations between the SecurityRuleScheme and SecurityRule metadata types.

Security Rules Associations Diagram

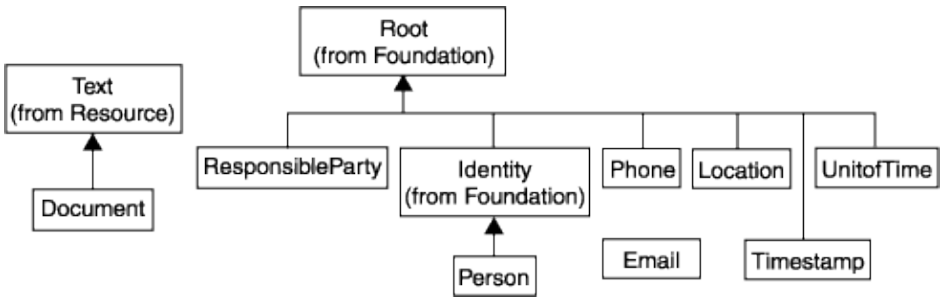


Diagrams for Business Information Metadata Types

Business Information Hierarchy

The Business Information Hierarchy Diagram depicts the hierarchy of the metadata types that describe people, their responsibilities, information about how to contact them, and business documents.

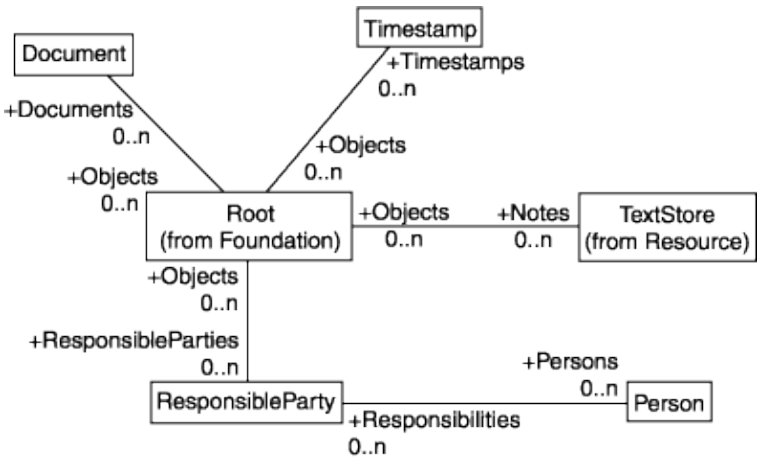
Business Information Hierarchy Diagram



Root Associations

The Root Associations Diagram illustrates the associations between the Root, Document, Timestamp, TextStore, ResponsibleParty, and Person metadata types.

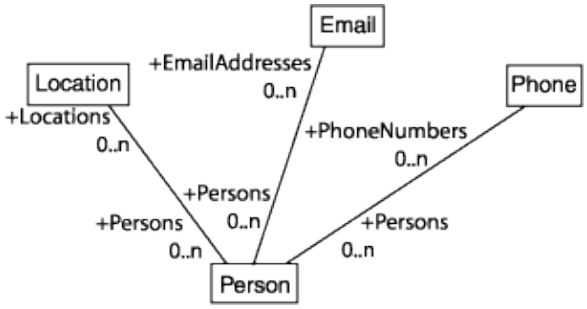
Root Associations Diagram



Person Associations

The Person Associations Diagram illustrates the associations between the Person, Location, Email, and Phone metadata types.

Person Associations Diagram



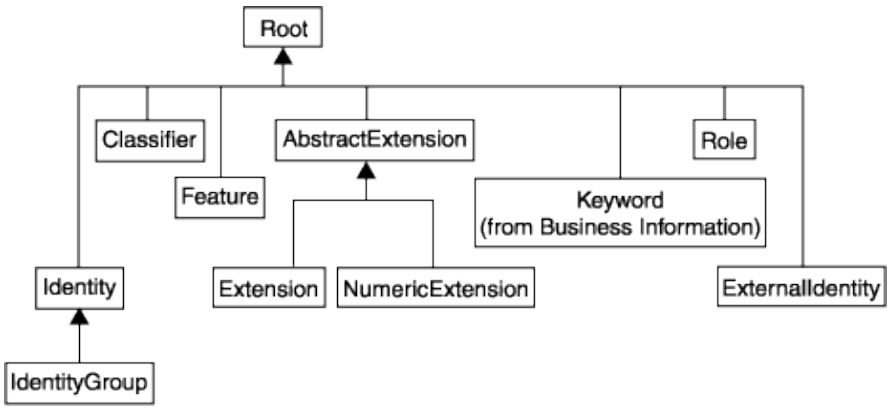
Previous Page | Next Page | Top of Page

Diagrams for Foundation Metadata Types

Foundation Hierarchy

The Foundation Hierarchy Diagram depicts the basic metadata types of the model, from which all other types are derived.

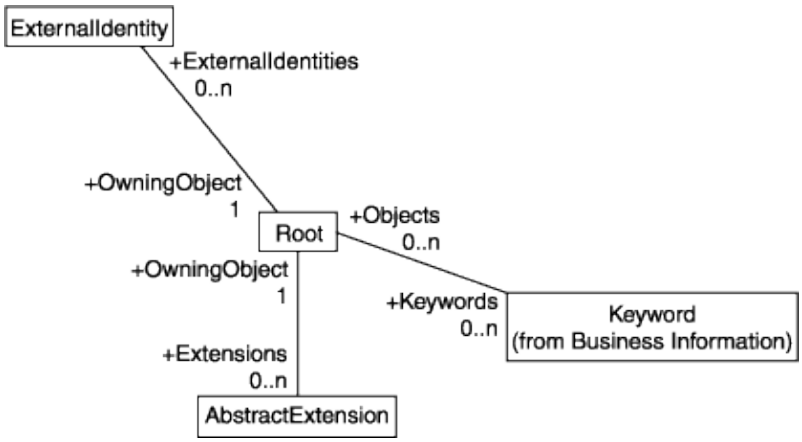
Foundation Hierarchy Diagram



Root Associations

The Root Associations Diagram illustrates the associations between the Root, ExternalIdentity, Keyword, and Abstract Extension metadata types.

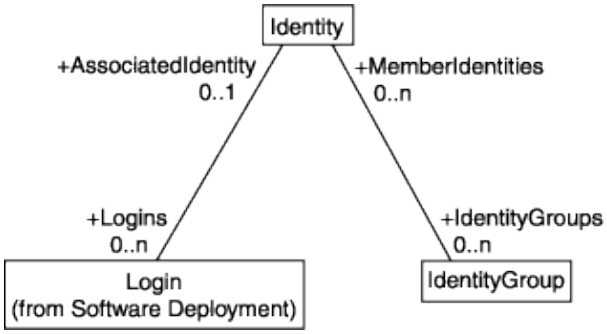
Root Associations Diagram



Identity Associations

The Identity Associations Diagram illustrates the associations between the Identity, Login, and IdentityGroup metadata types.

Identity Associations Diagram



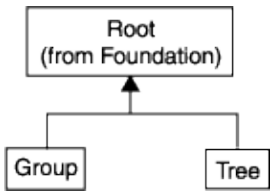
Previous Page | Next Page | Top of Page

Diagrams for Grouping Metadata Types

Grouping Hierarchy

The Grouping Hierarchy Diagram depicts the hierarchy of the metadata types defined to represent collections of data.

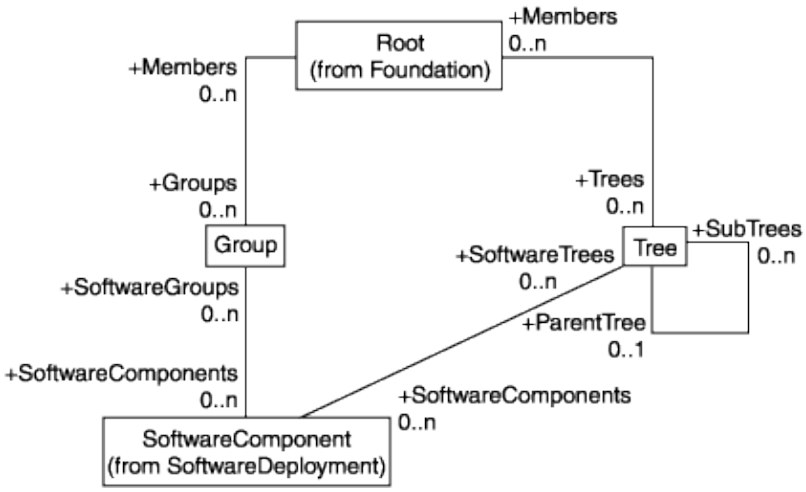
Grouping Hierarchy Diagram



Grouping Associations

The Grouping Associations Diagram illustrates the associations between the Root, Group, Tree, and SoftwareComponent metadata types.

Grouping Associations Diagram



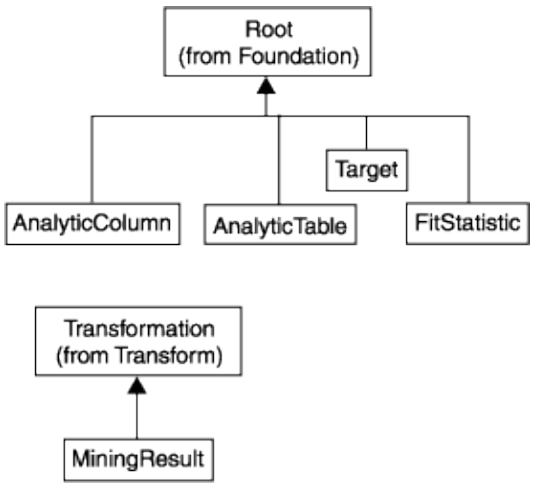
Previous Page | Next Page | Top of Page

Diagrams for Mining Metadata Types

Mining Hierarchy

The Mining Hierarchy Diagram depicts the hierarchy of the metadata types defined to represent analytic transformations.

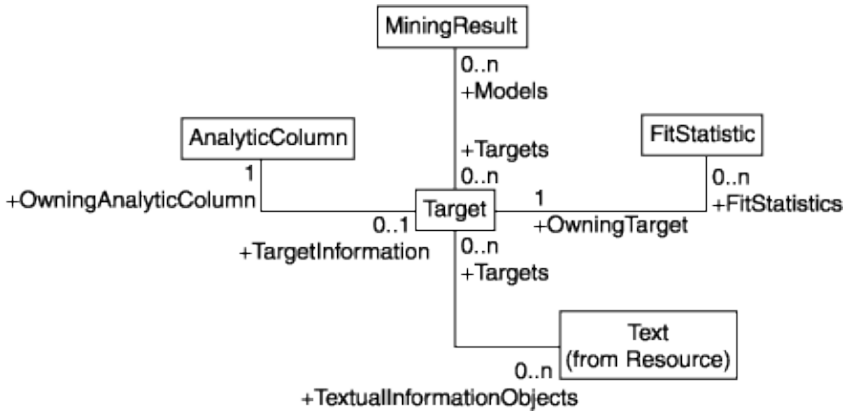
Mining Hierarchy Diagram



Target Associations

The Target Associations Diagram illustrates the associations between the AnalyticColumn, Target, MiningResult, Text, and FitStatistic metadata types.

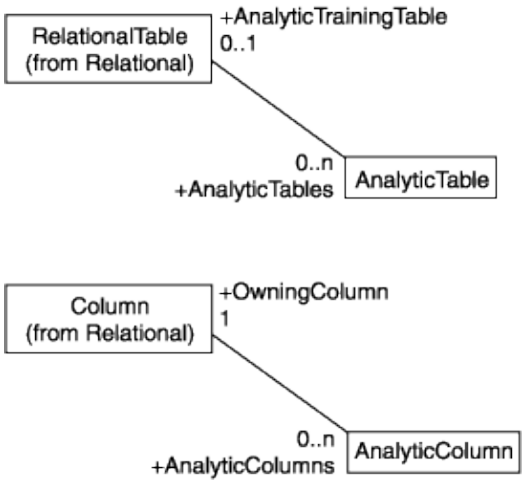
Target Associations Diagram



Analytic Table and Column Associations

The Analytic Table and Column Associations Diagram illustrates the associations between the RelationalTable and AnalyticTable metadata types, and the Column and AnalyticColumn metadata types.

