SAS THE POWER TO KNOW®

# SAS® 9.1.3 Open Metadata Interface

Reference

Second Edition

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2005. *SAS® 9.1.3 Open Metadata Interface: Reference, Second Edition*. Cary, NC: SAS Institute Inc.

**SAS® 9.1.3 Open Metadata Interface: Reference, Second Edition**

ISBN 978–1–59047–812–7

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

1st printing, July 2005
2nd printing, November 2006
3rd printing, June 2007

# Contents

# What's New

## Overview of the SAS Open Metadata Interface

The SAS Open Metadata Interface is an XML-based API for interacting with metadata. It is the basic interface that SAS clients use to deliver metadata services in the SAS Open Metadata Architecture, which was introduced with SAS 9.

This section describes the features of SAS Open Metadata Interface that are new or enhanced since SAS 9. It also introduces a new SAS language interface for maintaining metadata.

## Details of SAS Open Metadata Interface Enhancements

- Beginning with SAS 9.1.3, Service Pack 3, the IServer class Status method supports <PlatformVersion/> and <ModelVersion/> parameters. These parameters enable clients to retrieve the SAS Metadata Server and SAS Metadata Model version numbers, respectively. For more information, see "Status" on page 187.
- Beginning with SAS 9.1.3, Service Pack 3, the IOMI class CheckinMetadata method supports an unlimited number of characters in the changeDesc parameter. changeDesc enables clients to store a description of object modifications in a Change metadata object that is generated by the checkin process. Information entered into changeDesc is now stored in a TextStore object instead of as a Change attribute. For more information, see "CheckinMetadata" on page 135.
- Beginning with SAS 9.1.3, Service Pack 3, the METALIB procedure is available to automate the creation and maintenance of the table metadata associated with a SAS library. For more information, see "METALIB Procedure" on page 245.

P A R T 1

# Understanding the SAS Open Metadata Interface

# CHAPTER 1 Introduction

## About This Book

This book describes the SAS 9.1.3 Open Metadata Interface and contains all of the information you need to write a SAS Open Metadata Interface client that creates and manages metadata in SAS Metadata Repositories. This book describes

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

Hierarchy and association diagrams are provided to help you to understand the relationships between application-related metadata types in the SAS Metadata Model. You need to understand these relationships before you can issue SAS Open Metadata Interface method calls that read and write metadata. In addition, program-specific examples are provided that show how to issue SAS Open Metadata Interface method calls in Java, Visual Basic, and C++ clients.

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
```

This book also contains reference information about SAS language interfaces for metadata. The SAS language interfaces for metadata are SAS Open Metadata Interface clients that enable you to read and write metadata in SAS Metadata Repositories from base SAS.

*Note:* Due to production issues, reference information about the SAS namespace metadata types is provided only in online versions of this book. When you come upon a reference to the "Alphabetical Listing of SAS Namespace Metadata Types," look for this listing in *SAS Help and Documentation* or in *SAS OnlineDoc*. △

## Prerequisites

The SAS Open Metadata Interface is shipped as part of Base SAS software and uses the Integrated Object Model (IOM) to communicate with a SAS Metadata Server. Currently, this interface supports Visual Basic, C++, and Java clients.

The following software must be installed on computers where you will develop SAS Open Metadata Interface clients:

- Base SAS software
- The client software for the following SAS Integration Technologies Software IOM interface:
  - The IOM Bridge for COM – for Windows clients connecting to a SAS Metadata Server in a Windows environment or another operating environment.
  - The IOM Bridge for Java – for Java clients connecting to a SAS Metadata Server in any operating environment.

  SAS Integration Technologies Software can be installed from the SAS Software Installation Kit CD-ROM that is shipped along with SAS. For details about this IOM interface, see the Integration Technologies documentation.
- Software for the intended programming environment.

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

- 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.

## What Is the SAS Open Metadata Architecture?

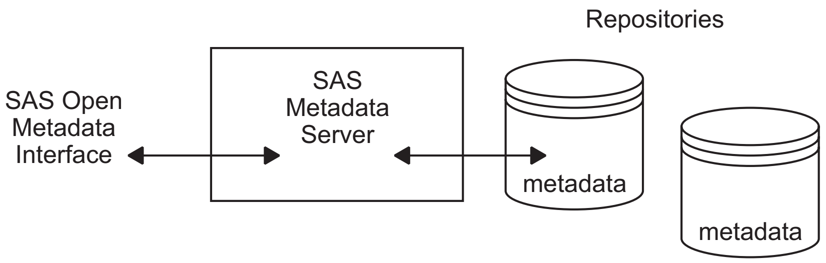
The SAS Open Metadata Architecture is a general-purpose metadata management facility that provides common metadata services to SAS applications. Using the metadata architecture, separate SAS applications can exchange metadata, which makes it easier for these applications to work together. The metadata 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 metadata 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.

**Display 1.1** 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 perform the following tasks:

- 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 you use SAS Open Metadata Interface clients provided by 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 Open Metadata Interface clients like SAS Data Integration Studio, SAS Management Console, PROC METADATA, PROC METAOPERATE, SAS Metadata DATA step functions, the SAS Java Metadata Interface, and the SAS Foundation Services Information Service Interface enable you to create and access metadata within a SAS Metadata Repository without having to know the details of the SAS Open Metadata Interface.

SAS Data Integration Studio
: is a thin-client application that enables you to create and manage ETL process flows – sequences of steps for the extraction, transformation, and loading of data. Using SAS Data Integration Studio, you create metadata objects that define sources, targets, and the transformations that connect them. The software then uses this metadata to generate or retrieve code that reads sources and creates targets in a file system. For more information, see the *SAS Data Integration Studio: User's Guide*.

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

PROC METADATA
: enables you to issue XML-formatted method calls to create, update, and query metadata from within a SAS program.

PROC METAOPERATE
: enables you to pause, resume, refresh, stop, and get the status of the metadata server from a SAS program.

SAS Language Metadata DATA Step Functions
: 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.

SAS Java Metadata Interface
: provides a Java programming 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 programming 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.

This book describes the SAS Open Metadata Interface and how to write clients that use this interface directly. It also provides reference information about the "METADATA Procedure" on page 240, the "METAOPERATE Procedure" on page 262, and the "SAS Metadata DATA Step Functions" on page 273. For information about the other SAS Open Metadata Interface clients, see their respective documentation.

## Important Concepts

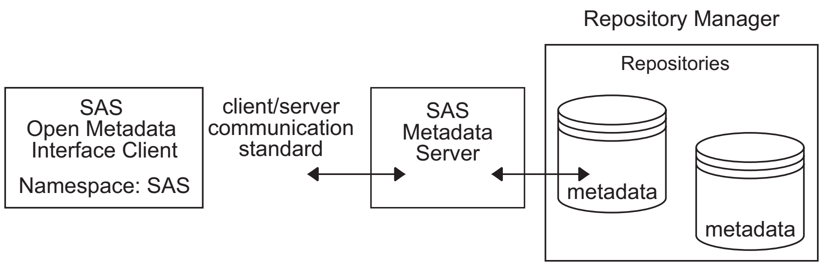
| | |
|---|---|
| metadata type | specifies a template that models the metadata for a particular kind of object. For example, the metadata type Column models the metadata for a SAS table column, and the metadata type RepositoryBase models the metadata for a repository. |
| namespace | specifies a group of related metadata types and their properties. Names are used to partition metadata into different contexts. The SAS Open Metadata Interface defines two namespaces: SAS and REPOS. The SAS namespace contains metadata types that describe application elements such as tables and columns. The REPOS namespace contains metadata types that describe 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 has the following characteristics:

- □ 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.

**Display 1.2** Accessing Metadata Defined in the SAS Namespace



For general information about writing SAS Open Metadata Interface clients, see Chapter 2, “Open Client Requirements,” on page 11. For an overview of the metadata types in the SAS namespace, see Chapter 4, “SAS Namespace Submodels,” on page 45. Reference information about each metadata type is provided in the “Alphabetical

Listing of SAS Namespace Metadata Types," which is provided only in online versions of this book. Details about the methods that are used to read or write metadata objects are provided in Chapter 7, "Methods for Reading and Writing Metadata (IOMI Class)," on page 115.

## 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 writing a SAS Open Metadata Interface client. However, the preferred method for registering repositories is to use SAS Management Console. SAS Management Console creates default authorization metadata and templates in each repository.

For information about registering repositories using SAS Management Console, see the Help for the product. For information about the metadata types used to represent repositories, see Chapter 6, "REPOS Namespace Metadata Types," on page 107.

## Controlling 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 the administrator might want to refresh the server to change certain configuration or invocation options, or to pause and resume repositories and the repository manager to temporarily change their state, for example, in preparation for a backup. He might also want to stop the metadata server. For more information about tasks that might require an administrator to control the metadata server, see the *SAS Intelligence Platform: System Administration Guide*.

The SAS Open Metadata Interface provides IServer class methods for controlling the metadata server. You can write a Java, Visual Basic, or C++ client that issues IServer class methods to control the metadata server. Or you can use a SAS Open Metadata Interface clients such as SAS Management Console or PROC METAOPERATE, which automate the IServer methods, to perform server control tasks.

A user must have *administrative user* status on the metadata server in order to issue IServer class methods, except Status. For more information, see "User and Group Management" in the *SAS Intelligence Platform: Security Administration Guide*.

For information about writing a SAS Open Metadata Interface client that controls the metadata server, see Chapter 2, "Open Client Requirements," on page 11 and Chapter 9, "Repository and Server Control Methods (IServer Class)," on page 181. For information about controlling the metadata server using SAS Management Console, see the SAS Management Console documentation. For information about PROC METAOPERATE, see "METAOPERATE Procedure" on page 262.

## Security

The SAS Metadata Server supports a variety of authentication providers to determine who can access the metadata server and uses an authorization facility to control user access to metadata on the server. Only users who have been granted *unrestricted user* status on the metadata server have unrestricted access to metadata on the server. A user must be either an *unrestricted user* or an *administrative user* on the metadata server in order to be able to create and delete repositories, modify a repository's registrations, change the state of a repository, and to register users. For more information, see "User and Group Management" in the *SAS Intelligence Platform: Security Administration 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. 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.

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 or 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 Chapter 8, "Security Methods (ISecurity Class)," on page 171.

For information about how the authorization facility makes authorization decisions, see the *SAS Intelligence Platform: Security Administration Guide*.

CHAPTER
2

# Open Client Requirements

## Types of SAS Open Metadata Interface Clients

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 to the SAS Metadata Server.

Most of your clients will read and write application metadata objects.

Clients write to repository objects in order 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 to 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 performing a backup. A client that controls the metadata server can also refresh a server to change certain server invocation and configuration options.

## Connecting to the SAS Metadata Server

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 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 that will be performed.

The client will need to establish a connection to the metadata server each time it issues a method call. To facilitate connection, it is recommended that you create a connection object that can be referenced in individual method calls.

### Server Connection Parameters

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

host
: The host name or Internet Protocol (IP) address of the computer 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. *XXXX* must be a unique number from 0-64K.

username
: An authenticated username on the metadata server. See the *SAS Intelligence Platform: Security Administration Guide* for information about authentication requirements.

password
: The password corresponding to the authenticated username.

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 Chapter 7, “Methods for Reading and Writing Metadata (IOMI Class),” on page 115. For more information about the IServer methods, see Chapter 9, “Repository and Server Control Methods (IServer Class),” on page 181. For more information about the ISecurity methods, see Chapter 8, “Security Methods (ISecurity Class),” on page 171.

## Call Interfaces

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 approach involves setting the name and data type of each parameter directly in the client, then referencing the variables in the method call. This approach 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. The XML string is passed to the server in the DoRequest method’s *inMetadata* parameter. This approach is referred to as the “DoRequest method”.

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” on page 144.

## 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 shows 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);
   }
}
```

## 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.

## Java IOMI Class Signature Summary Table

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

| 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 |

| Method | Parameter | Definition |
|---|---|---|
| | 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 |

| Method | Parameter | Definition |
|---|---|---|
| | 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 |

| Method | Parameter | Definition |
|---|---|---|
| | ns | java.lang.String |
| | flags | int |
| | options | java.lang.String |

## Visual Basic IOMI Class Signature Summary Table

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

| Method | Parameter | Definition |
|---|---|---|
| AddMetadata | InMetadata | As String |
| | Reposid | As String |
| | OutMetadata | As String |
| | Ns | As String |
| | Flags | As Long |
| Options | As Long | As String |
| 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<br>As Long |
| CheckoutMetadata | InMetadata | As String |
| | ProjectReposid | As String |
| | OutMetadata | As String |
| | Ns | As String |
| | Flags | As Long |
| | Options | As String<br>As Long |
| CopyMetadata | InMetadata | As String |
| | TargetReposid | As String |
| | OutMetadata | As String |
| | Ns | As String |

| Method | Parameter | Definition |
|---|---|---|
| | 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 |

| Method | Parameter | Definition |
|---|---|---|
| | 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 |

| Method | Parameter | Definition |
|---|---|---|
| | Flags | As Long |
| | Options | As String |
| | | As Long |

## Visual C++ IOMI Class Signature Summary Table

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

| 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] |

| Method | Parameter | Definition |
|---|---|---|
| | 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] |

| Method | Parameter | Definition |
|---|---|---|
| | 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 | long* [out, retval] | 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] |

| Method | Parameter | Definition |
|---|---|---|
| | Ns | BSTR [in] |
| | Flags | long [in] |
| | Options | BSTR [in] |
| | retval | long* [out, retval] |

## 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 querying a specific repository.

The sample program uses the standard call interface. Numbers have been inserted at the beginning of each step.Refer to the numbered descriptions following the code sample for an explanation of each 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 server connection parameters.
   ***************************************************************************/
   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.
 ******************************************************************************/
 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.

## 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.

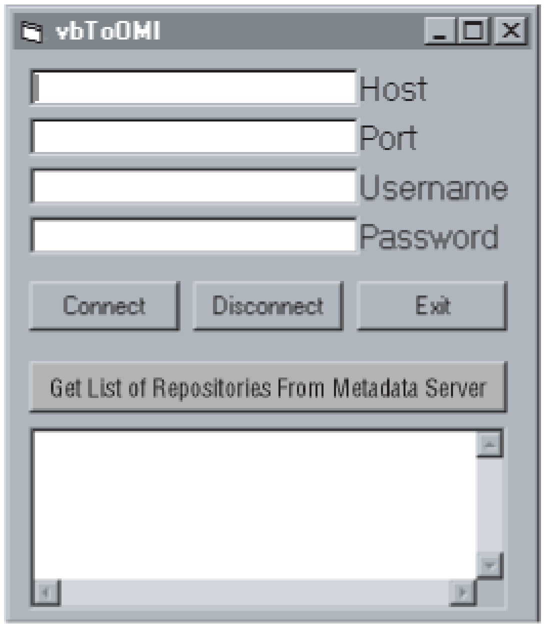
Clients that use the IOM Bridge for COM must additionally select the following type library:

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 a 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 querying 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. Refer to the numbered descriptions following the code sample for an explanation of each 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. |

## 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.

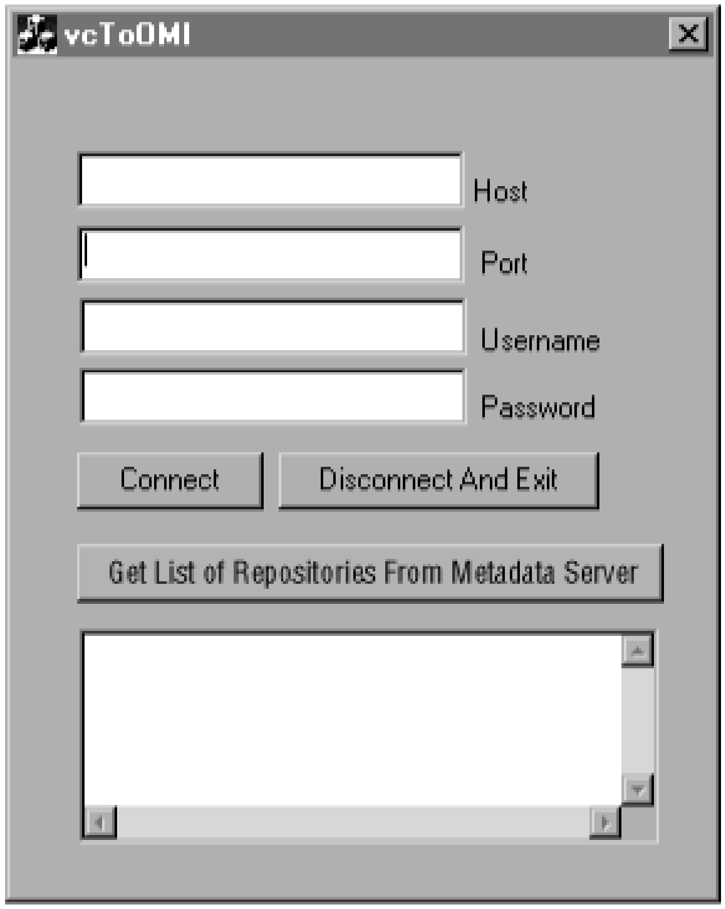
Clients that use the IOM Bridge for COM must additionally select the following:

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 querying 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,
      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();
          }