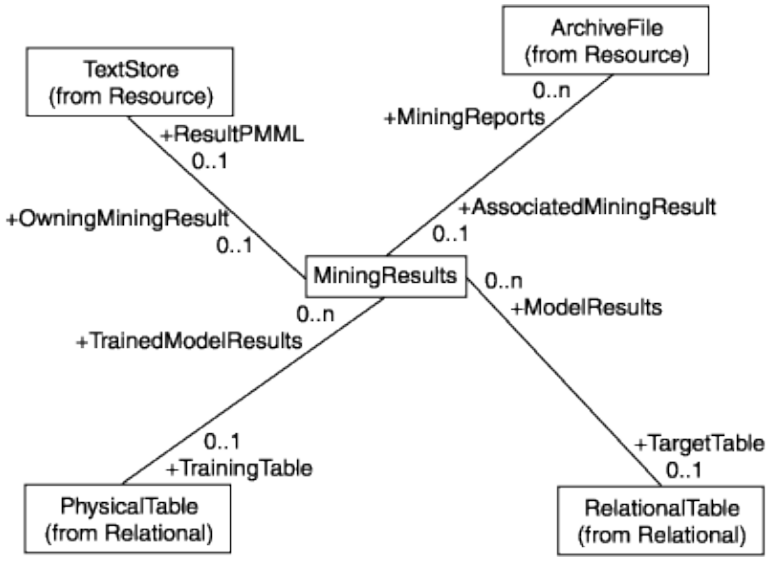
Analytic Table and Column Associations Diagram



ModelResult Associations

The ModelResult Associations Diagram illustrates the associations between the MiningResult, TextStore, ArchiveFile, RelationalTable, and PhysicalTable metadata types.

ModelResult Associations Diagram



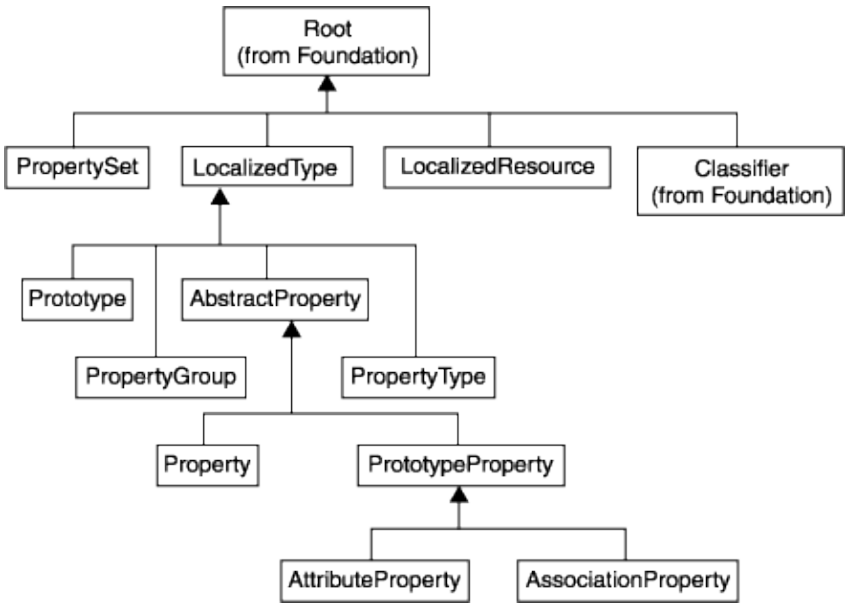
SAS 9.1 Open Metadata Interface: Reference

Diagrams for Property Metadata Types

Property Hierarchy

The Property Hierarchy Diagram depicts the hierarchy of the metadata types defined to represent options and extended properties in the SAS Metadata Model.

Property Hierarchy Diagram



Property Associations

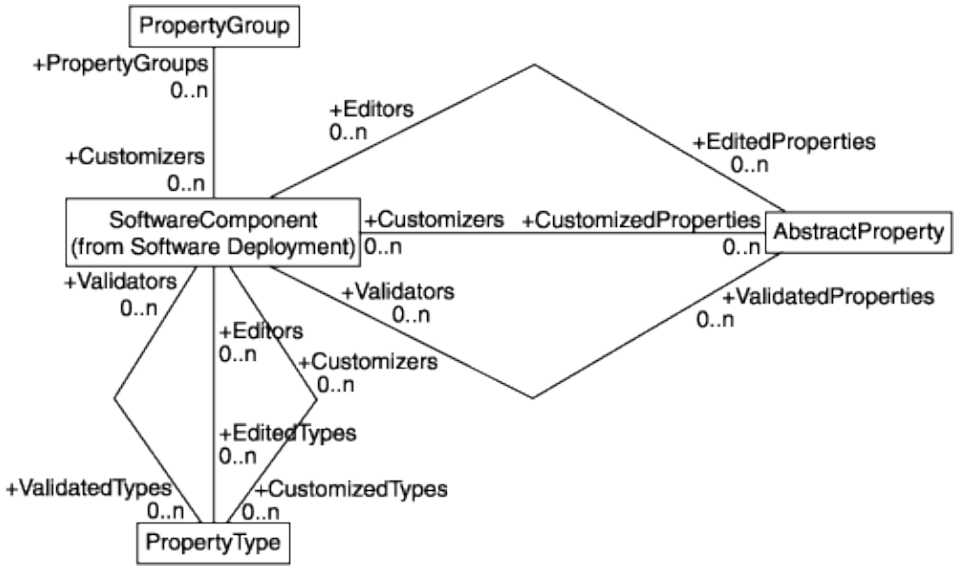
The Property Associations Diagram illustrates the associations between the Root, PropertySet, Property, PropertyType, AttributeProperty, PropertyGroup, AbstractProperty, and Text metadata types.

Property Associations Diagram

Configuration Associations

The Configuration Associations Diagram illustrates the associations between the PropertyGroup, SoftwareComponent, AbstractProperty, and PropertyType metadata types.

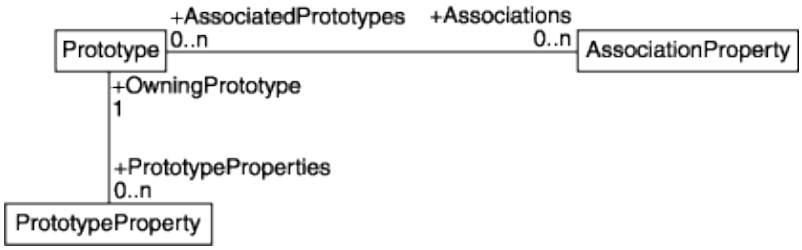
Configuration Associations Diagram



Prototype Associations

The Prototype Associations Diagram illustrates the associations between the Prototype, PrototypeProperty, and AssociationProperty metadata types.

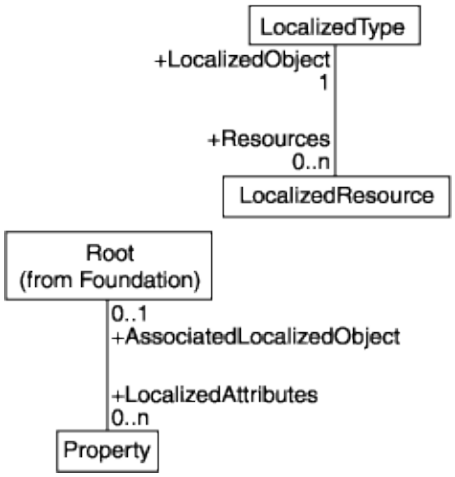
Prototype Associations Diagram



Locale Associations

The Locale Associations Diagram illustrates the associations between the LocalizedType and LocalizedResource metadata types, and the Root and Property metadata types.

Locale Associations Diagram



PropertyType Array Associations

The PropertyType Array Associations Diagram illustrates the associations between PropertyTypes.

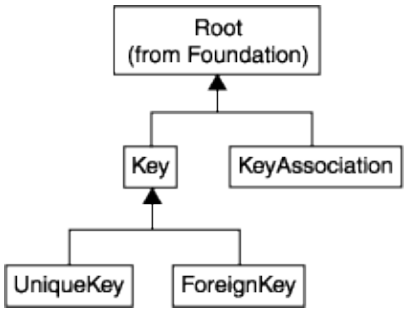
PropertyType Array Associations Diagram

Diagrams for Relational Metadata Types

Key Hierarchy

The Key Hierarchy Diagram depicts the hierarchy of the metadata types that represent relational keys.

Key Hierarchy Diagram



Column Hierarchy

The Column Hierarchy Diagram depicts the type hierarchy of the metadata types that represent relational columns.

Column Hierarchy Diagram

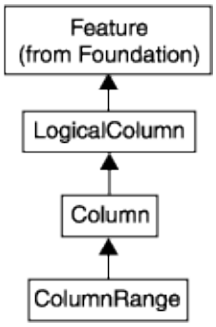
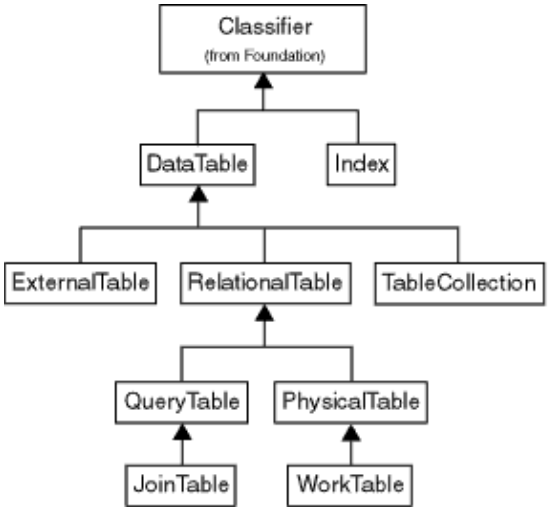


Table Hierarchy

The Table Hierarchy Diagram depicts the hierarchy of the metadata types that represent tables.

Table Hierarchy Diagram

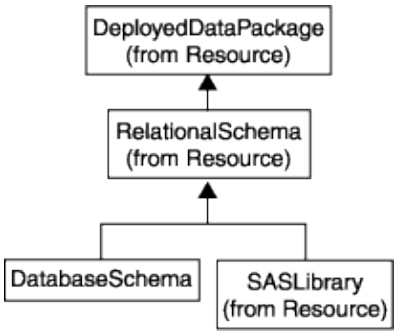


DataTable is an abstract metadata type that models the properties for three classes of subtypes: ExternalTable, RelationalTable, and TableCollection. RelationalTable has as subtypes QueryTable, PhysicalTable, JoinTable, and WorkTable, which also inherit from each other.

DeployedDataPackage Hierarchy

The DeployedDataPackage Hierarchy Diagram depicts the hierarchy of the metadata types that represent a relational schema.

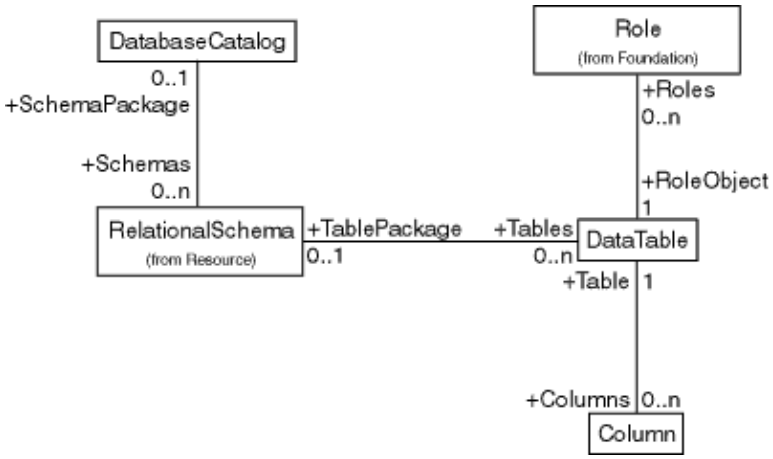
DeployedDataPackage Hierarchy Diagram



Schema, Table, Role, and Column Associations

The Schema, Table, Role, and Column Associations Diagram illustrates the associations between the DatabaseCatalog, RelationalSchema, DataTable, Role, and Column metadata types.

Schema, Table, Role, and Column Associations Diagram



Table, Password, and Index Associations

The Table, Password, and Index Associations Diagram illustrates the associations between the PhysicalTable, SASPassword, Index, and Column metadata types.

Table, Password, and Index Associations Diagram

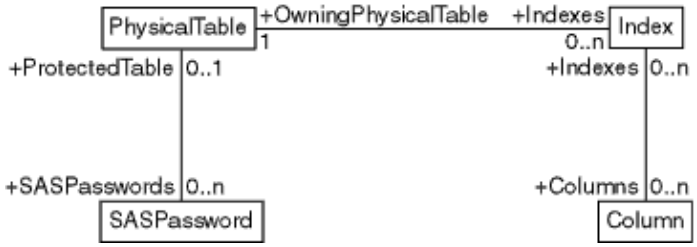
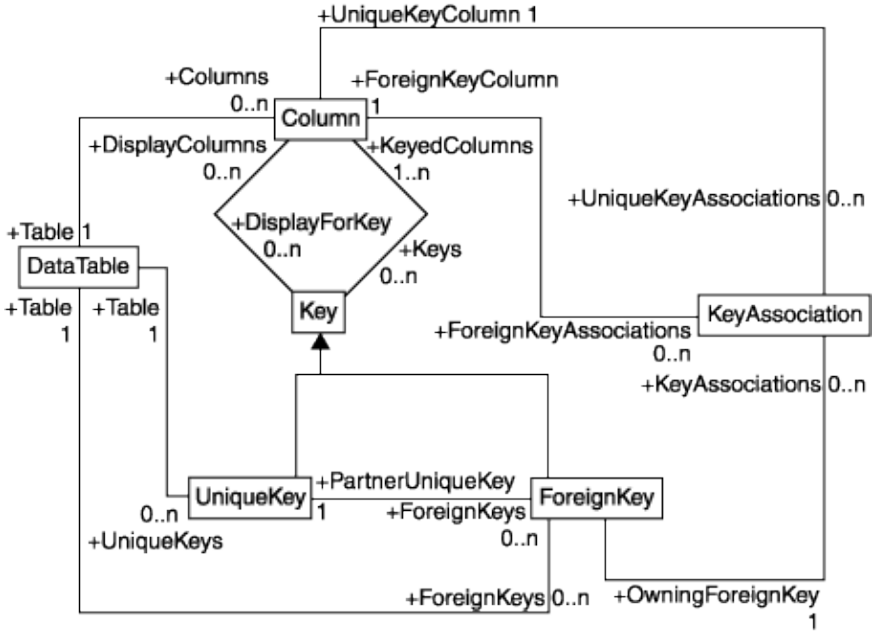


Table and Key Associations

The Table and Key Associations Diagram illustrates the associations between DataFile, Column, Key and KeyAssociation metadata types.

Table and Key Associations Diagram



TKTS Data Source Name Associations

The TKTS Data Source Name Associations Diagram illustrates the associations between RelationalSchema, DataSourceName, Login, and Connection metadata types.

TKTS Data Source Name Associations Diagram

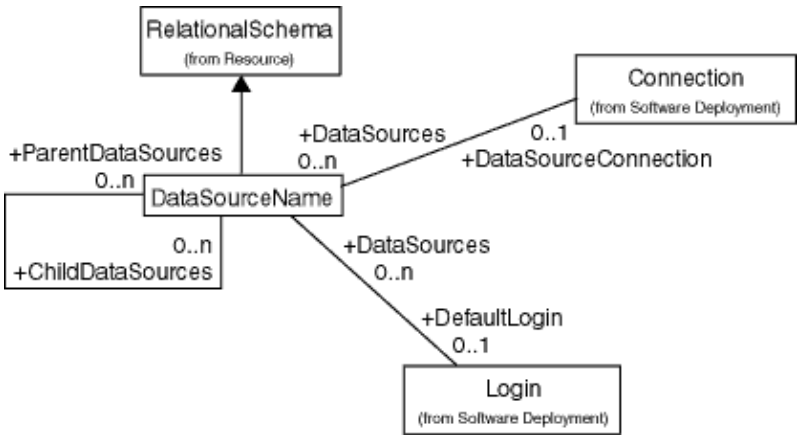
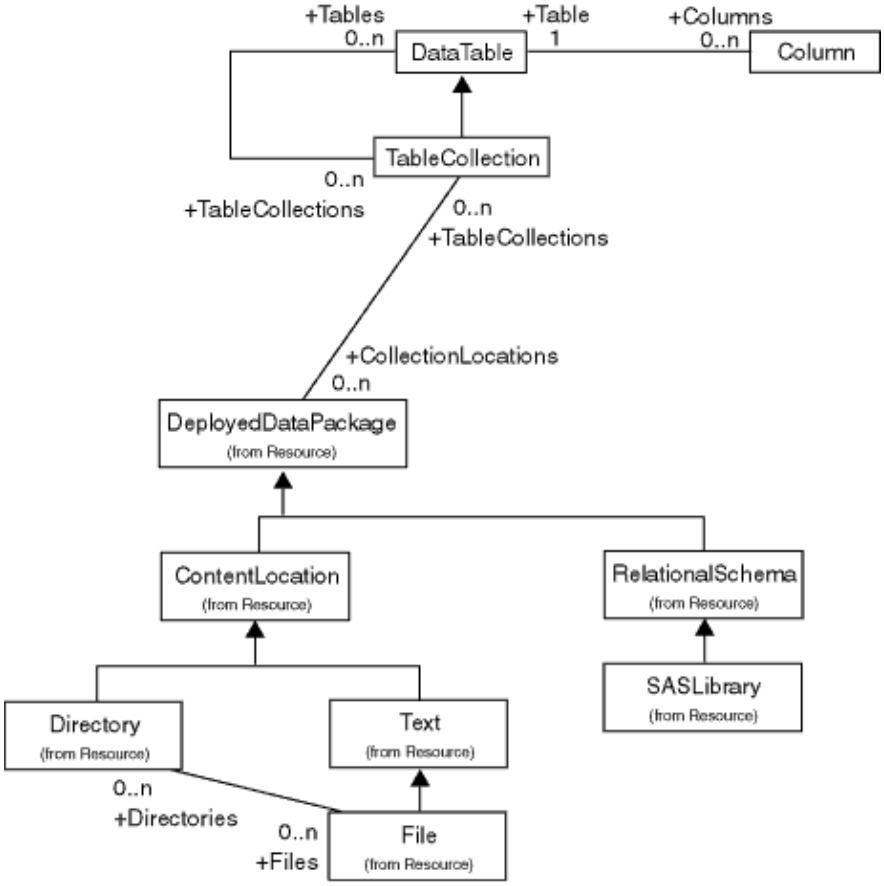


Table Collection Associations

The Table Collection Associations Diagram illustrates the associations between the DataTable, Column, and TableCollection metadata types, and the TableCollection, DeployedDataPackage, ContentLocation, Directory, Text, File, RelationalSchema, and SASLibrary metadata types.

Table Collection Associations Diagram



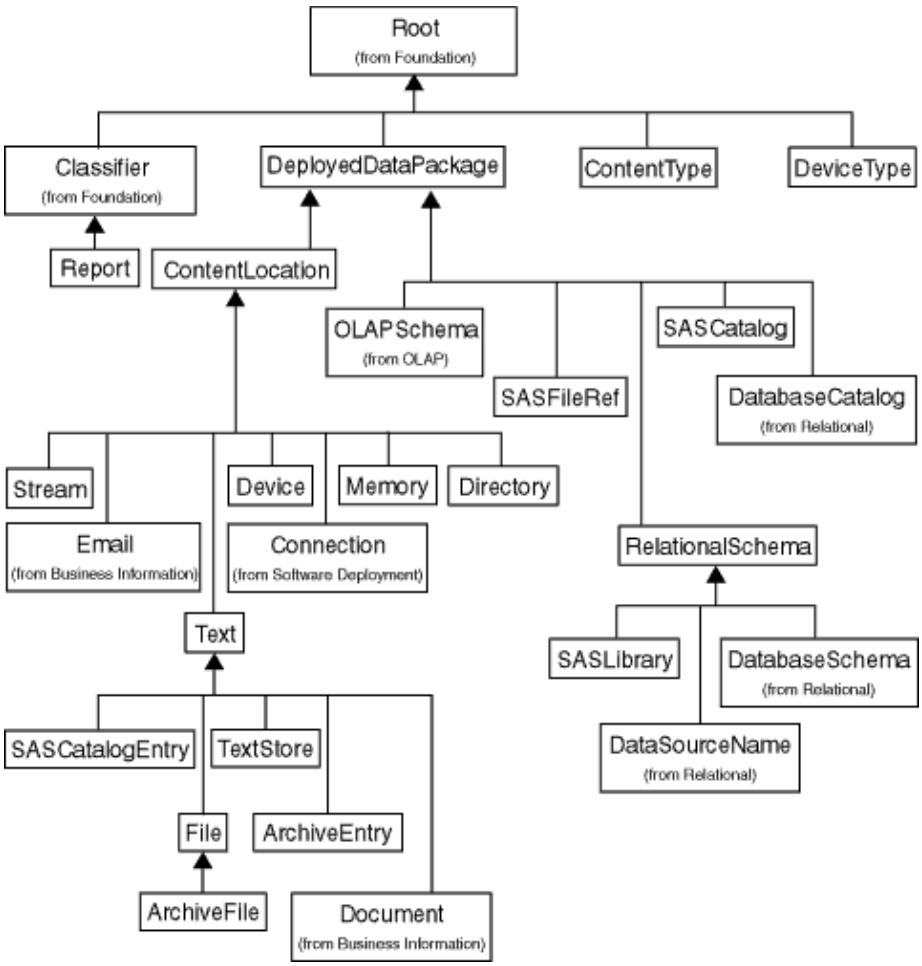
Previous Page | Next Page | Top of Page

Diagrams for Resource Metadata Types

Resource Hierarchy

The Resource Hierarchy Diagram depicts the hierarchy of the metadata types defined to represent data resources.

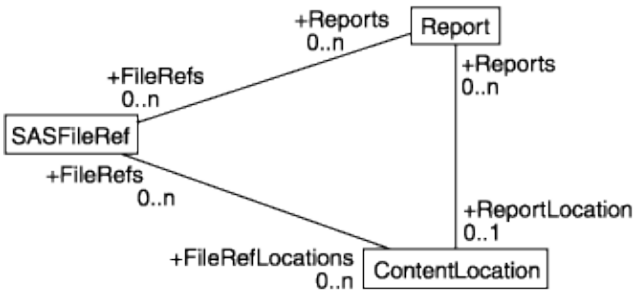
Resource Hierarchy Diagram



Report, SASFileRef Associations

The Report and SASFileRef Associations Diagram illustrates the associations between the Report, SASFileRef, and ContentLocation metadata types.

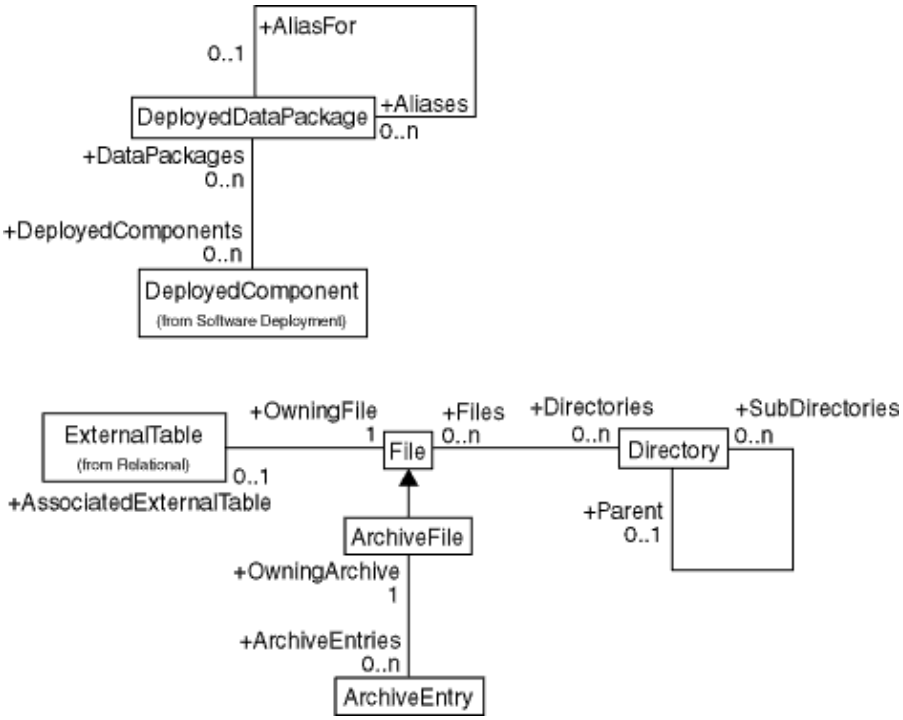
Report and SASFileRef Associations Diagram



DeployedDataPackage, File Associations

The DeployedDataPackage and File Associations Diagram illustrates the associations between the DeployedDataPackage and DeployedComponent metadata types and the File, ExternalTable, ArchiveFile, ArchiveEntry, and Directory metadata types.