  }
```

## 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 Java Metadata Interface: User's Guide*.

P A R T 2

# SAS Metadata Model

# CHAPTER 3
# Overview of the SAS Metadata Model and Model Documentation

## Namespaces

The SAS Open Metadata Interface provides 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 Data Integration 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. Other 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.

## Understanding Associations

An association is a property that describes the relationship that joins two metadata types. In the SAS Metadata Model, two associations exist between any 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 lets 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.

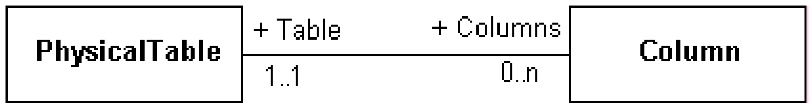
### Cardinality

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 the upper bound 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.

The following association diagram illustrates the concept of cardinality:



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.

See Chapter 5, "Hierarchy and Association Diagrams," on page 69 for a graphical representation of the main associations that are defined between the metadata types in each submodel.

## Overview of the SAS Metadata Model Documentation

This reference provides the following information about each metadata type:

- Chapter 4, "SAS Namespace Submodels," on page 45 provides an overview of the SAS namespace metadata types and groups metadata types that are used to support particular functions into submodels to help you navigate the SAS Metadata Model.
- The "Hierarchical Listing of Metadata Types" on page 65 shows the object class hierarchy.
- Chapter 5, "Hierarchy and Association Diagrams," on page 69 contains hierarchy diagrams that depict the inheritance structure in each submodel (from the most abstract types to the most specialized types) and association diagrams that illustrate the relationships between the metadata types within a submodel.
- An "Alphabetical Listing of SAS Namespace Metadata Types" provides reference information about each SAS namespace metadata type. This listing is available only in online versions of this book. Look for the listing in *SAS Help and Documentation* or *SAS OnlineDoc*.
- Chapter 6, "REPOS Namespace Metadata Types," on page 107 provides reference information about the metadata types defined to represent repositories.

The metadata type descriptions in the "Alphabetical Listing of SAS Namespace Metadata Types" contain the details necessary to construct a metadata property string. Before reading this documentation, see "Using the Metadata Types Reference" on page 42 for important information about how the reference information is structured.

When you are ready to begin creating metadata objects, see "Model Usage Scenarios" in the *SAS Open Metadata Interface: User's Guide*. These scenarios describe the specific metadata types and associations needed to represent the most common application elements. They also provide sample XML requests that show how to create the metadata objects.

## Using the Metadata Types Reference

The "Alphabetical Listing of SAS Namespace Metadata Types" contains the following information for each metadata type:

- 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
- an overview of the type's usage
- 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 |

*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 |
|---|---|---|---|---|---|
| Name:<br>AccessControls<br>Subelements:<br>AccessControl<br>AccessControlEntry<br>AccessControlTemplates<br>Partner Name:<br>Objects | the access controls that are defined for this object | 0 to * | No | Yes | Yes |

In the 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 metadata type.

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 object.

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.

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, the access control object must exist before an association can be created. 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.

## SAS Metadata Model 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. For more information, see “GetMetadata” on page 148.

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. Because objects are persisted to disk with a GMT datetime value, an object created in local time might have a different datetime value on disk. For example, an object created at ’30May2003:16:20:01’ CST could have a persisted datetime value of ’30May2003:21:20:01’. To accommodate the storage conversion, the server converts values that you specify in an XMLSELECT search string to GMT for you. However, the datetime values returned by the server will look different than the values that you submitted in the search string.

### "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.

CHAPTER

4

# SAS Namespace Submodels

## Overview of SAS Namespace Submodels

The SAS namespace contains the metadata types for defining application metadata objects. The SAS namespace metadata 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, metadata types that are used to support a particular scenario are grouped into submodels. The groupings are for documentation purposes only.

All methods in the IOMI class are valid with the SAS namespace metadata types. For information about these methods, see Chapter 7, "Methods for Reading and Writing Metadata (IOMI Class)," on page 115.

To see descriptions of the metadata types, see the "Alphabetical Listing of SAS Namespace Metadata Types" in online versions of this reference. This reference is available in the SAS Online Help and Documentation and in SAS OnlineDoc.

## Submodels in the SAS Metadata Model

The following submodels have been identified to help you to navigate the SAS Metadata Model. The submodels provide functional groupings of the metadata types.

Analysis submodel
: includes metadata types that are used to describe statistical transformations, multidimensional data sources, and OLAP information. For more information, see "Analysis Submodel" on page 48.

Authorization submodel
: includes metadata types that are used to define access controls. Metadata objects based on authorization metadata 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. For more information, see "Authorization Submodel" on page 49.

Business Information submodel
: includes metadata 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 more information, see "Business Information Submodel" on page 49.

Foundation submodel
: includes the basic metadata types of the model, from which all other types are derived, and some utility metadata types. For more information, see "Foundation Submodel" on page 50.

Grouping submodel
: includes metadata types that are used to group metadata objects together in a particular context or hierarchy. For more information, see "Grouping Submodel" on page 52.

Mining submodel
: includes metadata types that are used to store analytic information associated with data mining. For more information, see "Mining Submodel" on page 52.

Property submodel
: includes metadata 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 more information, see "Property Submodel" on page 53.

Relational submodel
: includes metadata 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 more information, see "Relational Submodel" on page 55.

Resource submodel
: includes metadata types that are used to describe data resources such as files, directories, SAS libraries, SAS catalogs and catalog entries. For more information, see "Resource Submodel" on page 57.

Software Deployment submodel
: includes metadata types that are used to describe software, servers, and connection information. For more information, see "Software Deployment Submodel" on page 59.

Transform submodel
: includes metadata 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 more information, see "Transform Submodel" on page 61.

XML submodel
: includes metadata types that are used to describe XML constructs such as SAS XML LIBNAME Engine map definitions and XPath location paths. For more information, see "XML Submodel" on page 64.

## Analysis Submodel

This submodel contains the metadata types that are used by the SAS OLAP server.

### Metadata Types

The following metadata types, relevant to the Analysis submodel, are classified as Container, Aggregation, and Other:

AggregateAssociation

Aggregation

Cube

Dimension

Hierarchy

Level

Measure

OLAPProperty

OLAPSchema

### Container Metadata Types

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

### Aggregation Metadata Types

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

### Other Metadata Types

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

### Usage

These metadata 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.

*Note:* Application developers are discouraged from creating or consuming Cube metadata directly with the SAS Open Metadata Interface. Instead, Cubes should be defined by using either SAS OLAP Cube Studio or PROC OLAP. For more information, see the documentation for SAS OLAP Cube Studio and PROC OLAP. △

## Authorization Submodel

The Authorization submodel contains the metadata types that are used to define access controls.

### Metadata Types

The Authorization submodel has the following metadata types:

AccessControl

AccessControlEntry

AccessControlTemplate

Permission

PermissionCondition

SecurityRule

SecurityRuleScheme

SecurityTypeContainmentRule

### Usage

Application developers are discouraged from creating or consuming access control and permission metadata directly with the SAS Open Metadata Interface. Instead, access control and permission metadata should be defined by using the SAS Management Console Authorization Manager plug–in. For more information, see the documentation for SAS Management Console.

The security rule metadata types define inheritance rules for the SAS Authorization Facility. These types are for internal use.

## Business Information Submodel

This submodel contains the metadata types that are used to describe people and documents.

### Metadata Types

The Business Information submodel has the following metadata types:

Document

Email

Keyword

Location

Person

Phone

ResponsibleParty

Timestamp

UnitofTime

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

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

### Usage

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

## Foundation Submodel

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

### Metadata Types

The following metadata types, relevant to the Foundation submodel, are classified as Basic, Extension, Identity, and Role:

AbstractExtension

Classifier

Extension

ExternalIdentity
Feature
Identity
IdentityGroup
NumericExtension
Role
Root

## Basic Metadata Types

Root is the supertype for all metadata 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.

The ExternalIdentity metadata type 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 Metadata Types

AbstractExtension is the supertype for the extension metadata 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 metadata type is used for character values and the NumericExtension metadata type is used for numeric values.

## Identity Metadata Types

The Identity and IdentityGroup metadata types are used for access control.

## Role Metadata Types

There is only one Role metadata type, and it identifies the various “roles” associated with an object. For example, a relational table may have a role of “Source” for SAS Data Integration Studio. This means that SAS Data Integration Studio reads data from that table; it is not a table created by SAS Data Integration Studio. Another role may be “Summary” for the OLAP product. This identifies the way the OLAP product will use the table.

## Usage

The Foundation metadata types are primarily supertypes for other metadata types. The exceptions are the Extension metadata types and ExternalIdentity. 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 object's identity from another context.

## Grouping Submodel

The Grouping submodel contains the metadata types that are used to group metadata objects together in a particular context, as well as metadata types that are used to construct a hierarchy of metadata objects.

### Metadata Types

The Grouping submodel has the following metadata types:

Group

Tree

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

### Usage

Each of these metadata types may be associated to a SoftwareComponent object or DeployedComponent object (see "Software Deployment Submodel" on page 59). 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 SoftwareComponent object that represents Enterprise Miner 5.0. When any copy of Enterprise Miner 5.0 is run, it can query the metadata server for the SoftwareComponent object that represents its type of software, obtain the Tree object associated with that SoftwareComponent, and locate its "projects".

## Mining Submodel

The Mining submodel contains the metadata types that are used to store metadata about mining models, as created by SAS Enterprise Miner. The metadata types in this submodel are used in conjunction with the "Transform Submodel" on page 61.

### Metadata Types

The Mining submodel has the following metadata types:

AnalyticColumn

AnalyticTable

FitStatistic

MiningResult

Target

The MiningResult metadata type is a subtype of Transformation that is used to represent an Enterprise Miner Model.

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

The AnalyticTable metadata type contains analytic attributes 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.

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

The Target metadata type models the event or the value of interest for a mining model.

### Usage

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

## Property Submodel

The Property submodel contains the metadata types that are used to

- provide parameter information for a stored process, including the valid choices for the parameter
- 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
- create a property sheet used to drive a user interface.

### Metadata Types

The Property submodel contains the following metadata types:

AbstractProperty

AssociationProperty

AttributeProperty

LocalizedResource

LocalizedType

Property

PropertyGroup

PropertySet

PropertyType

Prototype

PrototypeProperty

The AbstractProperty, LocalizedType, and PrototypeProperty metadata types are supertypes that aren't expected to be instantiated; they exist so that 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 metadata object is required to have have an owning PropertyType metadata object. The PropertyType metadata object stores the SQL type as an integer. If the property type 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 can also have a StoredConfiguration metadata object defined. The StoredConfiguration object will contain additional information such as a list of enumerated values.

The LocalizedResource metadata type 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 metadata object can have a set of LocalizedResource metadata objects, one for each textual object that can be displayed and its locale.

The Prototype metadata type is used to provide a template for creating a set of metadata objects. For example, some of the metadata types used to describe a Workspace Server are ServerComponent, a set of Connection objects (COMConnection and TCPIPConnection), and possibly Transformation, to describe 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

### Providing Parameter Information for a Stored Process

A stored process is a SAS program that can 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: Analytical Solutions (ANT), Software Architecture (SAR), and 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
- through a PropertyGroup object.

If the Property is going to be displayed in a user interface and have its value selected through the user interface, then it should be part of a PropertyGroup. PropertyGroups can be organized in a hierarchy. For example, there can be a PropertyGroup for properties associated with Operating System and a set of subgroups for Windows 2000, z/OS, 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 can 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 can have no host-specific objects. Property objects, however, can be available through PropertyGroups that provide specific information for a specific environment.

### Supporting Default Values and Other Pre-Configured Parameters

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 Analytical Solutions (ANT) group wants an alternative setting for the Property to be "ANT", then a copy of the Property should be made and associated to a PropertySet. When the stored process is run for the ANT group, the property set for ANT is used, not the default settings.