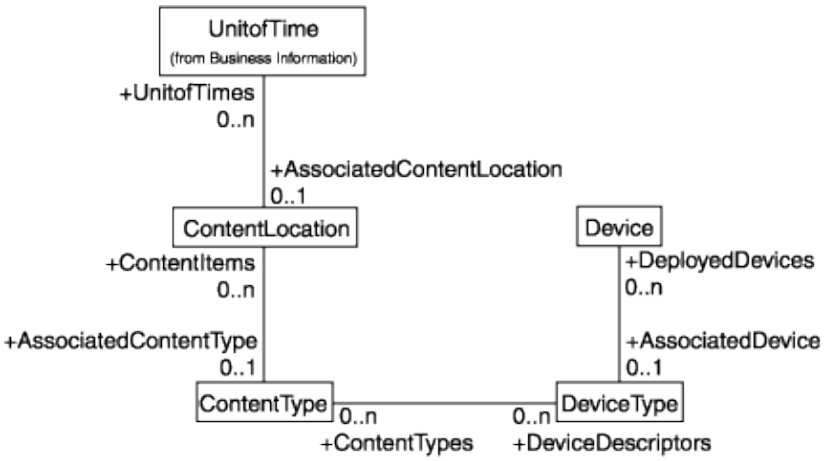
DeployedDataPackage and File Associations Diagram



Resource Content Type, Devices Associations

The Resource Content Type, Devices Associations Diagram illustrates the associations between the UnitofTime, ContentLocation, ContentType, DeviceType, and Device metadata types.

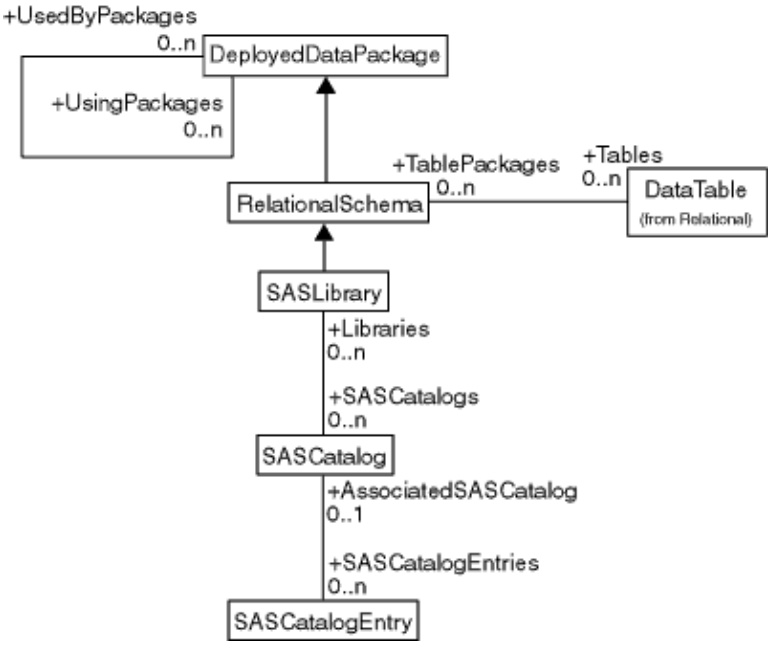
Resource Content Type, Devices Associations Diagram



SASLibrary, SASCatalog Associations

The SASLibrary, SASCatalog Associations Diagram illustrates the associations between the DeployedDataPackage, RelationalSchema, DataTable, SASLibrary, SASCatalog, and SASCatalogEntry metadata types.

SASLibrary, SASCatalog Associations Diagram



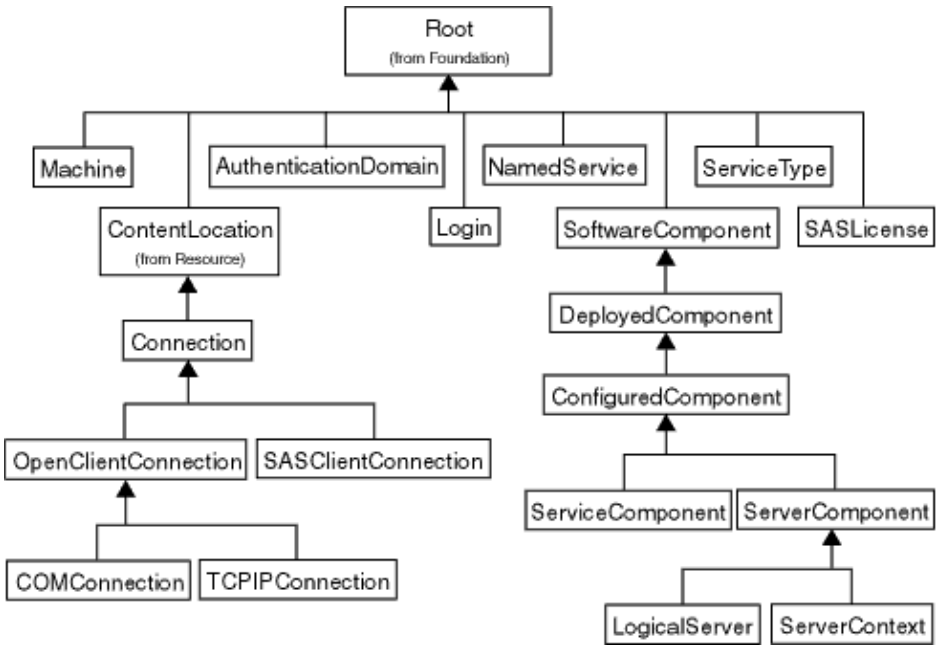
Previous Page | Next Page | Top of Page

Diagrams for Software Deployment Metadata Types

Software Deployment Hierarchy

The Software Deployment Hierarchy Diagram depicts the hierarchy of the metadata types defined to describe software, servers, and connection information.

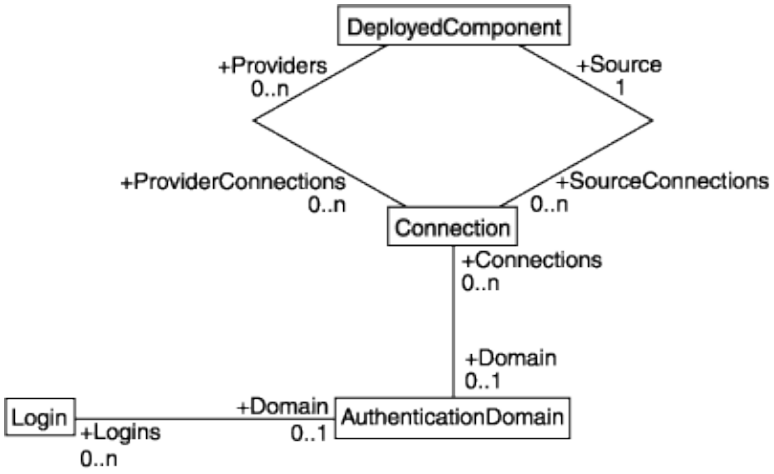
Software Deployment Hierarchy Diagram



Login, DeployedComponent Associations

The Login and DeployedComponent Associations Diagram illustrates the associations between the Login, DeployedComponent, Connection, and AuthenticationDomain metadata types.

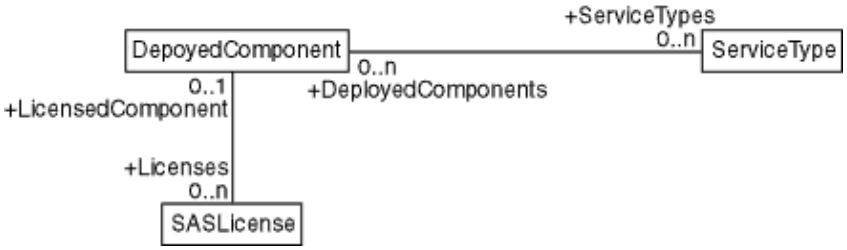
Login and DeployedComponent Associations Diagram



DeployedComponent, ServiceType Association

The DeployedComponent, ServiceType Association Diagram illustrates the associations between the DeployedComponent and SASLicense metadata types, and the DeployedComponent and ServiceType metadata types.

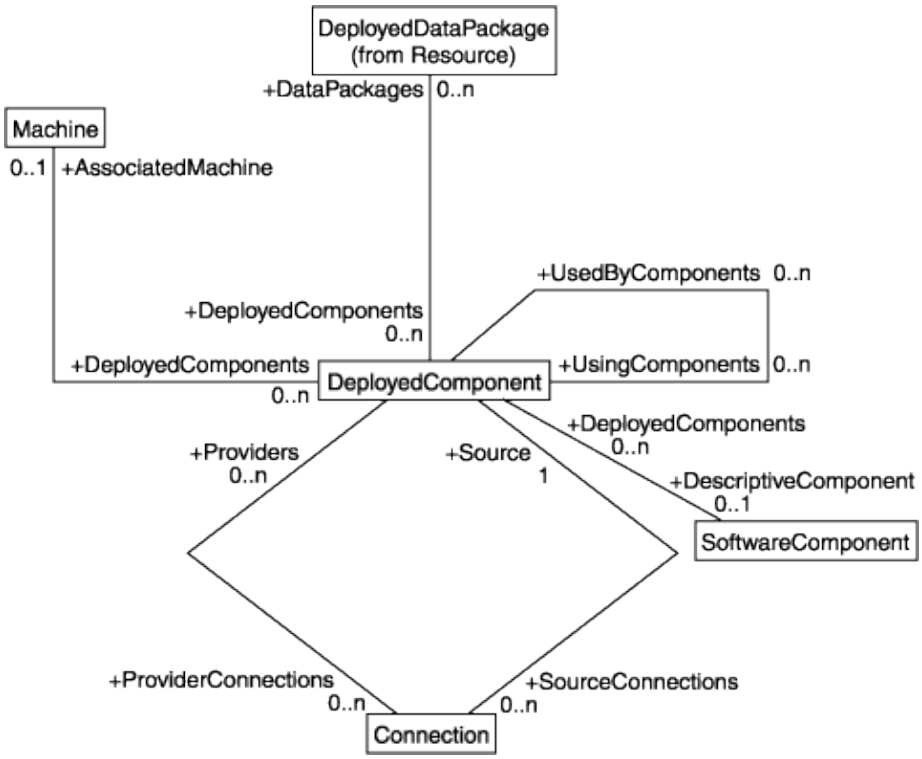
DeployedComponent, ServiceType Association Diagram



DeployedComponent Associations

The DeployedComponent Associations Diagram illustrates the associations between the DeployedComponent, Machine, DeployedDataPackage, SoftwareComponent, and Connection metadata types.

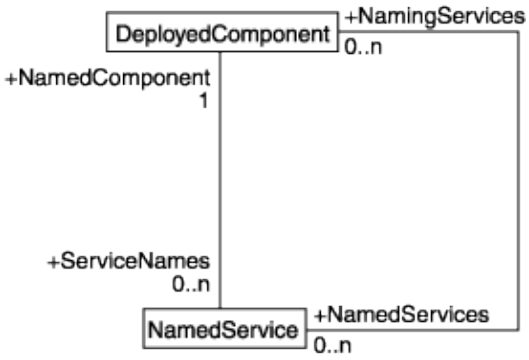
DeployedComponent Associations Diagram



DeployedComponent, NamedService Association

The DeployedComponent and NamedService Association Diagram illustrates the associations between the DeployedComponent and NamedService metadata types.

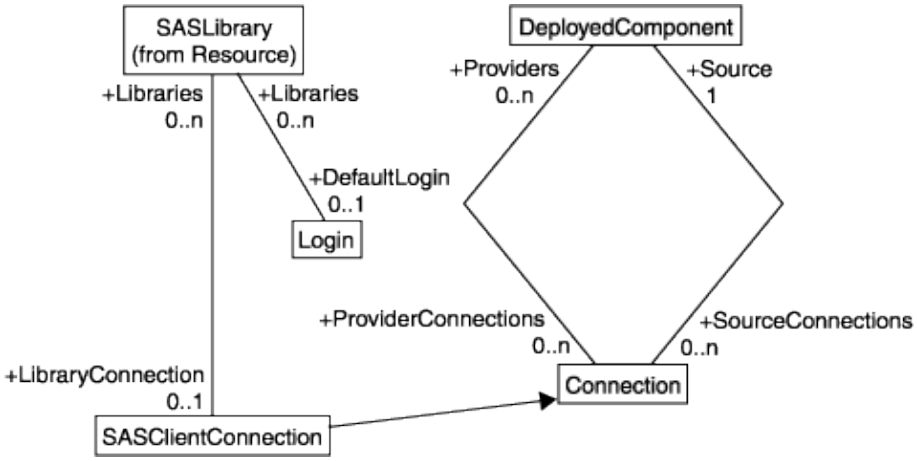
DeployedComponent and NamedService Association Diagram



SASLibrary, Database Associations

The SASLibrary and Database Associations Diagram illustrates the associations between the SASLibrary, SASClientConnection, Login, Connection, and DeployedComponent metadata types.

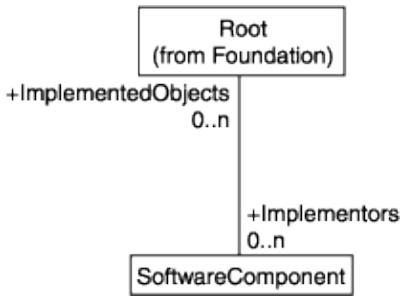
SASLibrary and Database Associations Diagram



SoftwareComponent, Root Association

The SoftwareComponent and Root Associations Diagram illustrates the associations between the SoftwareComponent and Root metadata types.

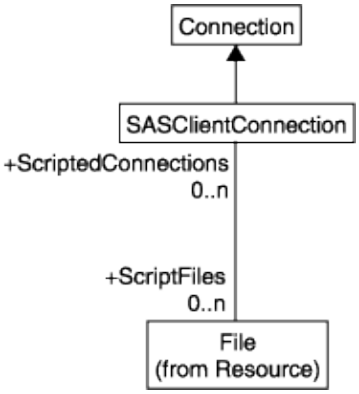
SoftwareComponent and Root Associations Diagram



Connection, Script File Associations

The Connection and Script File Associations Diagram illustrates the associations between the Connection, SASClientConnection, and File metadata types.

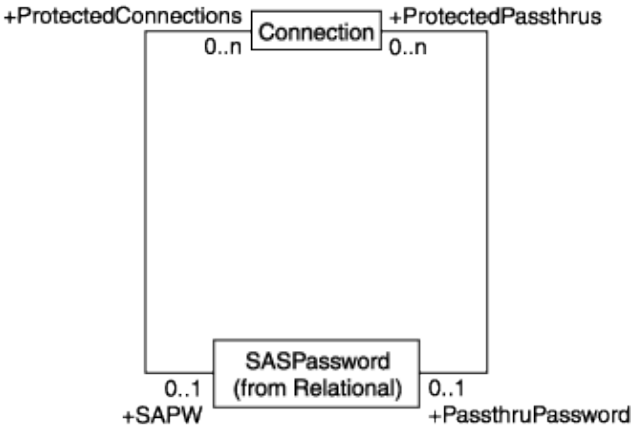
Connection and Script File Associations Diagram



Connection, SASPassword Associations

The Connection and SASPassword Associations Diagram illustrates the associations between the Connection and SASPassword metadata types.

Login and DeployedComponent Associations Diagram



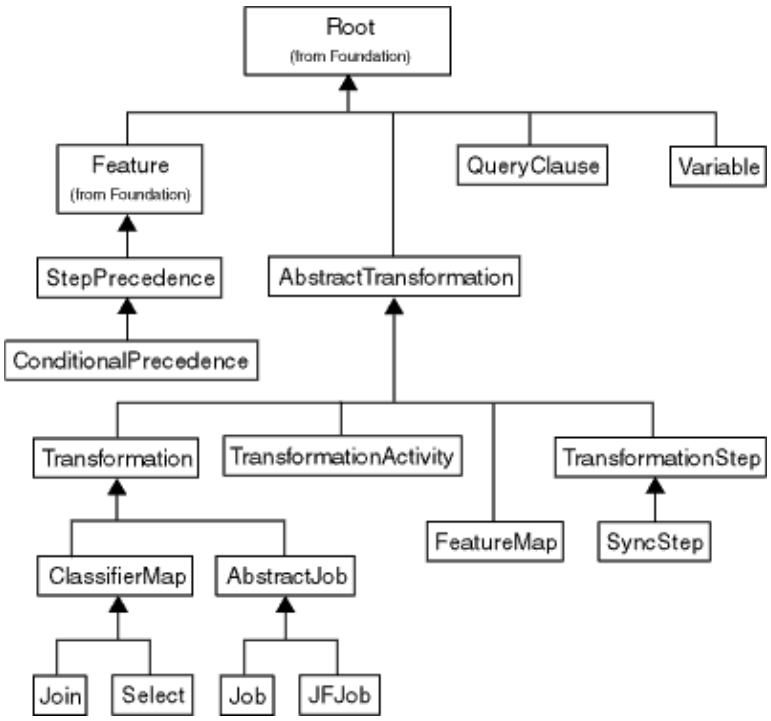
Previous Page | Next Page | Top of Page

Diagrams for Transformation Metadata Types

Transformation Hierarchy

The Transformation Hierarchy Diagram depicts the hierarchy of the metadata types defined to describe transformation of data.

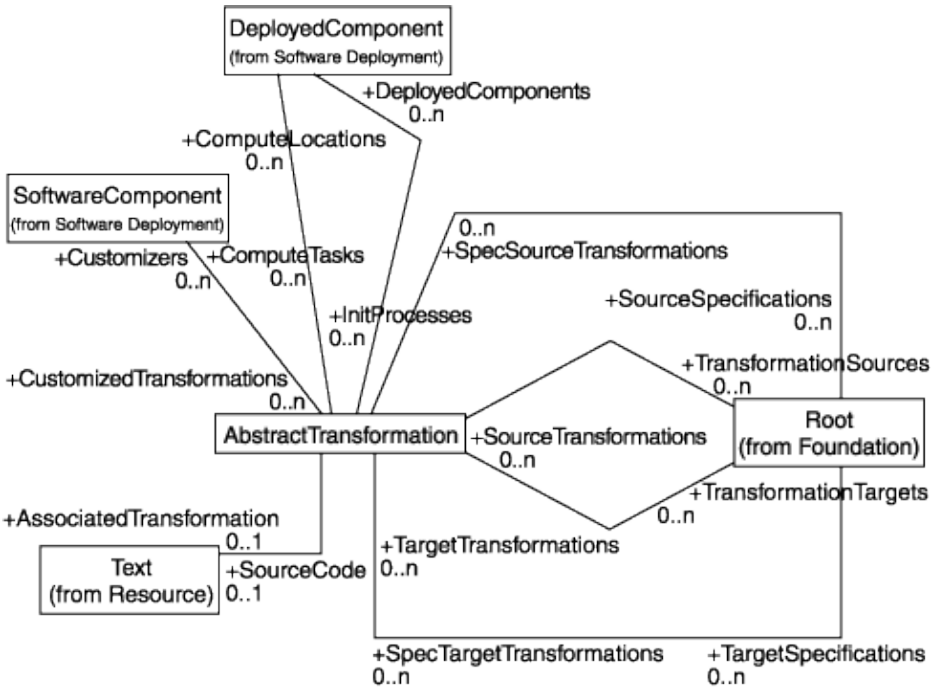
Transformation Hierarchy Diagram



Root, Transformation Associations

The Root and Transformation Associations Diagram illustrates the associations between the Root, AbstractTransformation, DeployedComponent, SoftwareComponent, and Text metadata types.

Root and Transformation Associations Diagram



Transformation Associations Overview

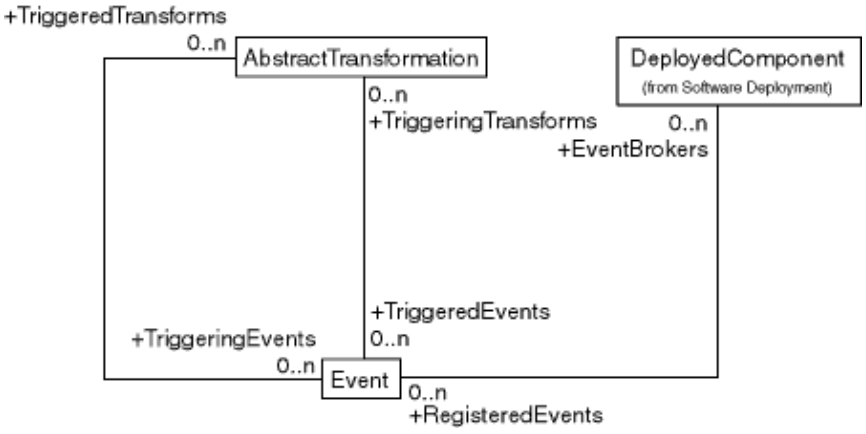
The Transformation Associations Overview Diagram illustrates the associations between the TransformationActivity, TransformationStep, StepPrecedence, Transformation, ClassifierMap, Classifier, FeatureMap, and Feature metadata types.

Transformation Associations Overview Diagram

Event Associations

The Event Associations Diagram illustrates the associations between the AbstractTransformation, DeployedComponent, and Event metadata types.

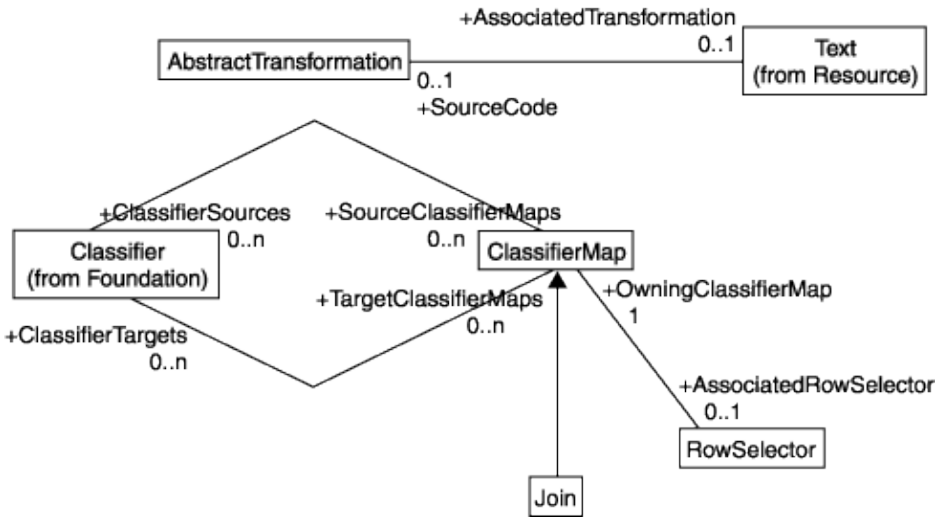
Event Associations Diagram



ClassifierMap Associations

The ClassifierMap Associations Diagram illustrates the associations between the AbstractTransformation and Text, and Classifier, ClassifierMap, RowSelector, and Join metadata types.