It is frequently the case that a Property will be duplicated, with one copy in the Properties list, and another in the PropertySet list, and yet a third in the PropertyGroup list. This is a design tradeoff. It was decided that duplicating objects across the various grouping mechanisms was preferrable to the overhead that would be incurred by merging the Property objects across the various grouping mechanisms.

In summary:

- 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 can be grouped in hierarchies.

### Defining Property Sheets

A Prototype can also be used to define alternative values. For example, in a situation similar to the one described in “Supporting Default Values and Other Preconfigured Parameters” using a SASLibrary object as an example, instead of having PropertyGroups associated directly with a SASLibrary to define alternative values, a property sheet could be created by using the metadata type Prototype.

A Prototype is a metadata type that describes the settings of another type in a particular environment. For SASLibrary metadata objects, there might be a Prototype object for DB2 libraries, another for Oracle, and another for Base SAS. The Prototype would have a top-level PropertyGroup, which in turn might 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.

## Relational Submodel

The Relational submodel contains the metadata types that are 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 metadata types, relevant to the Relational submodel, are classified as Container, Table, Column, Key, and Other:

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

## Container Metadata Types

These metadata types are subtyped from the TablePackage metadata type in the "Resource Submodel" on page 57. The only container defined in the Relational submodel is DatabaseSchema.

## Table Metadata Types

DataTable is the supertype of the table metadata types and has an association to the Column metadata type that is inherited by all of the table metadata types. The RelationalTable metadata type is used to represent a table that does not have a physical representation. Use the RelationalTable metadata type to represent a table that is described in a data model. A relational table can have a fixed physical location, such as in a particular database schema. Use the PhysicalTable metadata type to represent a table that has a fixed physical location.

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

The ExternalTable metadata type 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 metadata type.

## Column Metadata Types

LogicalColumn is the supertype used to describe column-like objects. It is not a part of a relational table. Column is the metadata type used to describe columns in a relational table. The ColumnRange metadata type represents a range of columns that have the same characteristics and that will all be acted upon in the same manner for any transformation or query.

### Key Metadata Types

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

### Other Metadata Types

The Index metadata type is used to represent an index associated with a table, and the SASPassword metadata type 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 might be associated to a Directory or set of Directory metadata objects which contain the file system path. A table has a set of columns represented by the Column metadata type. For more information about defining keys, see “Usage Scenario: Creating Metadata for Tables, Columns, and Keys” in “Model Usage Scenarios” in the *SAS Open Metadata Interface: User’s Guide*.

## Resource Submodel

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

### Metadata Types

The following metadata types, relevant to the Resource submodel, are categorized as Container, Content, and Other:

ArchiveEntry
ArchiveFile
ContentLocation
ContentType
DeployedDataPackage
Device
DeviceType
Directory

File
Memory
SASCatalog
SASCatalogEntry
SASFileRef
SASLibrary
Stream
Text
TextStore

## Container Types

The Directory, SASLibrary, SASCatalog, and TablePackage metadata types describe containers for data and are subtyped from the metadata type DeployedDataPackage. TablePackage is the supertype for all data containers that contain relational tables. The DatabaseSchema and OLAPSchema metadata types are also subtyped from DeployedDataPackage, but these metadata types are contained in the Relational and Analysis submodels, respectively (see“Relational Submodel” on page 55 and “Analysis Submodel” on page 48).

## Content Types

The content metadata types describe physical location and are subtyped from the ContentLocation metadata type. These include file metadata types, text metadata types, and other location metadata types.

### Text Metadata Types

The text metadata types are supertypes for the content metadata types and represent a physical container for data. Unfortunately, the name “text” is a misnomer for these metadata types, because binary as well as textual information might be stored in objects represented by the Text metadata type. Metadata types that represent files, SAS catalogs, and URLs as well as the TextStore metadata type, which contains data stored directly in the metadata repository, are all subtyped from the Text metadata type. The Document metadata type, which is used to represent a URI, is also subtyped from Text.

### File Metadata Types

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

### Entry Metadata Types

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

### Device Metadata Types

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

### Other Metadata Types

The SASFileRef metadata type is used specifically with SAS software to identify the physical location used by SAS software. The Report metadata type is a generic metadata type subtyped from the Classifier metadata type (refer to the “Foundation Submodel” on page 50) that is used to describe the result of transforming data into another representation such as a textual table or a graph.

### Usage

The metadata types in the Resource submodel are 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 metadata type is used with an associated Directory metadata object. In addition, this file may be referenced by a SASFileref metadata object.

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 metadata object. The SASCatalog also has associated SASCatalogEntry objects. A SASCatalog is located by a SAS application through the SASLibrary metadata object. For more information about defining a SASLibrary metadata object, refer to “Usage Scenario: Creating Metadata for a SASLibrary” in “Model Usage Scenarios” in the *SAS Open Metadata Interface: User’s Guide*.

A Report can have a location in a file system. If it does, there would be an associated File or Directory metadata object if the report is actually a collection of files. A Report can also be delivered in other ways. For example, the Report can be sent directly to a printer Device, or an Email account. A Report can also be a static item that is stored, or 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 about defining transformations, refer to “Usage Scenario: Creating Metadata for a Stored Process” in “Model Usage Scenarios” in the *SAS Open Metadata Interface: User’s Guide*.

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

## Software Deployment Submodel

The Software Deployment submodel contains the metadata types that are used to describe software, servers, and connection information.

### Metadata Types

The following metadata types, relevant to the Software Deployment submodel, are classified as Software, Connection, Service, and Other:

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

## Software Metadata Types

The SoftwareComponent metadata type represents a type of software, for example, SAS/ACCESS for ORACLE, Version 9.0. The DeployedComponent metadata type describes software that is actually installed. An actual installation of SAS/ACCESS for ORACLE on any machine is represented by the DeployedComponent metadata type. An installation of software can have various configurations that are defined to run the software. This submodel contains several subtypes of DeployedComponent that represent different types of configured software.