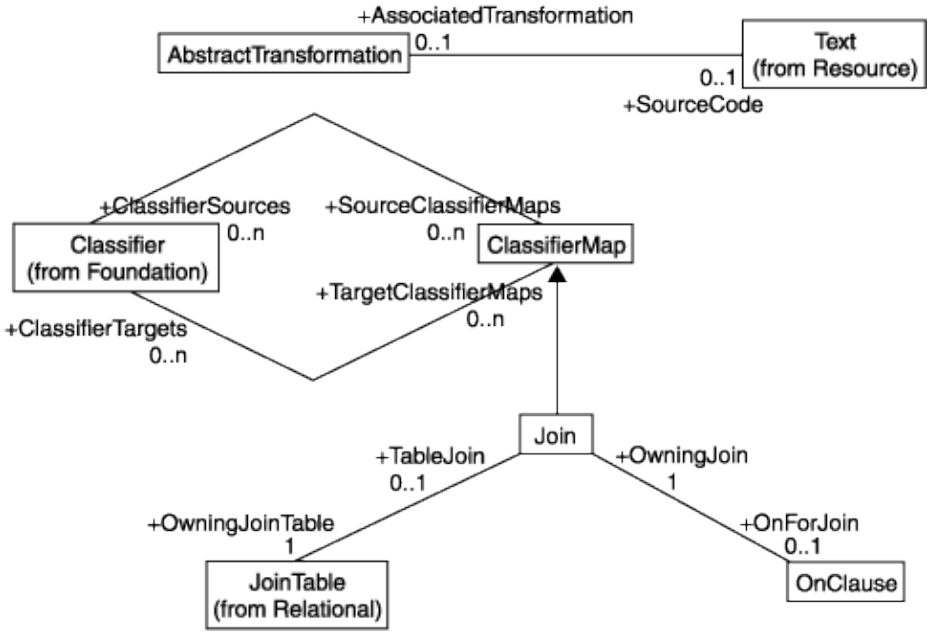
ClassifierMap Associations Diagram



Join Associations

The Join Associations Diagram illustrates the associations between the AbstractTransformation and Text, and Classifier, ClassifierMap, Join, JoinTable, and OnClause metadata types.

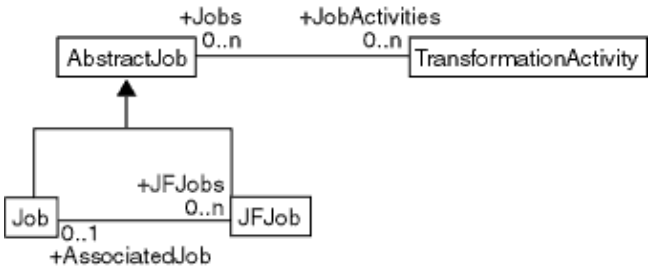
Join Associations Diagram



Job Associations

The Job Associations Diagram illustrates the associations between the AbstractJob, TransformationActivity, Job, and JFJob metadata types.

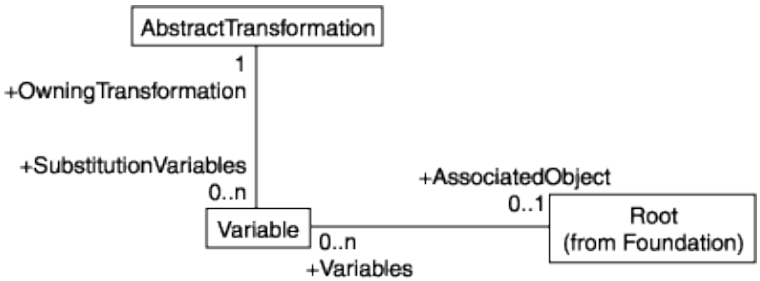
Job Associations Diagram



Variable Associations

The Variable Associations Diagram illustrates the associations between the Root, Variable, and AbstractTransformation metadata types.

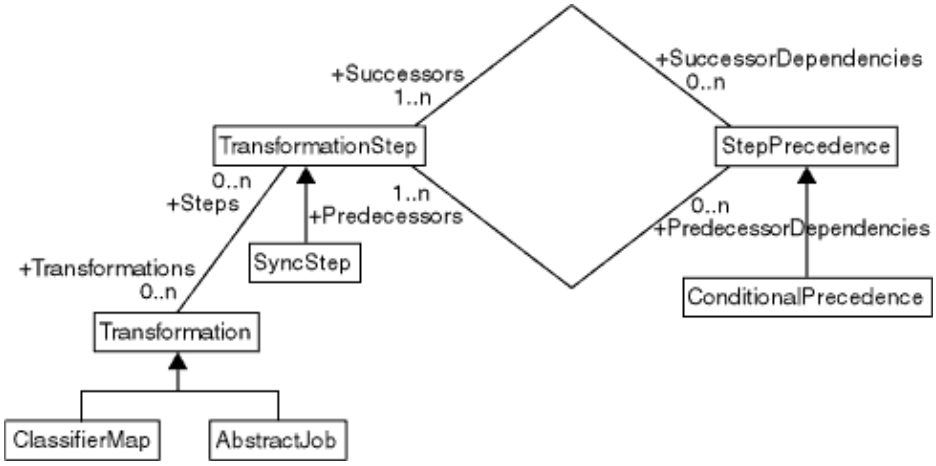
Variable Associations Diagram



Workflow Associations

The Workflow Associations Diagram illustrates the associations between the TransformationStep, SyncStep, Transformation, ClassifierMap, and AbstractJob metadata types and the TransformationStep, StepPrecedence, and ConditionalPrecedence metadata types.

Workflow Associations Diagram



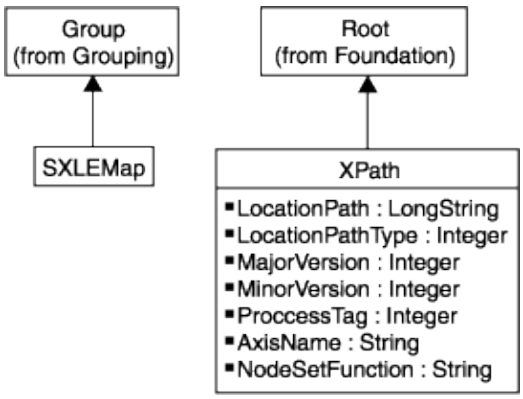
Previous Page | Next Page | Top of Page

Diagrams for XML Metadata Types

XML Hierarchy

The XML Hierarchy Diagram depicts the hierarchy of the metadata types defined to represent XML structures.

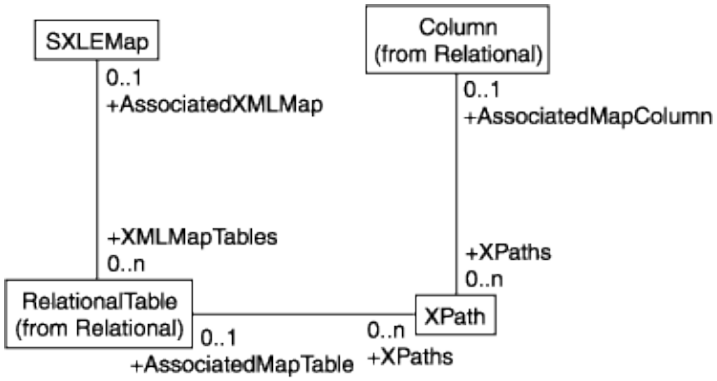
XML Hierarchy Diagram



XML Associations

The XML Associations Diagram illustrates the associations between the SXLEMap, RelationalTable, XPath, and Column metadata types.

XML Associations Diagram



Previous Page | Next Page | Top of Page

Overview to Program–Specific Method Examples

This section contains Java, Visual Basic, and Visual C++ examples of the method calls described in IOMI Class. The IOMI samples are code fragments that can be used in conjunction with the sample IOMI connection code presented in Sample Java IOMI Client, Sample Visual Basic OMI Client, and Sample Visual C++ IOMI Client to build a SAS Open Metadata Interface client. The examples show how to issue each method call using both the standard interface and the DoRequest method.

Previous Page | Next Page | Top of Page

Program–Specific AddMetadata Examples

The following code fragments create a PhysicalTable object using first the standard interface and then using the DoRequest method.

- Java Example of an AddMetadata Call
- Visual Basic Example of an AddMetadata Call
- Visual C++ Example of an AddMetadata Call

Java Example of an AddMetadata Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void addMetadata() {

    int returnFromOMI;
    String inMetadata;
    String Reposid;
    StringHolder outMetadata;
    String Namespace;
    int Flag;
    String Options;

    inMetadata="<PhysicalTable Name=\"New Table\" Desc=\"New Table added through API\"/>";
    Reposid="A0000001.A2345678";
    outMetadata = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 268435456;
    Options = "";

    try {
       returnFromOMI = connection.AddMetadata(inMetadata, Reposid, outMetadata,
          Namespace, Flag, Options);
    }

    catch (com.sas.iom.SASIOMDefs.GenericError e)
    {
       System.out.println(e);
    }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {
```

```
    int returnFromOMI;
    String inputXML;
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<AddMetadata>" +
               "<Metadata>" +
               "<PhysicalTable Name=\"New Table\" Desc=\"New Table added through API\"/>" +
               "</Metadata>" +
               "<Reposid>A0000001.A2345678</Reposid>" +
               "<NS>SAS</NS>" +
               "<Flags>268435456</Flags>" +
               "<Options/>" +
               "</AddMetadata>";
    try {
       returnFromOMI = connection.DoRequest(inputXML, outputXML);
    }

    catch (com.sas.iom.SASIOMDefs.GenericError e)
    {
       System.out.println(e);
    }

 }
```

Visual Basic Example of an AddMetadata Call

Standard Interface

```
'****************************************
'****************************************
VB Example of Standard Interface
'****************************************
'****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inMetadata    As String
   Dim Reposid       As String
   Dim outMetadata   As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   inMetadata="<PhysicalTable Name=""New Table"" Desc=""New Table added through API""/>"
   Reposid="A0000001.A2345678"
   Namespace="SAS"
   Flag = 268435456
   Options = ""

   returnFromOMI = obOMI.AddMetadata(inMetadata, Reposid, outMetadata, Namespace,
      Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()

  Dim returnFromOMI As Long
  Dim inputXML      As String
  Dim outputXML     As String

  inputXML = "<AddMetadata>" + _
             "<Metadata>" + _
             "<PhysicalTable Name=""New Table"" Desc=""New Table added through API""/>" + _
             "</Metadata>" + _
             "<Reposid>A0000001.A2345678</Reposid>" + _
             "<NS>SAS</NS>" + _
             "<Flags>268435456</Flags>" + _
             "<Options/>" + _
             "</AddMetadata>"

  returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of an AddMetadata Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString inMetadataStr("<PhysicalTable Name=\"New Table\" Desc=\"New Table added
       through API\"/>");
    BSTR inMetadata = inMetadataStr.AllocSysString();

    CString ReposidStr("A0000001.A2345678");
    BSTR Reposid = ReposidStr.AllocSysString();

    CString outMetadataStr("");
    BSTR outMetadata = outMetadataStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 268435456;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();
```

```
    returnFromOMI=pIOMI->AddMetadata(inMetadata, Reposid, &outMetadata, Namespace,
       Flag, Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<AddMetadata>"
                        "<Metadata>"
                        "<PhysicalTable Name=\"New Table\" Desc=\"New Table added
                           through API\"/>"
                        "</Metadata>"
                        "<Reposid>A0000001.A2345678</Reposid>"
                        "<NS>SAS</NS>"
                        "<Flags>268435456</Flags>"
                        "<Options/>"
                        "</AddMetadata>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific DeleteMetadata Examples

The following code fragments delete a SASLibrary object and a PhysicalTable object using first the standard interface and then using the DoRequest method.