- The ConfiguredComponent metadata type represents software that is configured to run. This includes software such as Java components or customizers.
- The ServiceComponent metadata type represents software that acts as a service.
- The ServerComponent metadata type is used to represent servers and spawners.
- The LogicalServer metadata type 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.
- The ServerContext metadata type groups non-homogeneous servers that all share common resources. ServerContext gives an application context for the grouped servers.

## Connection Metadata Types

These metadata types contain information about how to communicate with a DeployedComponent. The Connection metadata type is the parent class and has two primary subtypes, OpenClientConnection and SASClientConnection. The SASClientConnection metadata type was created because of specific limitations that 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 subtypes of OpenClientConnection, and each contains attributes that provide the protocol-specific connection information.

### Service Metadata Types

There are two metadata types in this category: ServiceType and NamedService. The ServiceType metadata type contains descriptive information about the types of services that are provided by a DeployedComponent. “DBMS” may be a ServiceType specified for an Oracle or DB2 server. The NamedService metadata type contains the name used by a naming service, such as the RMI registry, or Active Directory, to refer to a DeployedComponent.

### Other Metadata Types

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

### Usage

The metadata 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 metadata types are generic in nature since it is impractical to have a metadata type for every software system or every type of connection. In most use cases, these metadata types will have associated properties which give more specific information about the deployment of the system or provide additional options that can 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 “Usage Scenario: Creating Objects which Represent a DBMS” in “Model Usage Scenarios” in the *SAS Open Metadata Interface: User’s Guide*. Also see “Usage Scenario: Creating Metadata for a Workspace Server,” which contains information about using a prototype to create the metadata definition.

## Transform Submodel

The Transform submodel contains the metadata types that are used to

- describe stored processes
- represent ETL processes
- define queries
- schedule processes
- store initialization information for software components.

## Metadata Types

The following metadata 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 Metadata Types

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

## Event Metadata Type

The Event metadata type is used to describe conditions that must occur to drive other processes.

## Query Metadata Types

The GroupByClause, HavingClause, OnClause, WhereClause, and Join metadata types 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 will focus on the current metadata types.

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

The RowSelector and OrderByClause metadata types are used by the Transformation metadata type and subtypes to further qualify the transformation.

## Process Metadata Types

The process metadata 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 on the DeployedComponent that will be running the generated process.

The metadata for a process contains all of the information about the sources and the targets; therefore, if a change is made to any source, it is easy to identify the process and targets that might be impacted by the change.

The TransformationActivity metadata type represents a grouping of TransformationStep metadata objects. At this level, the TransformationSources and TransformationTargets associations represent the initial inputs to the activity and the final output of the activity. For detailed information about what is happening within the activity, the application should drill down first to the TransformationStep objects, then to the Transformation metadata objects. A TransformationStep is a grouping of Transformation metadata types. Transformation metadata types include ClassifierMap and Select. A ClassifierMap shows the mapping between Classifier metadata types. Examples of Classifier metadata types include PhysicalTable and Report. A Classifier often has features, for example, a PhysicalTable has Columns. These features are mapped by using a FeatureMap metadata type. The StepPrecendence metadata type is used to show the order of steps within an activity. If a StepPrecendence object is not defined for a Transformation, then it is assumed that the steps may run in parallel.

## Scheduling Metadata Types

Once a process has been defined and tested, the process can be scheduled. The Job metadata type groups TransformationActivity metadata objects into a runtime unit to be rescheduled. The JFJob metadata type represents a job which is scheduled in the LSF Job Flow.

## Usage

### Describing Stored Processes

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

### Representing 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 can be scheduled to run as a batch process or can be triggered by various external or internal events. StepPrecedence is used to show the order of TransformationSteps. Each level of the ETL process can have different locations where the process should be run. The ComputeLocations association is used to show the components that are capable of performing the process.

### Defining Queries

A query uses the Select metadata type, 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.

### Scheduling Processes

Any process defined in the metadata can be scheduled to run as a job. It is required that the process be part of a TransformationActivity. TransformationActivity objects can then be grouped together as a Job.

### Storing 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 “Usage Scenario: Creating Metadata for a Workspace Server” in “Model Usage Scenarios” in the *SAS Open Metadata Interface: User’s Guide*.

## XML Submodel

The XML submodel contains the metadata 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 metadata types:

SXLEMap

XPath

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

The XPath metadata type is used to store an XPath location path.

### Usage

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

## Hierarchical Listing of Metadata Types

The SAS Metadata Model is an object-oriented, hierarchical model. The following listing is illustrates the object hierarchy.

- Root
  - Hierarchy
  - SummaryStats (DEPRECATED)
  - Aggregation
  - AggregateAssociation (DEPRECATED)
  - AbstractTransformation
    - TransformationStep
      - SyncStep
    - TransformationActivity
    - FeatureMap
    - Transformation
      - ClassifierMap
        - Select
        - Join (DEPRECATED)
      - AbstractJob
        - Job
        - JFJob
      - MiningResult
  - QueryClause
    - WhereClause (DEPRECATED)
    - RowSelector
    - GroupByClause
    - HavingClause (DEPRECATED)
    - OrderByClause
    - OnClause (DEPRECATED)
  - Event
  - Variable
  - Feature
    - Level
    - OLAPProperty
    - StepPrecedence
      - ConditionalPrecedence
    - LogicalColumn
      - Measure
      - Column
        - ColumnRange
  - Classifier
    - Cube

        - Dimension
        - DataTable
            - ExternalTable
            - RelationalTable
                - PhysicalTable
                    - WorkTable
                - QueryTable
                    - JoinTable (DEPRECATED)
            - TableCollection
        - Index
        - Report
    - AbstractExtension
        - Extension
        - NumericExtension
    - Role
    - Identity
        - IdentityGroup
        - Person
    - ExternalIdentity
    - SASPassword
    - Key
        - UniqueKey
        - ForeignKey
    - KeyAssociation
    - Tree
    - Group
        - SXLEMap
    - DeployedDataPackage
        - OLAPSchema
        - DatabaseCatalog
        - SASCatalog
        - RelationalSchema
            - DatabaseSchema
            - DataSourceName
            - SASLibrary
        - ContentLocation
            - Text
                - TextStore
                - SASCatalogEntry