- Java Example of a DeleteMetadata Call
- Visual Basic Example of a DeleteMetadata Call
- Visual C++ Example of a DeleteMetadata Call

Java Example of a DeleteMetadata Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void deleteMetadata() {

    int returnFromOMI;
    String inMetadata;
    StringHolder outMetadata;
    String Namespace;
    int Flag;
    String Options;

    inMetadata="<SASLibrary Id=\"A2345678.A2000001\"/>
                <PhysicalTable Id=\"A2345678.A2000001\"/>";
    outMetadata = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 268436480;
    Options = "";

    try {
      returnFromOMI = connection.DeleteMetadata(inMetadata, outMetadata,
        Namespace, Flag, Options);
    }

    catch (com.sas.iom.SASIOMDefs.GenericError e)
    {
      System.out.println(e);
     }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

     private void runToMsg() {

        int returnFromOMI;
```

```
        String inputXML;
        StringHolder outputXML;

        outputXML = new org.omg.CORBA.StringHolder();

        inputXML = "<DeleteMetadata>" +
                   "<Metadata>" +
                   "<SASLibrary Id=\"A2345678.A2000001\" />" +
                   "<PhysicalTable Id=\"A2345678.A3000001\"/>" +
                   "</Metadata>" +
                   "<NS>SAS</NS>" +
                   "<Flags>268436480</Flags>" +
                   "<Options/>" +
                   "</DeleteMetadata>";

       try {
          returnFromOMI = connection.DoRequest(inputXML, outputXML);
       }

       catch (com.sas.iom.SASIOMDefs.GenericError e)
       {
          System.out.println(e);
       }

     }
```

Visual Basic Example of a DeleteMetadata Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inMetadata    As String
   Dim outMetadata   As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   inMetadata="<SASLibrary Id=""A2345678.A2000001""/>
    <PhysicalTable Id=""A2345678.A3000001""/>"
   Namespace="SAS"
   Flag = 268436480
   Options = ""

   returnFromOMI = obOMI.DeleteMetadata(inMetadata, outMetadata, Namespace,
     Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
```

```
**************************************
**************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<DeleteMetadata>" + _
              "<metadata>" + _
              "<SASLibrary Id=""A2345678.A2000001""/>" + _
              "<PhysicalTable Id=""A2345678.A3000001""/>" + _
              "</Metadata>" + _
              "<NS>SAS</NS>" + _
              "<Flags>268436480</Flags>" + _
              "<Options/>" + _
              "</DeleteMetadata>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a DeleteMetadata Call

Standard Interface

```
**************************************
**************************************
Visual C++  Example of Standard Interface
**************************************
**************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString inMetadataStr("<SASLibrary Id=\"A2345678.A2000001\"/>
    <PhysicalTable Id=\"A2345678.A3000001\"/>");
    BSTR inMetadata = inMetadataStr.AllocSysString();

    CString outMetadataStr("");
    BSTR outMetadata = outMetadataStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 268436480;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->DeleteMetadata(inMetadata, &outMetadata, Namespace,
      Flag, Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<DeleteMetadata>"
                        "<Metadata>"
                        "<SASLibrary Id=\"A2345678.A2000001\"/>"
                        "<PhysicalTable Id=\"A2345678.A3000001\"/>"
                        "</Metadata>"
                        "<NS>SAS</NS>"
                        "<Flags>268436480</Flags>"
                        "<Options/>"
                        "</DeleteMetadata>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific GetMetadata Examples

The following code fragments retrieve the name, description, and columns of the PhysicalTable object with the Id value of A2345678.A2000001, first using the standard interface and then using the DoRequest method.

- Java Example of a GetMetadata Call
- Visual Basic Example of a GetMetadata Call
- Visual C++ Example of a GetMetadata Call

Java Example of a GetMetadata Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void getMetadata() {

    int returnFromOMI;
    String inMetadata;
    StringHolder outMetadata;
    String Namespace;
    int Flag;
    String Options;

    inMetadata="<PhysicalTable Id=\"A2345678.A2000001\" Name=\"\" Desc=\"\">
          <Columns/>
        </PhysicalTable>";
    outMetadata = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 0;
    Options = "";

   try {
      returnFromOMI = connection.GetMetadata(inMetadata, outMetadata, Namespace,
        Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {
```

```
    int returnFromOMI;
    String inputXML;
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<GetMetadata>" +
               "<Metadata>" +
               "<PhysicalTable Id=\"A5K2EL3N.A4000001\  Name=\"\" Desc=\"\">
                 <Columns/>
                </PhysicalTable>" +
               "</Metadata>" +
               "<NS>SAS</NS>" +
               "<Flags>0</Flags>" +
               "<Options/>" +
               "</GetMetadata>";

   try {
      returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of a GetMetadata Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inMetadata    As String
   Dim outMetadata   As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   inMetadata="<PhysicalTable Id=""A2345678.A2000001"" Name="""" Desc="""">
                 <Columns/></PhysicalTable>"
   Namespace="SAS"
   Flag = 0
   Options = ""

   returnFromOMI = obOMI.GetMetadata(inMetadata, outMetadata, Namespace,
     Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetMetadata>" + _
              "<Metadata>" + _
              "<PhysicalTable Id=""A2345678.A200001"" Name="""" DESC="""">
                  <Columns/>
               </PhysicalTable>" + _
              "</Metadata>" + _
              "<NS>SAS</NS>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetMetadata>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a GetMetadata Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString inMetadataStr("<PhysicalTable Id=\"A2345678.A2000001\" Name=\"\" Desc=\"\">
            <Columns/></PhysicalTable>");
    BSTR inMetadata = inMetadataStr.AllocSysString();

    CString outMetadataStr("");
    BSTR outMetadata = outMetadataStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetMetadata(inMetadata, &outMetadata, Namespace,
       Flag, Options);
```

```
}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<GetMetadata>"
                        "<Metadata>"
                        "<PhysicalTable Id=\"A2345678.A2000001\" Name=\"\" Desc=\"\">
                           <Columns/>
                         </PhysicalTable>"
                        "</Metadata>"
                        "<NS>SAS</NS>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetMetadata>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific GetMetadataObjects Examples

The following code fragments retrieve all objects of type PhysicalTable using first the standard interface and then using the DoRequest method.

- Java Example of a GetMetadataObjects Call
- Visual Basic Example of a GetMetadataObjects Call
- Visual C++ Example of a GetMetadataObjects Call

Java Example of a GetMetadataObjects Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void getMetadataObjects() {

    int returnFromOMI;
    String Reposid;
    String AType;
    StringHolder Objects;
    String Namespace;
    int Flag;
    String Options;

    Reposid="A0000001.A2345678";
    AType="PhysicalTable";
    Objects = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 0;
    Options = "";

   try {
      returnFromOMI = connection.GetMetadataObjects(Reposid, AType, Objects, Namespace,
         Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {
```

```
    int returnFromOMI;
    String inputXML;
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<GetMetadataObjects>" +
               "<Reposid>A0000001.A2345678</Reposid>" +
               "<Type>PhysicalTable</Type>" +
               "<Objects/>" +
               "<NS>SAS</NS>" +
               "<Flags>0</Flags>" +
               "<Options/>" +
               "</GetMetadataObjects>";

   try {
      returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of a GetMetadataObjects Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim Reposid       As String
   Dim AType         As String
   Dim Objects       As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   Reposid="A0000001.A2345678"
   AType="PhysicalTable"
   Namespace="SAS"
   Flag = 0
   Options = ""

   returnFromOMI = obOMI.GetMetadataObjects(Reposid, AType, Objects, Namespace,
      Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetMetadataObjects>" + _
              "<Reposid>A0000001.A2345678</Reposid>" + _
              "<Type>PhysicalTable</Type>" + _
              "<subtypes/>" + _
              "<NS>SAS</NS>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetMetadataObjects>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a GetMetadataObjects Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString ReposidStr("A0000001.A2345678");
    BSTR Reposid = ReposidStr.AllocSysString();

    CString ATypeStr("PhysicalTable");
    BSTR AType = ATypeStr.AllocSysString();

    CString ObjectsStr("");
    BSTR Objects = ObjectsStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetMetadataObjects(Reposid, AType, &Objects, Namespace,
      Flag, Options);
```

```
}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<GetMetadataObjects>"
                        "<Reposid>A0000001.A2345678</Reposid>"
                        "<Type>PhysicalTable</Type>"
                        "<Objects/>"
                        "<NS>SAS</NS>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetMetadataObjects>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific GetNamespaces Examples

The following code fragments issue a GetNamespaces call using first the standard interface and then using the DoRequest method.

- Java Example of a GetNamespaces Call
- Visual Basic Example of a GetNamespaces Call
- Visual C++ Example of a GetNamespaces Call

Java Example of a GetNamespaces Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

     private void getNamespaces() {

        int returnFromOMI;
        StringHolder Namespace;
        int Flag;
        String Options;

        Namespace = new org.omg.CORBA.StringHolder();
        Flag = 0;
        Options = "";

       try {
          returnFromOMI = connection.GetNamespaces(Namespace, Flag, Options);
       }

       catch (com.sas.iom.SASIOMDefs.GenericError e)
       {
          System.out.println(e);
       }

     }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

     private void runToMsg() {

        int returnFromOMI;
        String inputXML;
        StringHolder outputXML;

        outputXML = new org.omg.CORBA.StringHolder();

        inputXML = "<GetNamespaces>" +
```

```
                    "<Namespaces/>" +
                    "<Flags>0</Flags>" +
                    "<Options/>" +
                    "</GetNamespaces>";

        try {
           returnFromOMI = connection.DoRequest(inputXML, outputXML);
        }

        catch (com.sas.iom.SASIOMDefs.GenericError e)
        {
           System.out.println(e);
        }

      }
```

Visual Basic Example of a GetNamespaces Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   Flag=0
   Options=""

   returnFromOMI = obOMI.GetNamespaces(Namespace, Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetNamespaces>" + _
              "<Namespaces/>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetNamespaces>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)
```

```
End Sub
```

Visual C++ Example of a GetNamespaces Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString NamespaceStr("");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetNamespaces(&Namespace,Flag,Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<GetNamespaces>"
                        "<Namespaces/>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetNamespaces>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

SAS 9.1 Open Metadata Interface: Reference

Program–Specific GetRepositories Examples

The following code fragments issue a GetRepositories method call using first the standard interface and then using the DoRequest method.

- Java Example of a GetRepositories Call
- Visual Basic Example of a GetRepositories Call
- Visual C++ Example of a GetRepositories Call

Java Example of a GetRepositories Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************


     private void getRepositories() {

        int returnFromOMI;
        StringHolder Repositories;
        int Flag;
        String Options;

        Repositories = new org.omg.CORBA.StringHolder();
        Flag = 0;
        Options = "";

       try {
          returnFromOMI = connection.GetRepositories(Repositories, Flag, Options);
       }

       catch (com.sas.iom.SASIOMDefs.GenericError e)
       {
          System.out.println(e);
       }

     }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

     private void runToMsg() {

        int returnFromOMI;
        String inputXML;
        StringHolder outputXML;

        outputXML = new org.omg.CORBA.StringHolder();
```

```
        inputXML = "<GetRepositories>" +
                   "<Repositories/>" +
                   "<Flags>0</Flags>" +
                   "<Options/>" +
                   "</GetRepositories>";

       try {
          returnFromOMI = connection.DoRequest(inputXML, outputXML);
       }

       catch (com.sas.iom.SASIOMDefs.GenericError e)
       {
          System.out.println(e);
       }