                - File
                    - ArchiveFile
                - ArchiveEntry

                - Document
            - Directory
            - Device
            - Stream
            - Memory
            - Connection
                - SASClientConnection
                - OpenClientConnection
                    - TCPIPConnection
                    - COMConnection
            - Email
        - SASFileRef
- ContentType
- DeviceType
- AuthenticationDomain
- SoftwareComponent
    - DeployedComponent
        - ConfiguredComponent
            - ServerComponent
                - LogicalServer
                - ServerContext
            - ServiceComponent
- Login
- Machine
- ServiceType
- NamedService
- SASLicense
- ResponsibleParty
- Location
- Phone
- Keyword
- Timestamp
- UnitofTime
- LocalizedResource
- LocalizedType
    - PropertyGroup
    - PropertyType
    - Prototype
    - AbstractProperty
        - Property
        - PrototypeProperty
            - AttributeProperty

            - AssociationProperty

- PropertySet
- AnalyticColumn
- AnalyticTable
- FitStatistic
- Target
- XPath
- Change
- Permission
- PermissionCondition
- AccessControl
    - AccessControlEntry
    - AccessControlTemplate
- SecurityRuleScheme
- SecurityRule
    - SecurityTypeContainmentRule
- EMModel
- EMRules
- PSPortalProfile
- PSPortalPage
- PSPortlet
- PSLayoutComponent
    - PSColumnLayoutComponent
    - PSGridLayoutComponent
- ITTransportAlias
- ITQueueAlias
- ITMsmqModel
- ITChannel
- ITSubscriber
    - ITContentSubscriber )
    - ITEventSubscriber
- ITFilter
- ITModel
- ITMap
- ITRendModel

CHAPTER

5

# Hierarchy and Association Diagrams

## Overview to Hierarchy and Association Diagrams

This section contains graphical representations of the metadata type relationships described in “Overview of SAS Namespace Submodels” on page 46. Hierarchy diagrams illustrate the supertype and subtype relationships defined in the SAS Metadata Model. Association diagrams illustrate the associations between related metadata types. The purpose of the diagrams is to help you understand the relationships between the SAS namespace metadata types. By understanding these relationships, you can

- identify which metadata types you need to use 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
- determine how to 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. After reviewing the diagrams, see "Model Usage Scenarios" in the *SAS Open Metadata Interface: User's Guide* for examples of how to use the SAS namespace metadata types to create actual objects of the most commonly used application elements.

### 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 is the association name and describes the nature of the connected square's association to the square.
- The *n..n* describes the cardinality of the association.

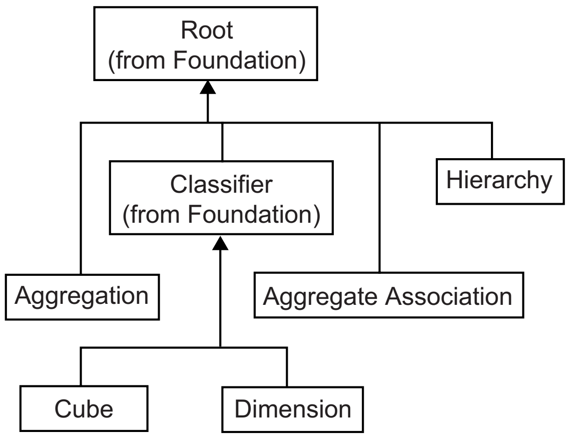
For more information about associations and cardinality, see "Understanding Associations" on page 40.

## Diagrams for Analysis Metadata Types

### Analysis Hierarchy

The Analysis Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent statistical transformations, multidimensional data sources, and OLAP information.

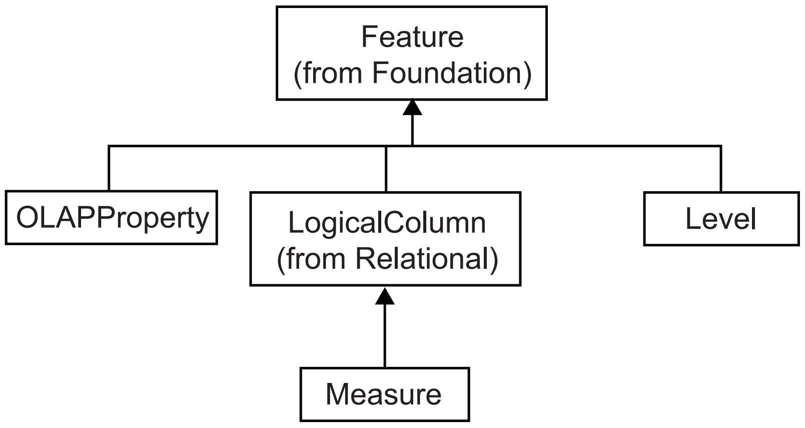
**Figure 5.1** Analysis Hierarchy Diagram



## Level, Measure Hierarchy

The Level and Measure Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent OLAP information.

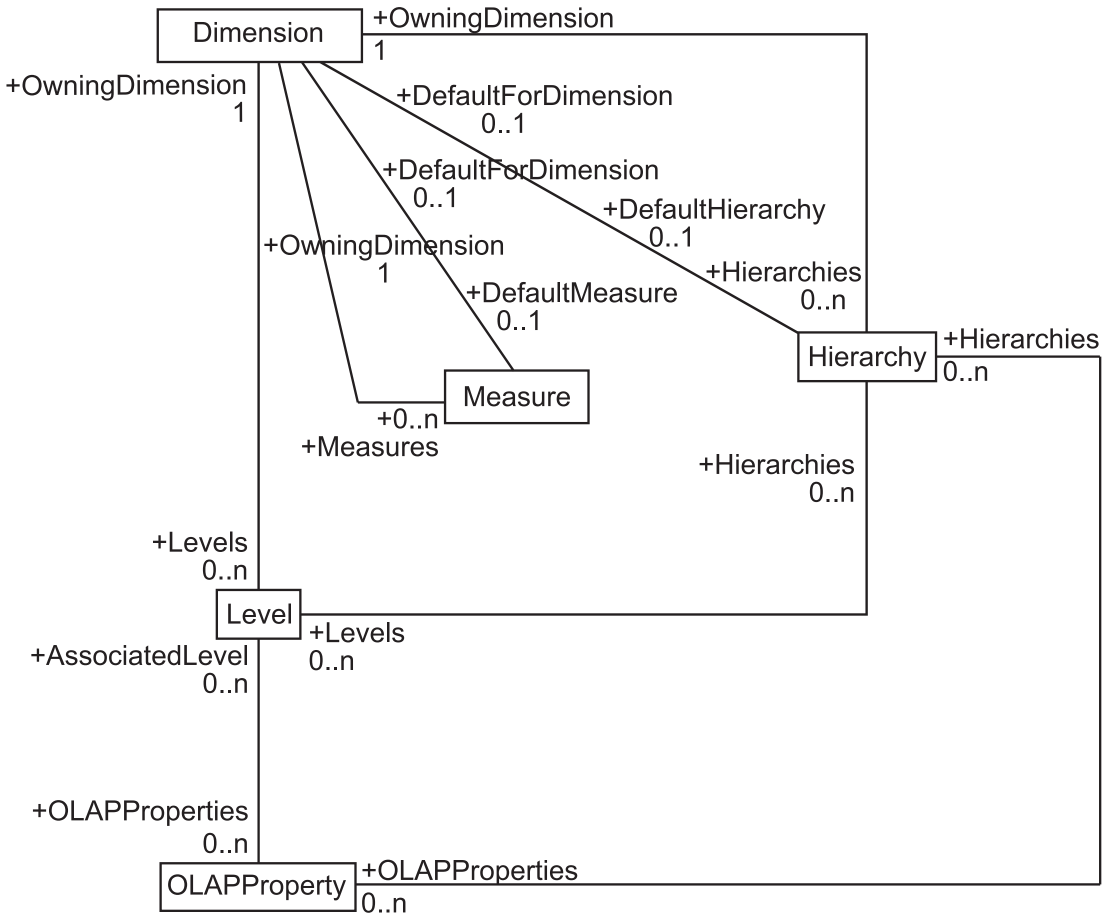
**Figure 5.2** Level and Measure Hierarchy Diagram



## Dimension Associations

The Dimension Associations Diagram illustrates the associations between the Dimension, Level, OLAPProperty, Measure, and Hierarchy metadata types.

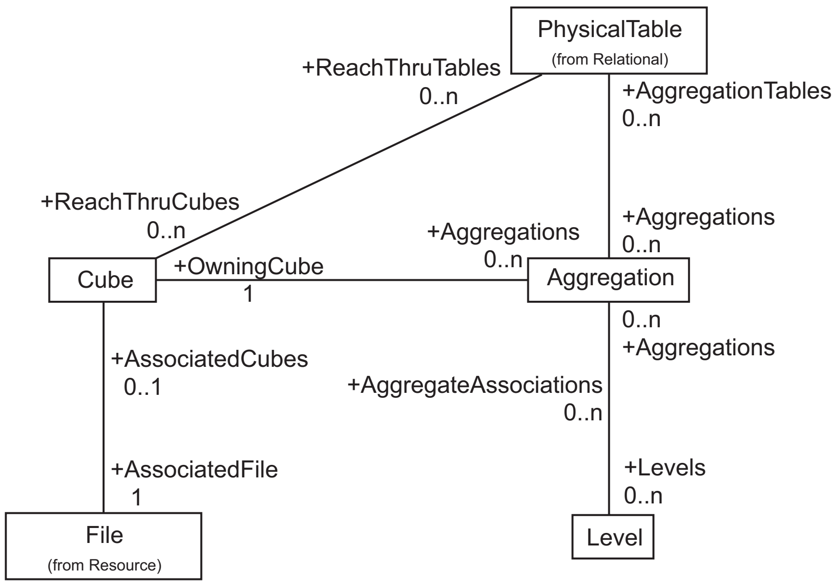
**Figure 5.3** Dimension Associations Diagram



## Physical Associations

The Physical Associations Diagram illustrates the associations between the Cube, File, PhysicalTable, Aggregation, and Level metadata types. The tables from which the Cube is derived are associated via one or more ClassifierMap objects. For more information, see the description of the “Transform Submodel” on page 61.

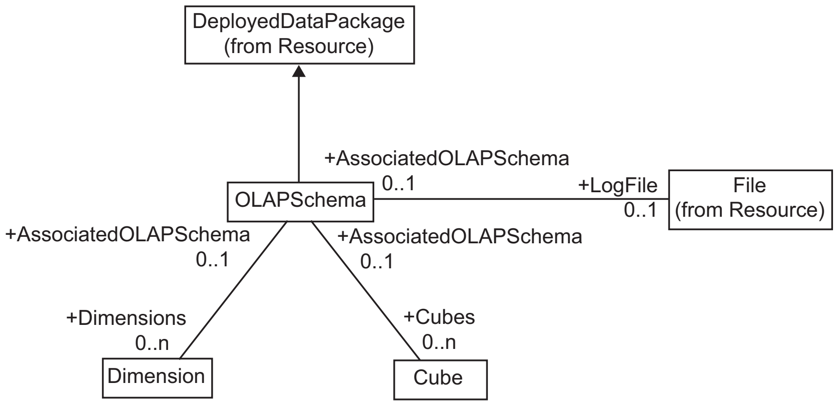
**Figure 5.4** Physical Associations Diagram



## OLAP Schema

The OLAP Schema Diagram illustrates the associations between the DeployedDataPackage, OLAPSchema, Dimension, File, and Cube metadata types.

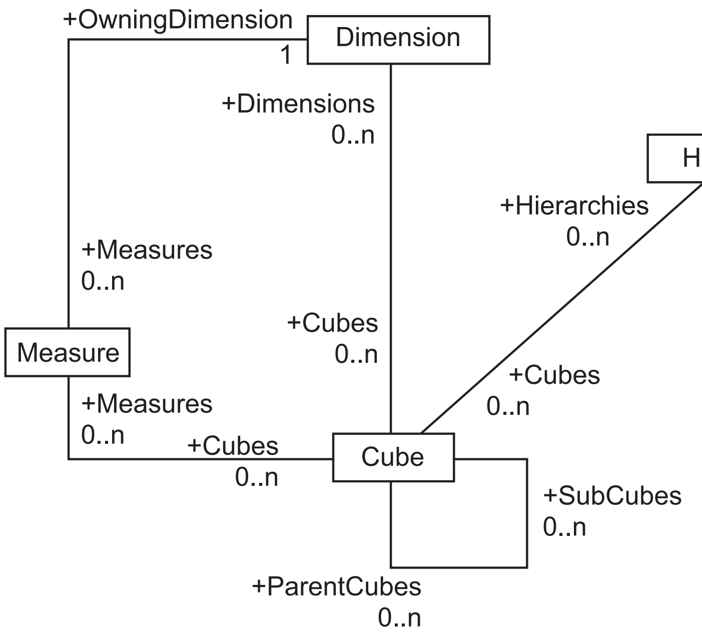
**Figure 5.5** OLAP Schema Diagram



## Cube Associations

The Cube Associations Diagram illustrates the associations between the Dimension, Measure, Cube, and Hierarchy metadata types.

**Figure 5.6** Cube Associations Diagram

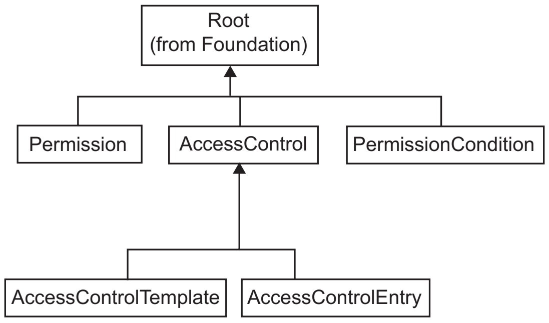


## Diagrams for Authorization Metadata Types

### Authorization Hierarchy

The Authorization Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent access controls.

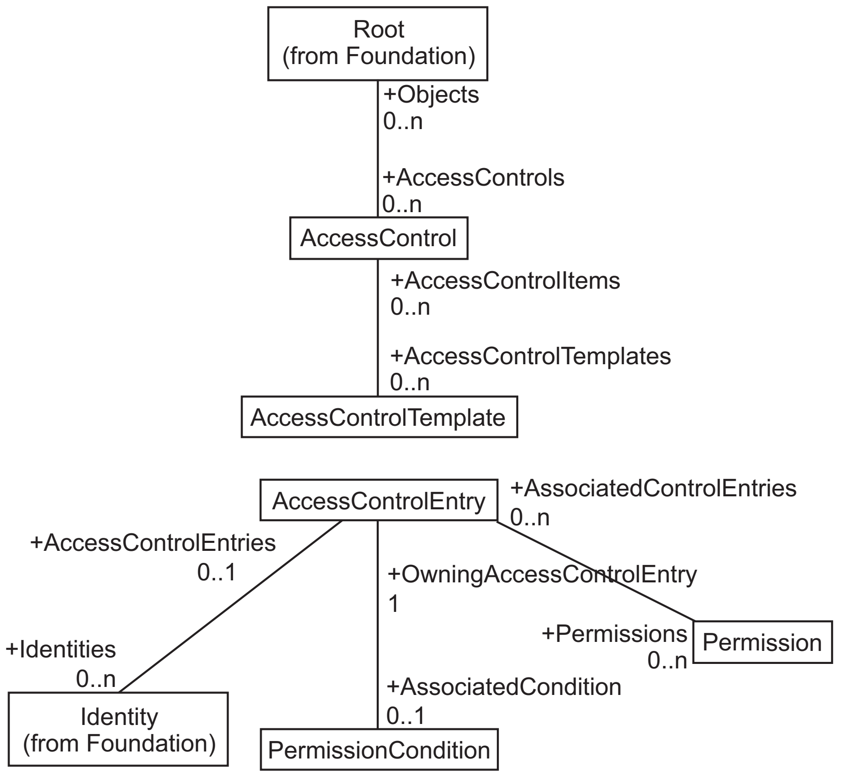
**Figure 5.7** Authorization Hierarchy Diagram



### 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.

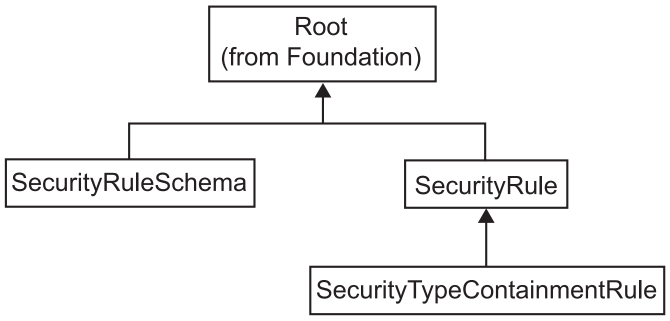
**Figure 5.8** Authorization Associations Diagram



## Security Rules Hierarchy

The Security Rules Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent security rules.

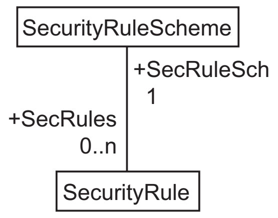
**Figure 5.9** Security Rules Hierarchy Diagram



## Security Rules Associations

The Security Rules Associations Diagram illustrates the associations between the SecurityRuleScheme and SecurityRule metadata types.

**Figure 5.10** Security Rules Associations Diagram

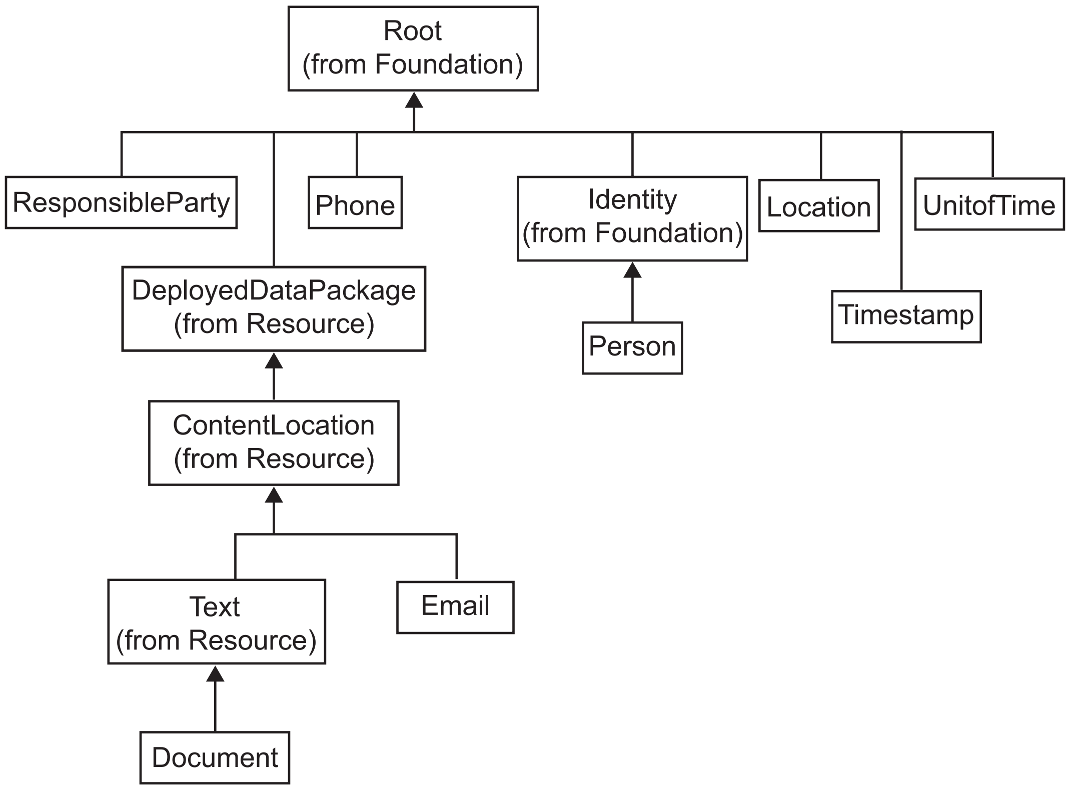


# Diagrams for Business Information Metadata Types

## Business Information Hierarchy

The Business Information Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent people, their responsibilities, information about how to contact them, and business documents.

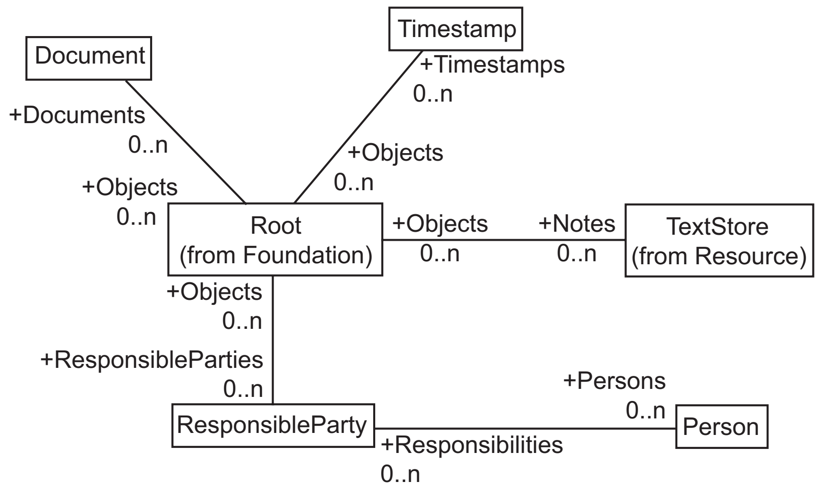
**Figure 5.11** Business Information Hierarchy Diagram



## Root Associations

The Root Associations Diagram illustrates the associations between the Root, Document, Timestamp, TextStore, ResponsibleParty, and Person metadata types.

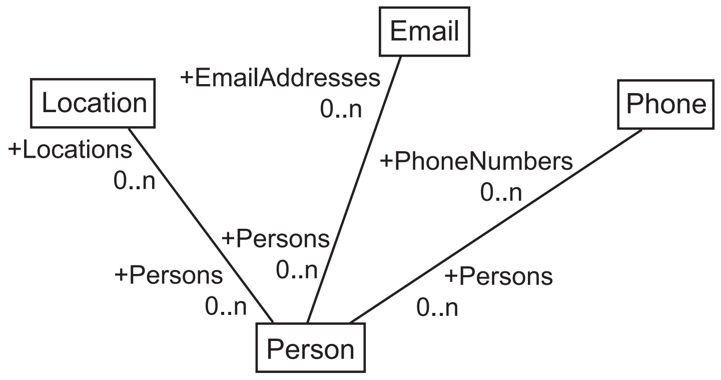
**Figure 5.12** Root Associations Diagram



## Person Associations

The Person Associations Diagram illustrates the associations between the Person, Location, Email, and Phone metadata types.

**Figure 5.13** Person Associations Diagram

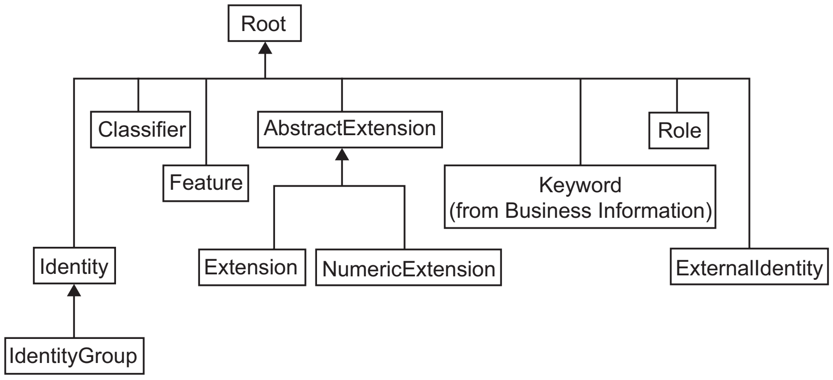


## Diagrams for Foundation Metadata Types

### Foundation Hierarchy

The Foundation Hierarchy Diagram illustrates the basic metadata types of the SAS Metadata Model, from which all other metadata types are derived.

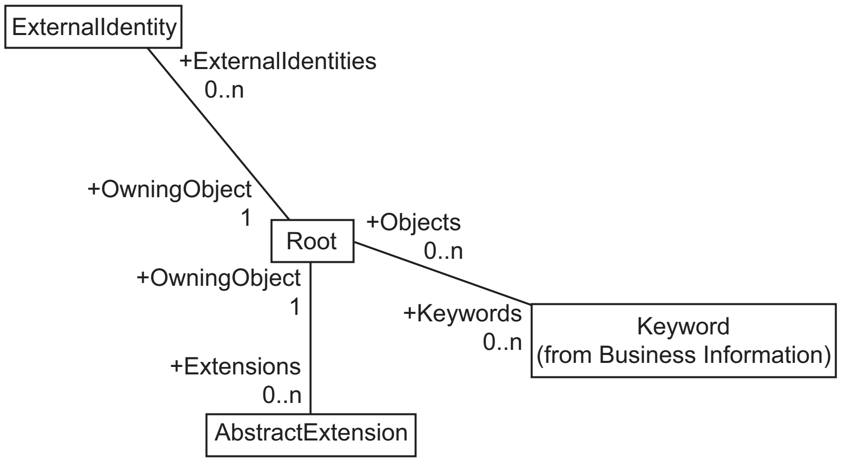
**Figure 5.14** Foundation Hierarchy Diagram



## Root Associations

The Root Associations Diagram illustrates the associations between the Root, ExternalIdentity, Keyword, and AbstractExtension metadata types.

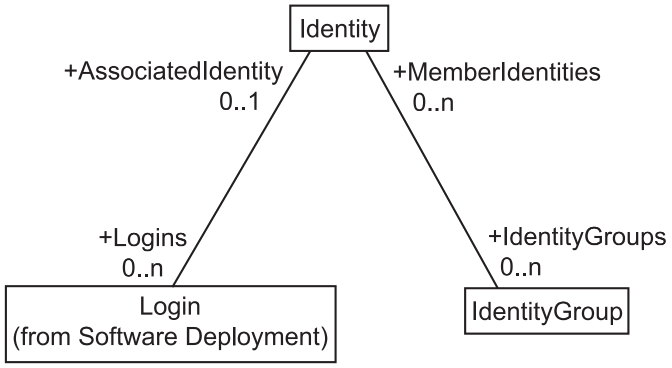
**Figure 5.15** Root Associations Diagram



## Identity Associations

The Identity Associations Diagram illustrates the associations between the Identity, Login, and IdentityGroup metadata types.

**Figure 5.16** Identity Associations Diagram

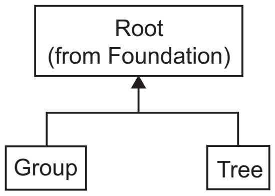


## Diagrams for Grouping Metadata Types

### Grouping Hierarchy

The Grouping Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent collections of data.

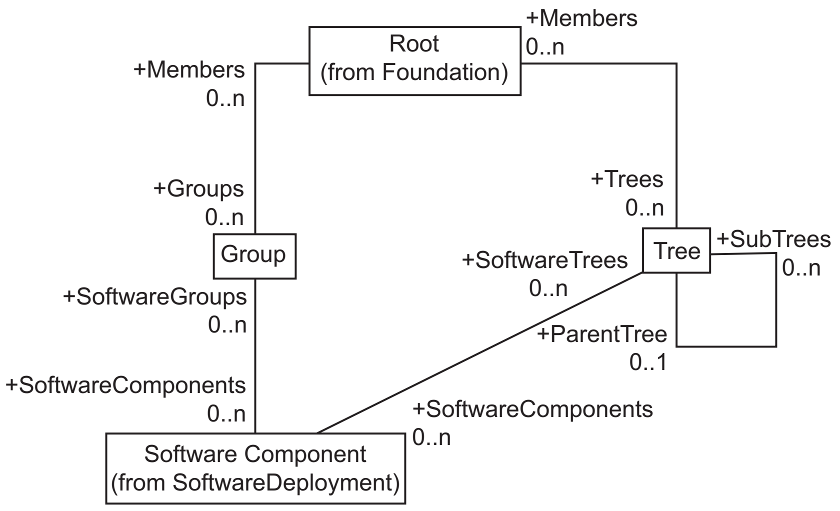
**Figure 5.17** Grouping Hierarchy Diagram



### Grouping Associations

The Grouping Associations Diagram illustrates the associations between the Root, Group, Tree, and SoftwareComponent metadata types.

**Figure 5.18** Grouping Associations Diagram

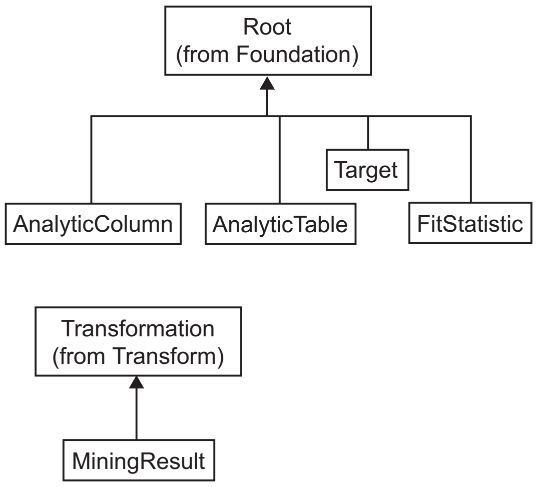


## Diagrams for Mining Metadata Types

### Mining Hierarchy

The Mining Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent analytic transformations.

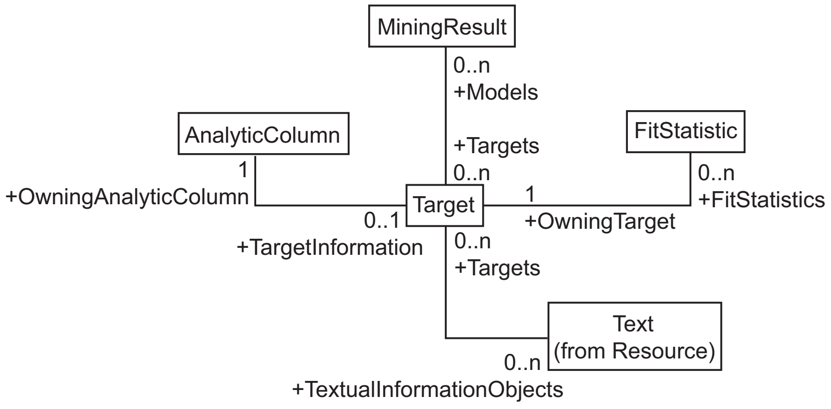
**Figure 5.19** Mining Hierarchy Diagram



### Target Associations

The Target Associations Diagram illustrates the associations between the AnalyticColumn, Target, MiningResult, Text, and FitStatistic metadata types.

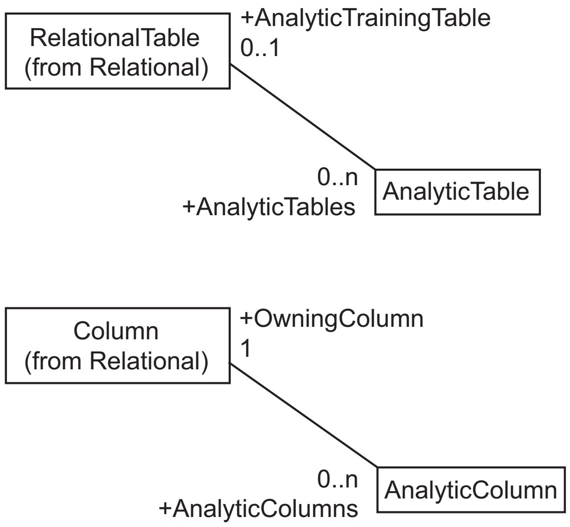
**Figure 5.20** Target Associations Diagram



## 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.

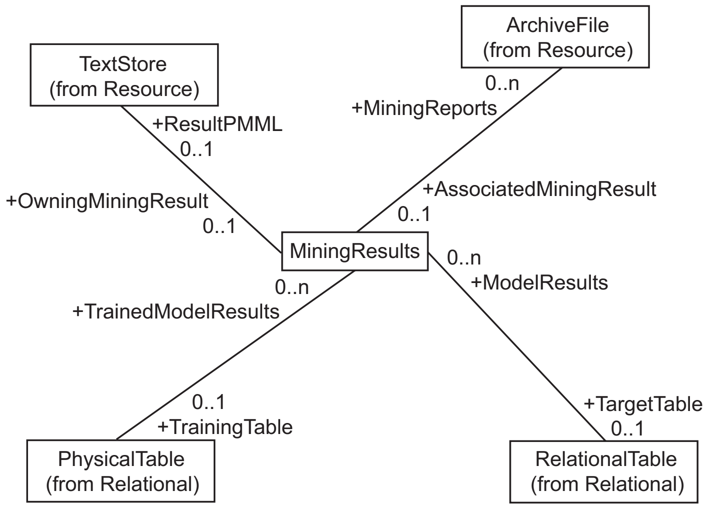