     }
```

Visual Basic Example of a GetRepositories Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim Repositories  As String
   Dim Flag          As Long
   Dim Options       As String

   Flag = 0
   Options = ""

   returnFromOMI = obOMI.GetRepositories(Repositories, Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetRepositories>" + _
              "<Repositories/>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetRepositories>"
```

```
   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a GetRepositories Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString RepositoriesStr("");
    BSTR Repositories = RepositoriesStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetRepositories(&Repositories,Flag,Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<GetRepositories>"
                        "<Repositories/>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetRepositories>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

SAS 9.1 Open Metadata Interface: Reference

Program–Specific GetSubtypes Examples

The following code fragments retrieve the Subtypes for supertype DataTable using first the standard interface and then using the DoRequest method.

- Java Example of a GetSubtypes Call
- Visual Basic Example of a GetSubtypes Call
- Visual C++ Example of a GetSubtypes Call

Java Example of a GetSubtypes Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void getSubtypes() {

    int returnFromOMI;
    String SuperType;
    StringHolder Subtypes;
    String Namespace;
    int Flag;
    String Options;

    SuperType="DataTable";
    Subtypes = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 0;
    Options = "";

   try {
      returnFromOMI = connection.GetSubtypes(SuperType, Subtypes, Namespace,
        Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {

    int returnFromOMI;
    String inputXML;
```

```
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<GetSubtypes>" +
               "<Supertype>DataTable</Supertype>" +
               "<Subtypes/>" +
               "<NS>SAS</NS>" +
               "<Flags>0</Flags>" +
               "<Options/>" +
               "</GetSubtypes>";

   try {
      returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of a GetSubtypes Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim SuperType     As String
   Dim Subtypes      As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   SuperType="DataTable"
   Namespace="SAS"
   Flag = 0
   Options = ""

   returnFromOMI = obOMI.GetSubtypes(SuperType, Subtypes, Namespace,
     Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()
```

```
   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetSubtypes>" + _
              "<Supertype>DataTable</Supertype>" + _
              "<Subtypes/>" + _
              "<NS>SAS</NS>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetSubtypes>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a GetSubtypes Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString SuperTypeStr("DataTable");
    BSTR SuperType = SuperTypeStr.AllocSysString();

    CString SubtypesStr("");
    BSTR Subtypes = SubtypesStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetSubtypes(SuperType, &Subtypes, Namespace,
      Flag, Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
```

```
{

    long returnFromOMI;

    CString inputXMLStr("<GetSubtypes>"
                        "<Supertype>DataTable</Supertype>"
                        "<Subtypes/>"
                        "<NS>SAS</NS>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetSubtypes>";

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific GetTypeProperties Examples

The following code fragments issue a GetTypeProperties call for metadata type Column using first the standard interface and then using the DoRequest method.

- Java Example of a GetTypeProperties Call
- Visual Basic Example of a GetTypeProperties Call
- Visual C++ Example of a GetTypeProperties Call

Java Example of a GetTypeProperties Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void getTypeProperties() {

    int returnFromOMI;
    String AType;
    StringHolder Properties;
    String Namespace;
    int Flag;
    String Options;

    AType="Column";
    Properties = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 0;
    Options = "";

   try {
     returnFromOMI = connection.GetTypeProperties(AType, Properties, Namespace,
       Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {

    int returnFromOMI;
    String inputXML;
```

```
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<GetTypeProperties>" +
               "<Type>Column</Type>" +
               "<Properties/>" +
               "<NS>SAS</NS>" +
               "<Flags>0</Flags>" +
               "<Options/>" +
               "</GetTypeProperties>";

   try {
      returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of a GetTypeProperties Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim AType         As String
   Dim Properties    As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   AType="Column"
   Namespace="SAS"
   Flag = 0
   Options = ""

   returnFromOMI = obOMI.GetTypeProperties(AType, Properties, Namespace,
     Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()
```

```
   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetTypeProperties>" + _
              "<Type>Column</Type>" + _
              "<Properties/>" + _
              "<NS>SAS</NS>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetTypeProperties>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a GetTypeProperties Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString ATypeStr("Column");
    BSTR AType = ATypeStr.AllocSysString();

    CString PropertiesStr("");
    BSTR Properties = PropertiesStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetTypeProperties(AType, &Properties, Namespace,
       Flag, Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
```

```
{

    long returnFromOMI;

    CString inputXMLStr("<GetTypeProperties>"
                        "<Type>Column</Type>"
                        "<Properties/>"
                        "<NS>SAS</NS>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetTypeProperties>";

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific GetTypes Examples

The following code fragments issue a GetTypes method call using first the standard interface and then using the DoRequest method.

- Java Example of a GetTypes Call
- Visual Basic Example of a GetTypes Call
- Visual C++ Example of a GetTypes Call

Java Example of a GetTypes Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void getTypes() {

    int returnFromOMI;
    StringHolder Types;;
    String Namespace;
    int Flag;
    String Options;

    Types = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 0;
    Options = "";

   try {
     returnFromOMI = connection.GetTypes(Types, Namespace, Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {

    int returnFromOMI;
    String inputXML;
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();
```

```
    inputXML = "<GetTypes>" +
               "<Types/>" +
               "<NS>SAS</NS>" +
               "<Flags>0</Flags>" +
               "<Options/>" +
               "</GetTypes>";

   try {
     returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of a GetTypes Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim Types         As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   Namespace="SAS"
   Flag = 0
   Options = ""

   returnFromOMI = obOMI.GetTypes(Types, Namespace, Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<GetTypes>" + _
              "<Types/>" + _
```

```
              "<NS>SAS</NS>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</GetTypes>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of a GetTypes Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString TypesStr("");
    BSTR Types = TypesStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->GetTypes(&Types, Namespace, Flag, Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<GetTypes>"
                        "<Types/>"
                        "<NS>SAS</NS>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</GetTypes>");

    BSTR inputXML = inputXMLStr.AllocSysString();
```

```
    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific IsSubtypeOf Examples

The following code fragments issue an IsSubtypeOf method call using first the standard interface and then the DoRequest method. The purpose of the call is to determine whether the JoinTable metadata type is a subtype of the DataTable metadata type.

- Java Example of an IsSubtypeOf Call
- Visual Basic Example of an IsSubtypeOf Call
- Visual C++ Example of an IsSubtypeOf Call

Java Example of an IsSubtypeOf Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void isSubTypeOf() {

    int returnFromOMI;
    String AType;
    String Supertype;
    BooleanHolder Result;
    String Namespace;
    int Flag;
    String Options;

    AType="JoinTable";
    Supertype="DataTable";
    Result = new org.omg.CORBA.BooleanHolder();
    Namespace="SAS";
    Flag=0;
    Options="";

   try {
      returnFromOMI = connection.IsSubtypeOf(AType, Supertype, Result, Namespace,
        Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

private void runToMsg() {
```

```
    int returnFromOMI;
    String inputXML;
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<IsSubtypeOf>" +
               "<Supertype>DataTable</Supertype>" +
               "<Type>JoinTable</Type>" +
               "<Result/>" +
               "<NS>SAS</NS>" +
               "<Flags>0</Flags>" +
               "<Options/>" +
               "</IsSubtypeOf>";

  try {
      returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }
   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of an IsSubtypeOf Call

Standard Interface

```
'****************************************
'****************************************
VB Example of Standard Interface
'****************************************
'****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim AType         As String
   Dim Supertype     As String
   Dim Result        As Boolean
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   AType="JoinTable"
   Supertype="DataTable"
   Namespace="SAS"
   Flag=0
   Options=""

   returnFromOMI = obOMI.IsSubtypeOf(AType, Supertype, Result, Namespace,
     Flag, Options)


End Sub
```

DoRequest Method

```
**************************************
**************************************
VB Example of DoRequest Method
**************************************
**************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<IsSubtypeOf>" + _
              "<Supertype>DataTable</Supertype>" + _
              "<Type>JoinTable</Type>" + _
              "<Result/>" + _
              "<NS>SAS</NS>" + _
              "<Flags>0</Flags>" + _
              "<Options/>" + _
              "</IsSubtypeOf>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of an IsSubtypeOf Call

Standard Interface

```
**************************************
**************************************
Visual C++  Example of Standard Interface
**************************************
**************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString ATypeStr("JoinTable");
    BSTR AType = ATypeStr.AllocSysString();

    CString SupertypeStr("DataTable");
    BSTR Supertype = SupertypeStr.AllocSysString();

    VARIANT_BOOL  Result;
    CString ResultStr("");

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 0;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->IsSubtypeOf(AType,Supertype,&Result,Namespace,
      Flag,Options);
```

```
    if (Result == VARIANT_TRUE) {
       ResultStr ="True";
    }
    else if (Result == VARIANT_FALSE)  {
       ResultStr = "False";
    }

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<IsSubtypeOf>"
                        "<Supertype>DataTable</Supertype>"
                        "<Type>JoinTable</Type>"
                        "<Result/>"
                        "<NS>SAS</NS>"
                        "<Flags>0</Flags>"
                        "<Options/>"
                        "</IsSubtypeOf>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page

Program–Specific UpdateMetadata Examples

The following code fragments issue an UpdateMetadata method call using first the standard interface and then the DoRequest method. The examples submit new attribute values for an object of metadata type Event.

- Java Example of an UpdateMetadata Call
- Visual Basic Example of an UpdateMetadata Call
- Visual C++ Example of an UpdateMetadata Call

Java Example of an UpdateMetadata Call

Standard Interface

```
****************************************
****************************************
JAVA Example of Standard Interface
****************************************
****************************************

 private void updateMetadata() {

    int returnFromOMI;
    String inMetadata;
    StringHolder outMetadata;
    String Namespace;
    int Flag;
    String Options;

    inMetadata="<Event Id=\"A5TBRRNR.AE0000AD\" Name=\"Updated Event 01\"
       Desc=\"Updated Event\"/>";
    outMetadata = new org.omg.CORBA.StringHolder();
    Namespace="SAS";
    Flag = 268435456;
    Options = "";

   try {
      returnFromOMI = connection.UpdateMetadata(inMetadata, outMetadata, Namespace,
        Flag, Options);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

DoRequest Method

```
****************************************
****************************************
JAVA Example of DoRequest Method
****************************************
****************************************

 private void runToMsg() {

    int returnFromOMI;
```

```
    String inputXML;
    StringHolder outputXML;

    outputXML = new org.omg.CORBA.StringHolder();

    inputXML = "<UpdateMetadata>" +
               "<Metadata>" +
               "<Event Id=\"A5TBRRNR.AE0000AD\" Name=\"Updated Event 01\"
                   Desc=\"Updated Event\"/>" +
               "</Metadata>" +
               "<NS>SAS</NS>" +
               "<Flags>268435456</Flags>" +
               "<Options/>" +
               "</UpdateMetadata>";

   try {
      returnFromOMI = connection.DoRequest(inputXML, outputXML);
   }

   catch (com.sas.iom.SASIOMDefs.GenericError e)
   {
      System.out.println(e);
   }

 }
```

Visual Basic Example of an UpdateMetadata Call

Standard Interface

```
****************************************
****************************************
VB Example of Standard Interface
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inMetadata    As String
   Dim outMetadata   As String
   Dim Namespace     As String
   Dim Flag          As Long
   Dim Options       As String

   inMetadata="<Event id=""A5TBRRNR.AE0000AD"" Name=""Updated Event 01""
     Desc=""Updated Event""/>"
   Namespace="SAS"
   Flag = 268435456
   Options = ""

   returnFromOMI = obOMI.UpdateMetadata(inMetadata, outMetadata, Namespace,
      Flag, Options)

End Sub
```

DoRequest Method

```
****************************************
****************************************
VB Example of DoRequest Method
```

```
****************************************
****************************************

Private Sub runtomsg_Click()

   Dim returnFromOMI As Long
   Dim inputXML      As String
   Dim outputXML     As String

   inputXML = "<UpdateMetadata>" + _
              "<Metadata>" + _
              "<Event id=""A5TBRRNR.AE0000AD"" Name=""Updated Event 01""
                Desc=""Updated Event""/>" + _
              "</Metadata>" + _
              "<NS>SAS</NS>" + _
              "<Flags>268435456</Flags>" + _
              "<Options/>" + _
              "</UpdateMetadata>"

   returnFromOMI = obOMI.DoRequest(inputXML, outputXML)

End Sub
```

Visual C++ Example of an UpdateMetadata Call

Standard Interface

```
****************************************
****************************************
Visual C++  Example of Standard Interface
****************************************
****************************************

void COmiVCPlusDlg::OnRuntomethod()
{

    long returnFromOMI;

    CString inMetadataStr("<Event Id=\"A5TBRRNR.AE0000AD\"
       Name=\"Updated Event 01\" Desc=\"Updated Event\"/>");
    BSTR inMetadata = inMetadataStr.AllocSysString();

    CString outMetadataStr("");
    BSTR outMetadata = outMetadataStr.AllocSysString();

    CString NamespaceStr("SAS");
    BSTR Namespace = NamespaceStr.AllocSysString();

    long Flag = 268435456;

    CString OptionsStr("");
    BSTR Options = OptionsStr.AllocSysString();

    returnFromOMI=pIOMI->UpdateMetadata(inMetadata, &outMetadata, Namespace,
       Flag, Options);

}
```

DoRequest Method

```
****************************************
****************************************
Visual C++  Example of DoRequest Method
****************************************
****************************************


void COmiVCPlusDlg::OnRuntomsg()
{

    long returnFromOMI;

    CString inputXMLStr("<UpdateMetadata>"
                        "<Metadata>"
                        "<Event Id=\"A5TBRRNR.AE0000AD\" Name=\"Updated Event 01\"
                          Desc=\"Updated Event\"/>"
                        "</Metadata>"
                        "<NS>SAS</NS>"
                        "<Flags>268435456</Flags>"
                        "<Options/>"
                        "</UpdateMetadata>");

    BSTR inputXML = inputXMLStr.AllocSysString();

    CString outputXMLStr;
    BSTR outputXML = outputXMLStr.AllocSysString();

    returnFromOMI = pIOMI->DoRequest(inputXML, &outputXML);
}
```

Previous Page | Next Page | Top of Page