**Figure 5.21** Analytic Table and Column Associations Diagram



## ModelResult Associations

The ModelResult Associations Diagram illustrates the associations between the MiningResult, TextStore, ArchiveFile, RelationalTable, and PhysicalTable metadata types.

**Figure 5.22** ModelResult Associations Diagram

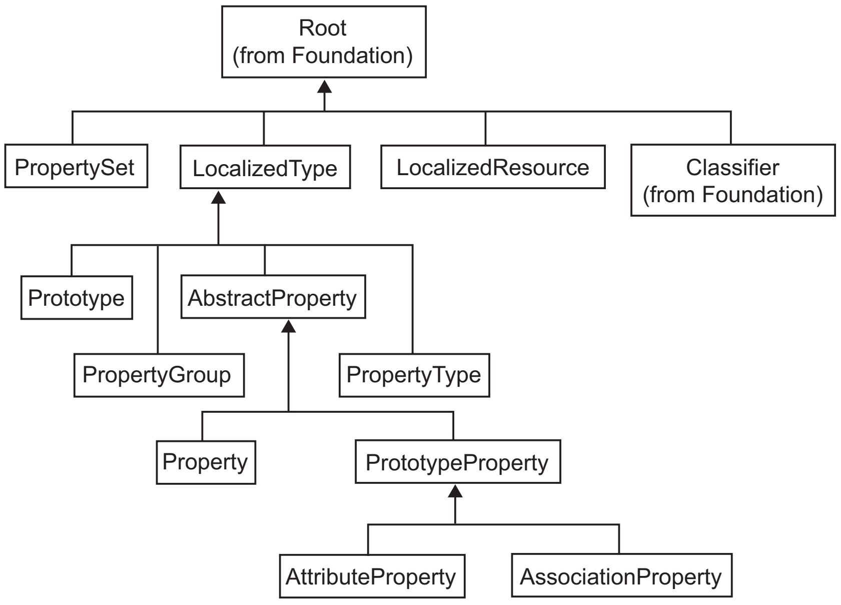


# Diagrams for Property Metadata Types

## Property Hierarchy

The Property Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent options and extended properties in the SAS Metadata Model.

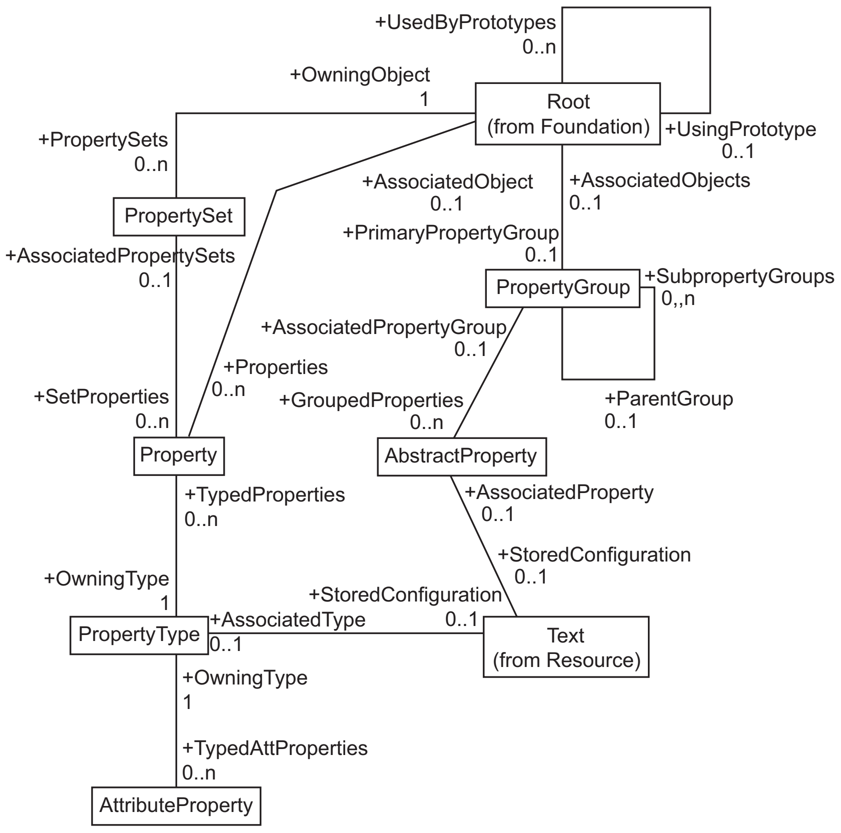
**Figure 5.23** Property Hierarchy Diagram



## Property Associations

The Property Associations Diagram illustrates the associations between the Root, PropertySet, Property, PropertyType, AttributeProperty, PropertyGroup, AbstractProperty, and Text metadata types.

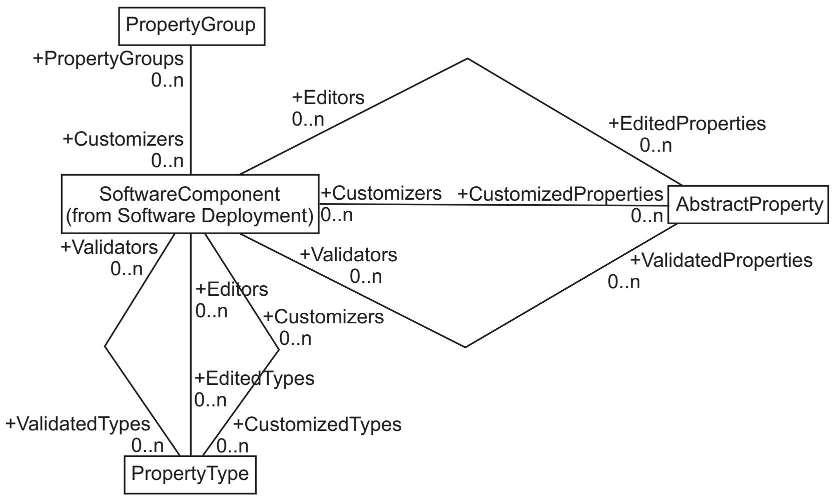
**Figure 5.24** Property Associations Diagram



## Configuration Associations

The Configuration Associations Diagram illustrates the associations between the PropertyGroup, SoftwareComponent, AbstractProperty, and PropertyType metadata types.

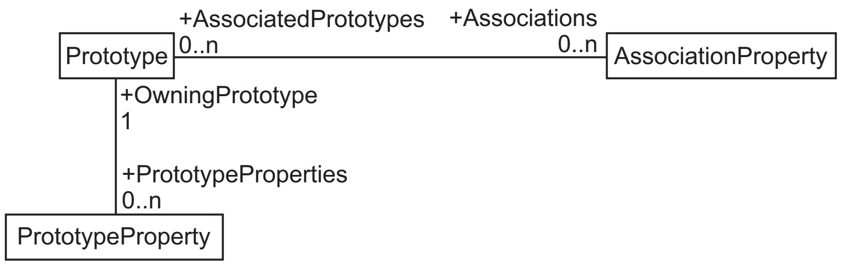
**Figure 5.25** Configuration Associations Diagram



## Prototype Associations

The Prototype Associations Diagram illustrates the associations between the Prototype, PrototypeProperty, and AssociationProperty metadata types.

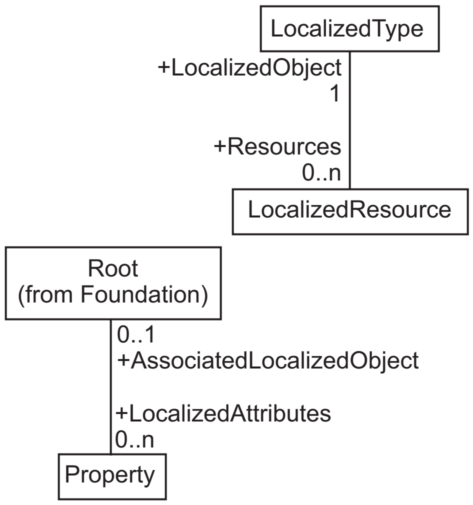
**Figure 5.26** Prototype Associations Diagram



## Locale Associations

The Locale Associations Diagram illustrates the associations between the LocalizedType and LocalizedResource metadata types, and the Root and Property metadata types.

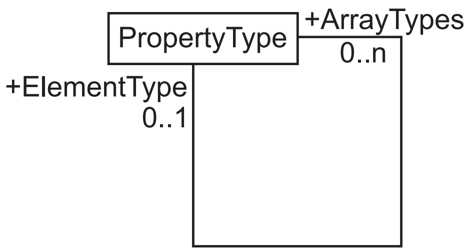
**Figure 5.27** Locale Associations Diagram



## PropertyType Array Associations

The PropertyType Array Associations Diagram illustrates the associations between PropertyType objects.

**Figure 5.28** PropertyType Array Associations Diagram

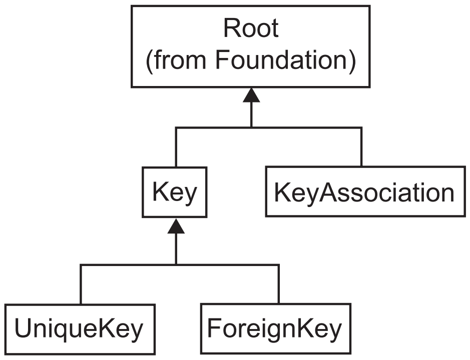


## Diagrams for Relational Metadata Types

### Key Hierarchy

The Key Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent relational keys in the SAS Metadata Model.

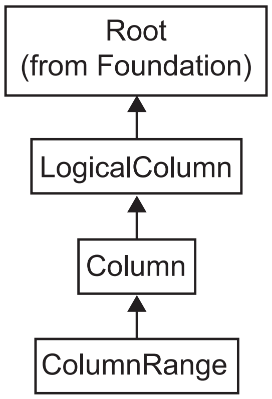
**Figure 5.29** Key Hierarchy Diagram



## Column Hierarchy

The Column Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent relational columns.

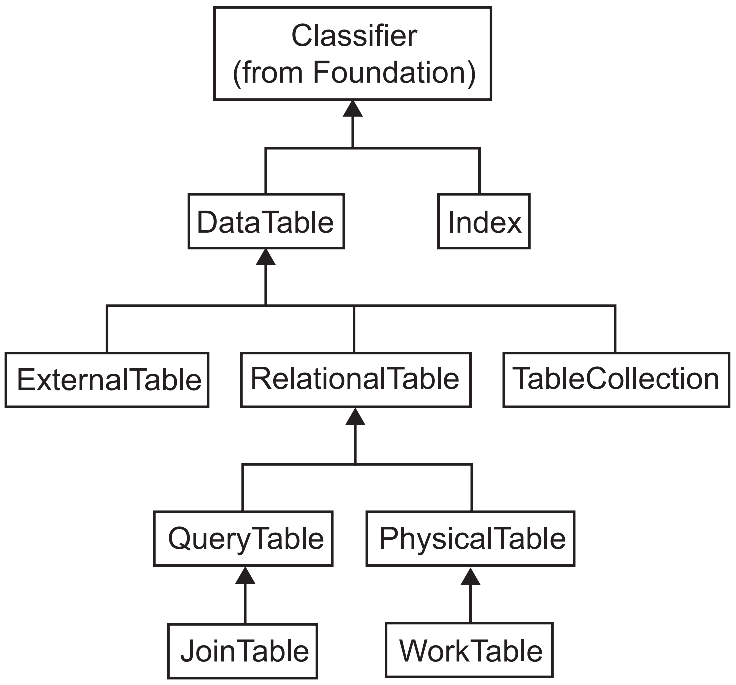
**Figure 5.30** Column Hierarchy Diagram



## Table Hierarchy

The Table Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent tables.

**Figure 5.31** Table Hierarchy Diagram

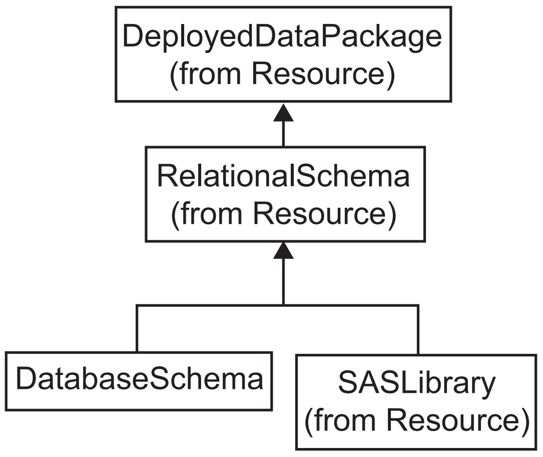


DataTable is an abstract metadata type that models the properties for three classes of subtypes: ExternalTable, RelationalTable, and TableCollection. RelationalTable has as subtypes QueryTable, PhysicalTable, JoinTable, and WorkTable, which also inherit from each other.

## DeployedDataPackage Hierarchy

The DeployedDataPackage Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent a relational schema.

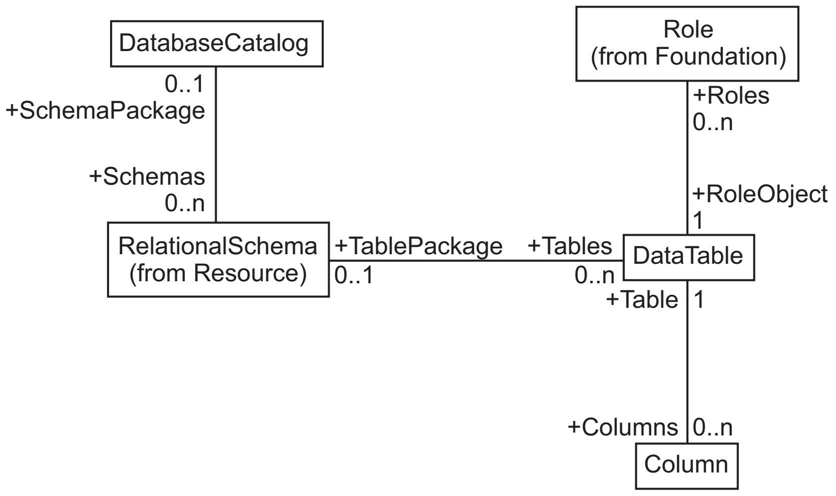
**Figure 5.32** DeployedDataPackage Hierarchy Diagram



## 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.

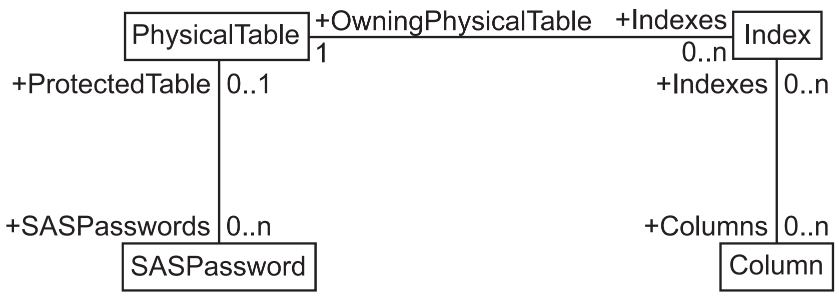
**Figure 5.33** Schema, Table, Role, and Column Associations Diagram



## Table, Password, and Index Associations

The Table, Password, and Index Associations Diagram illustrates the associations between the PhysicalTable, SASPassword, Index, and Column metadata types.

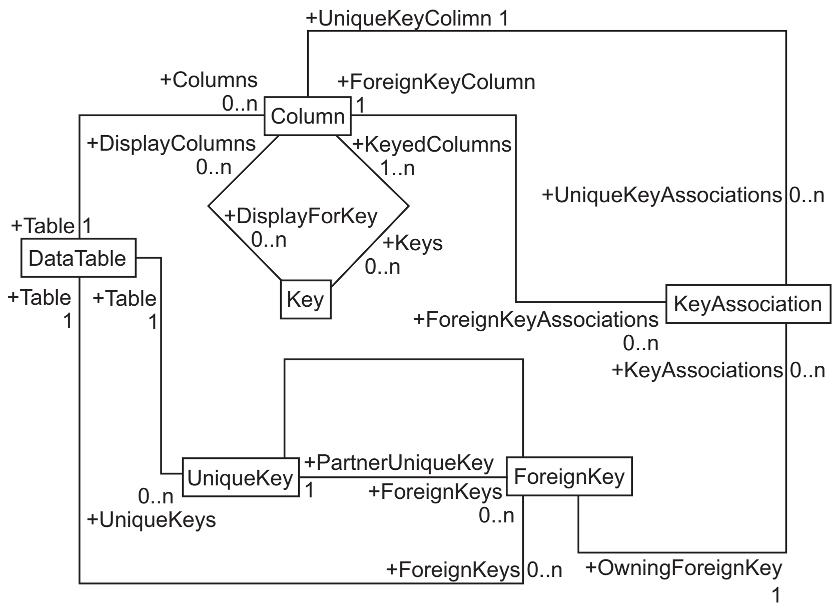
**Figure 5.34** Table, Password, and Index Associations Diagram



## Table and Key Associations

The Table and Key Associations Diagram illustrates the associations between DataFile, Column, Key and KeyAssociation metadata types.

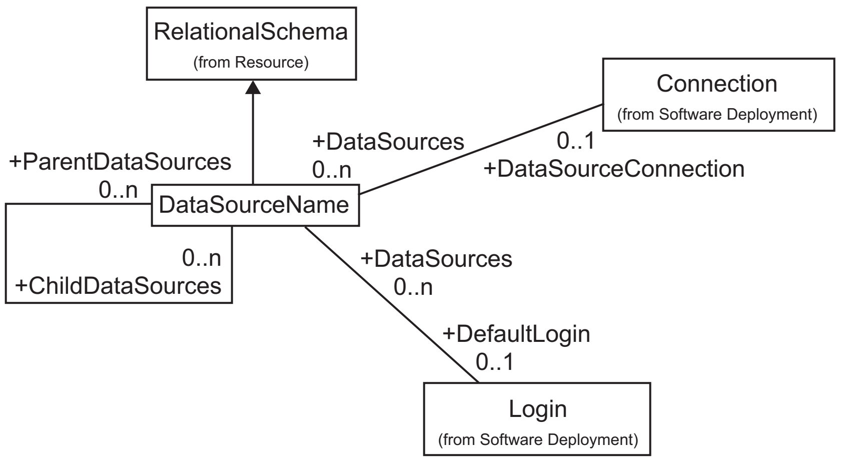
**Figure 5.35** Table and Key Associations Diagram



## Threaded Kernel Table Services Data Source Name Associations

The Threaded Kernel Table Services Data Source Name Associations Diagram illustrates the associations between RelationalSchema, DataSourceName, Login, and Connection metadata types.

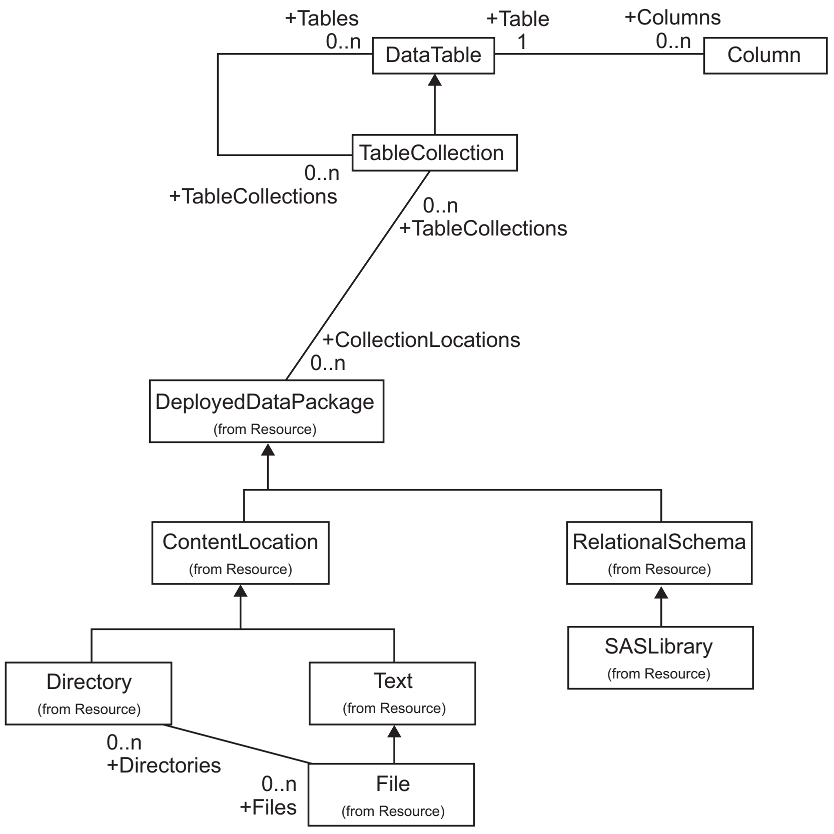
**Figure 5.36** Threaded Kernel Table Services Data Source Name Associations Diagram



## 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.

**Figure 5.37** Table Collection Associations Diagram

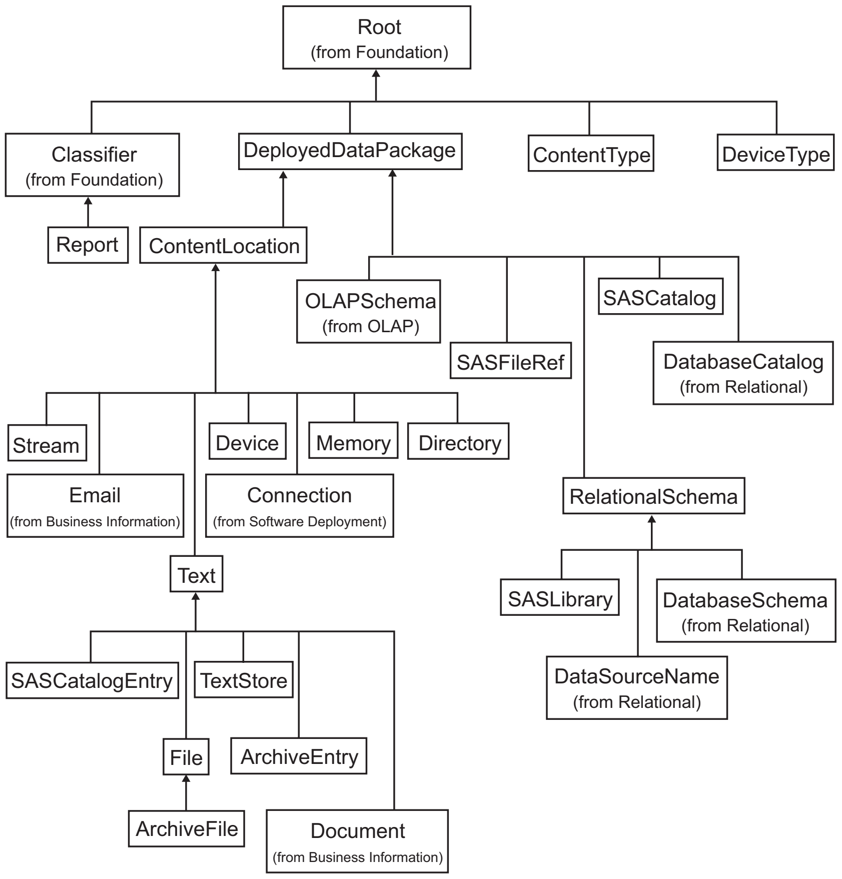


## Diagrams for Resource Metadata Types

### Resource Hierarchy

The Resource Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent data resources in the SAS Metadata Model.

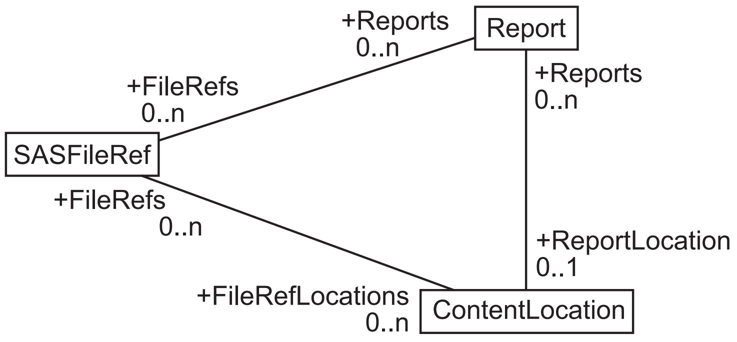
**Figure 5.38** Resource Hierarchy Diagram



## Report, SASFileRef Associations

The Report and SASFileRef Associations Diagram illustrates the associations between the Report, SASFileRef, and ContentLocation metadata types.

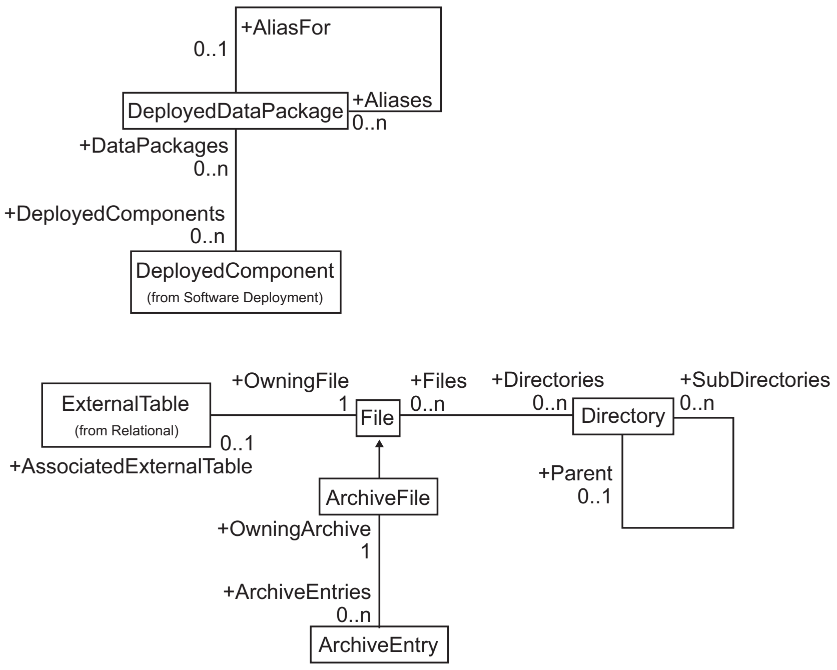
**Figure 5.39** Report and SASFileRef Associations Diagram



## 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.

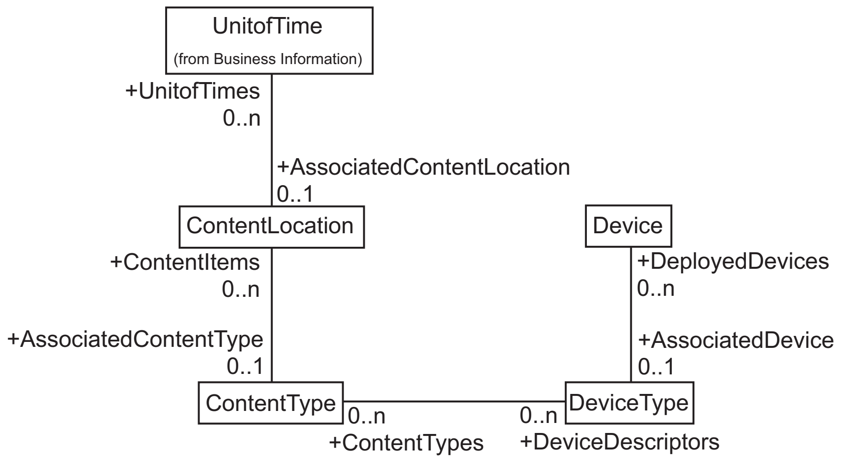
**Figure 5.40** DeployedDataPackage and File Associations Diagram



## 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.

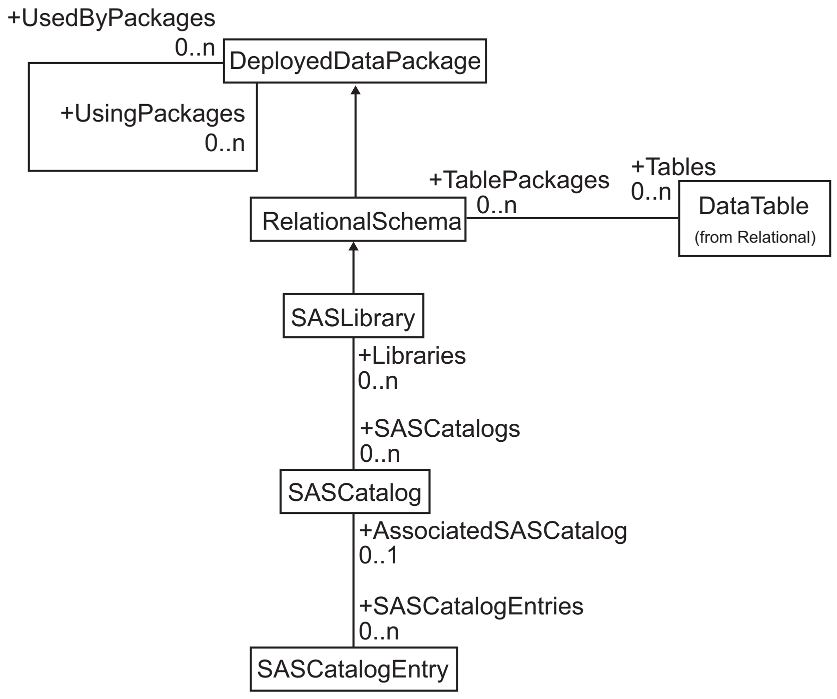
**Figure 5.41** Resource Content Type, Devices Associations Diagram



## SASLibrary, SASCatalog Associations

The SASLibrary, SASCatalog Associations Diagram illustrates the associations between the DeployedDataPackage, RelationalSchema, DataTable, SASLibrary, SASCatalog, and SASCatalogEntry metadata types.

**Figure 5.42** SASLibrary, SASCatalog Associations Diagram

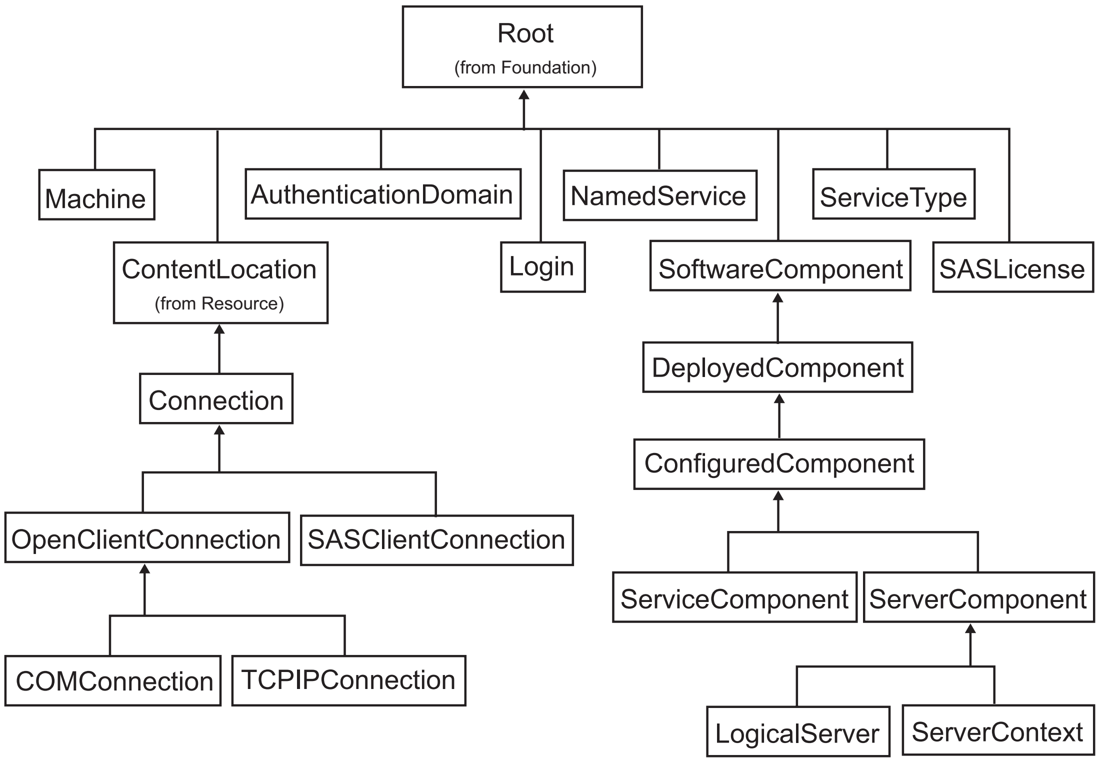


## Diagrams for Software Deployment Metadata Types

### Software Deployment Hierarchy

The Software Deployment Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to describe software, servers, and connection information.

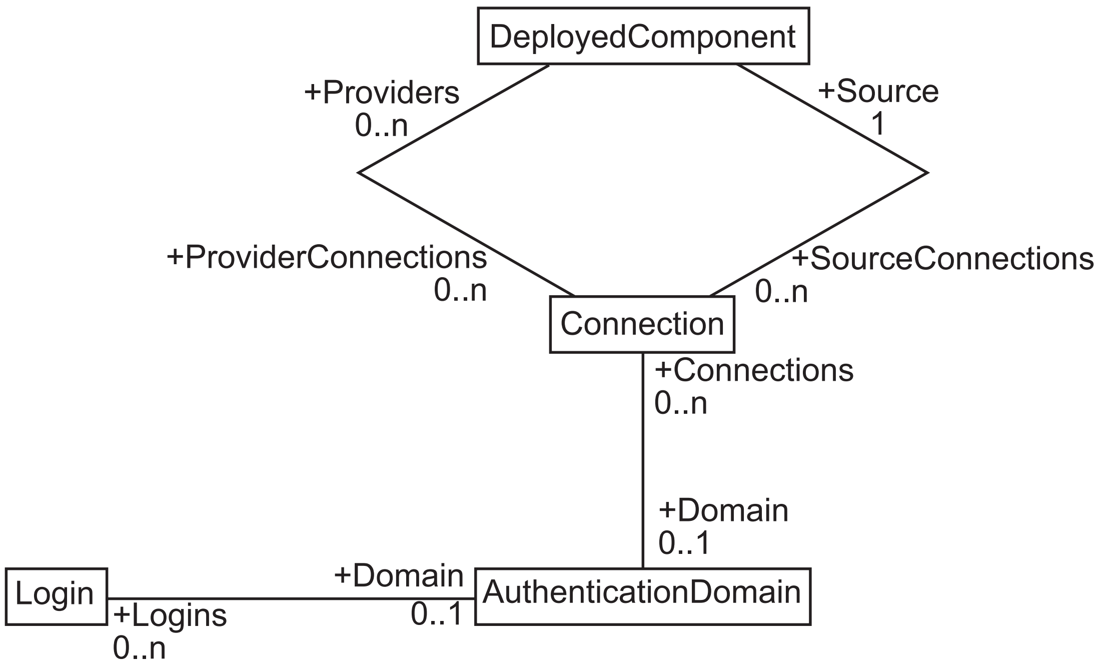
**Figure 5.43** Software Deployment Hierarchy Diagram



### Login, DeployedComponent Associations

The Login and DeployedComponent Associations Diagram illustrates the associations between the Login, DeployedComponent, Connection, and AuthenticationDomain metadata types.

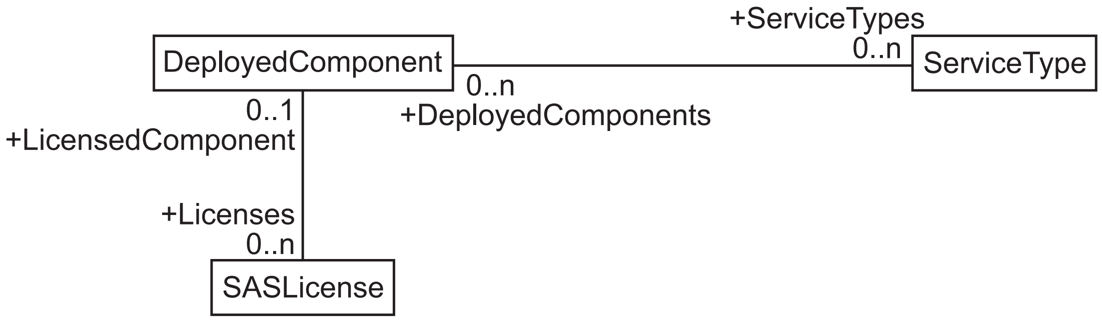
**Figure 5.44** Login and DeployedComponent Associations Diagram



## 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.

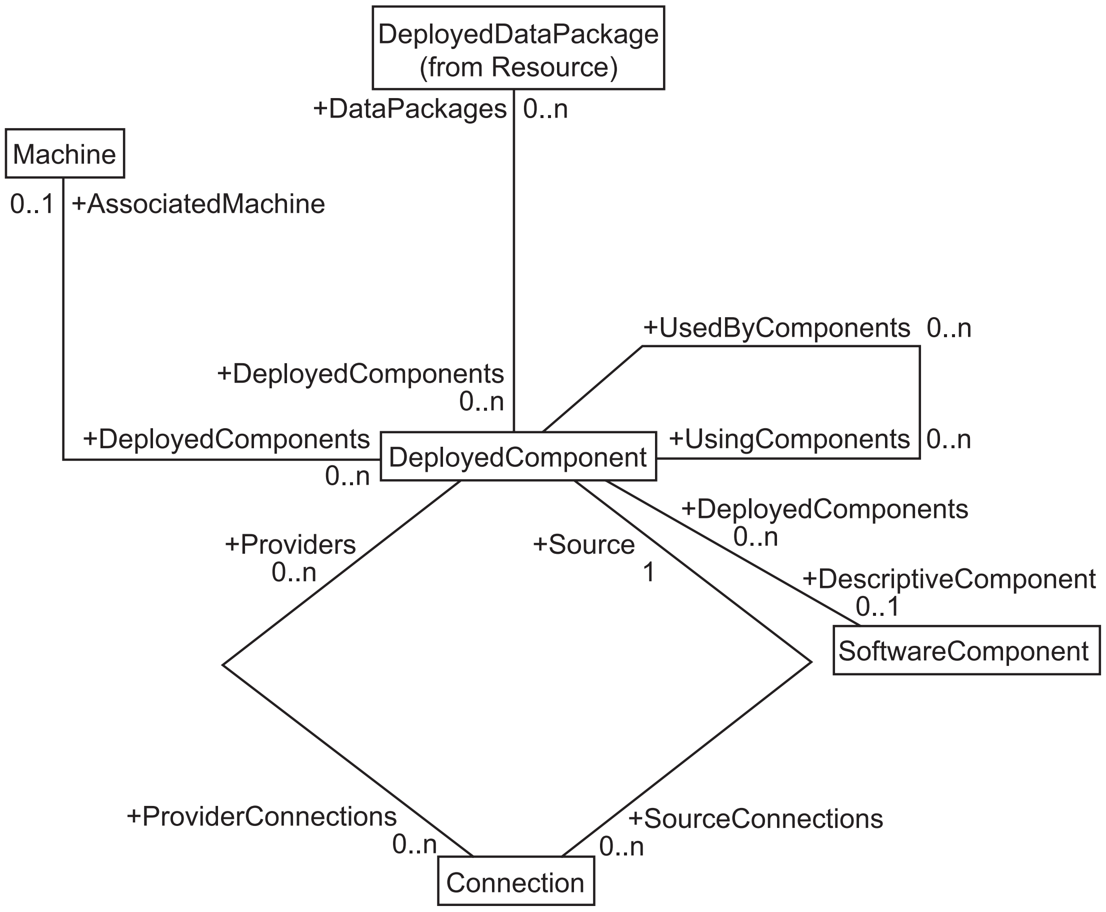
**Figure 5.45** DeployedComponent, ServiceType Association Diagram



## DeployedComponent Associations

The DeployedComponent Associations Diagram illustrates the associations between the DeployedComponent, Machine, DeployedDataPackage, SoftwareComponent, and Connection metadata types.

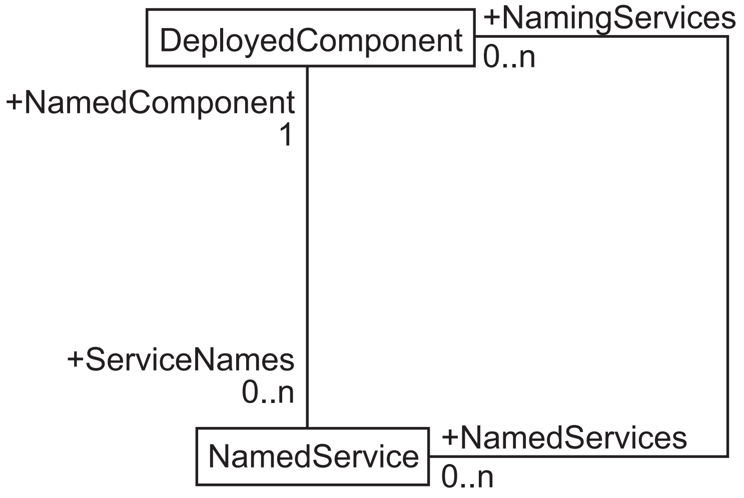
**Figure 5.46** DeployedComponent Associations Diagram



## DeployedComponent, NamedService Association

The DeployedComponent and NamedService Association Diagram illustrates the associations between the DeployedComponent and NamedService metadata types.

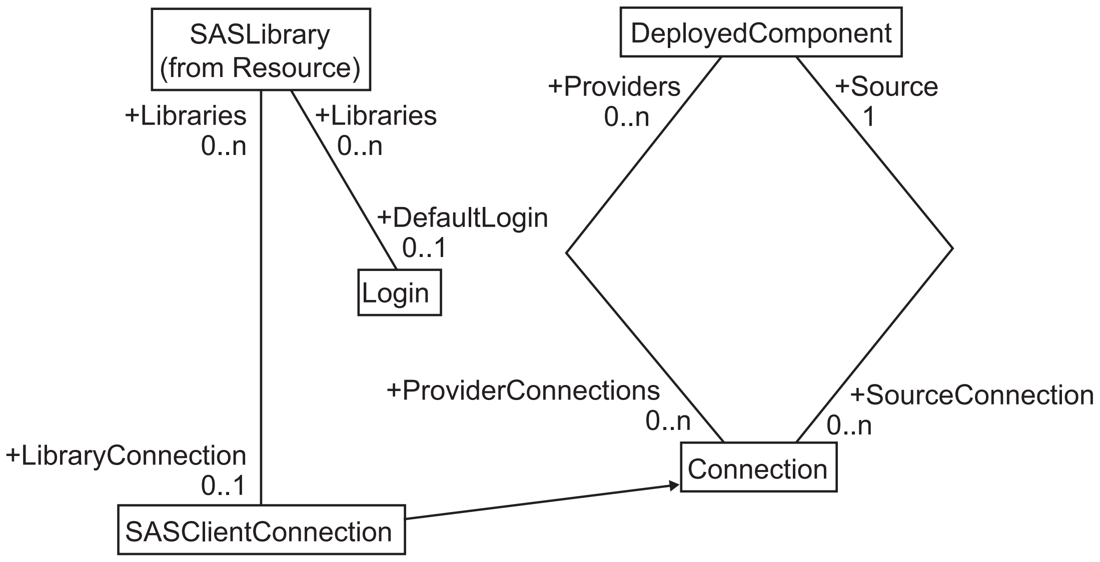
**Figure 5.47** DeployedComponent and NamedService Association Diagram



## SASLibrary, Database Associations

The SASLibrary and Database Associations Diagram illustrates the associations between the SASLibrary, SASClientConnection, Login, Connection, and DeployedComponent metadata types.

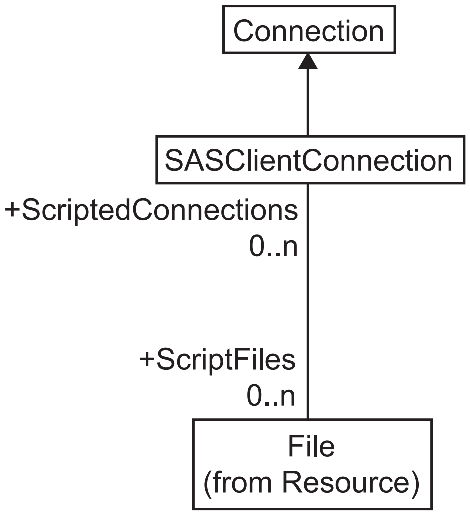
**Figure 5.48** SASLibrary and Database Associations Diagram



## SoftwareComponent, Root Association

The SoftwareComponent and Root Associations Diagram illustrates the associations between the SoftwareComponent and Root metadata types.

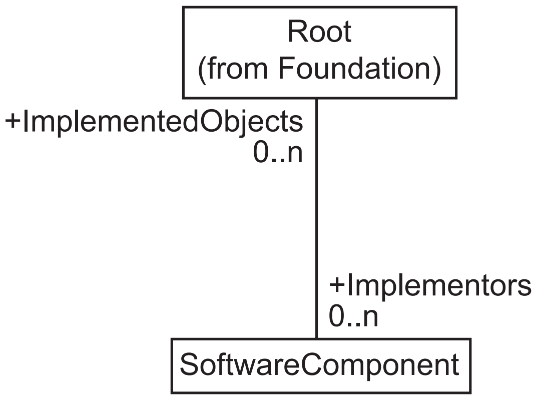
**Figure 5.49** SoftwareComponent and Root Associations Diagram



## Connection, Script File Associations

The Connection and Script File Associations Diagram illustrates the associations between the Connection, SASClientConnection, and File metadata types.

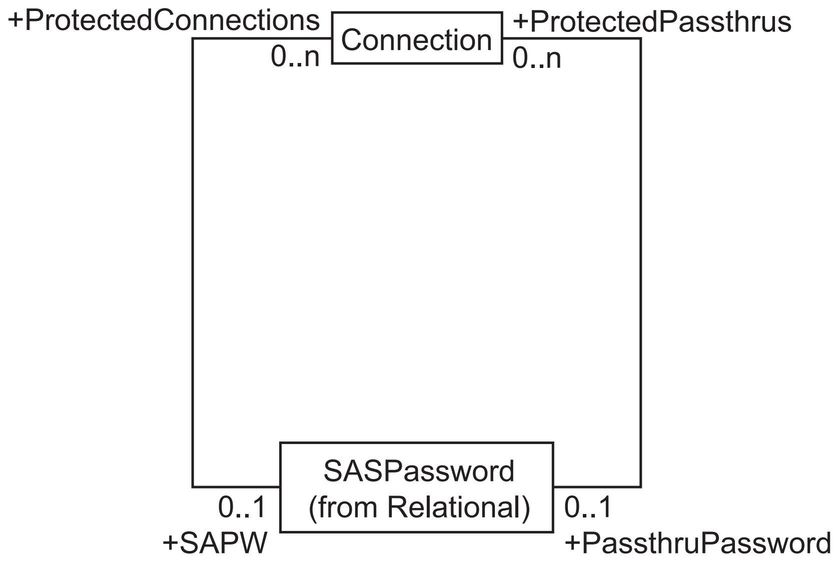
**Figure 5.50** Connection and Script File Associations Diagram



## Connection, SASPassword Associations

The Connection and SASPassword Associations Diagram illustrates the associations between the Connection and SASPassword metadata types.

**Figure 5.51** Login and DeployedComponent Associations Diagram

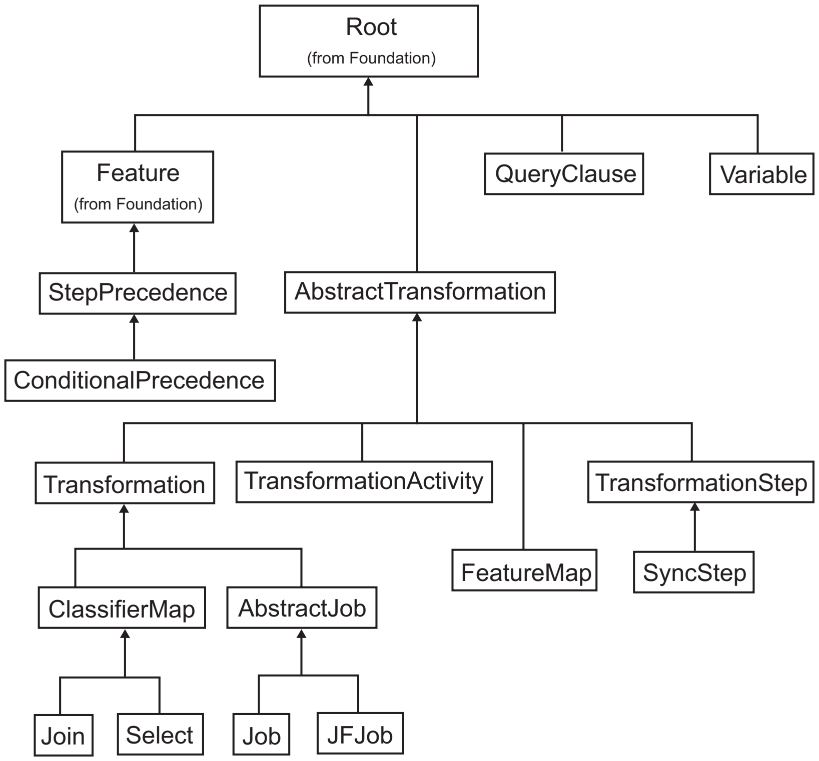


## Diagrams for Transformation Metadata Types

### Transformation Hierarchy

The Transformation Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to describe transformation of data.

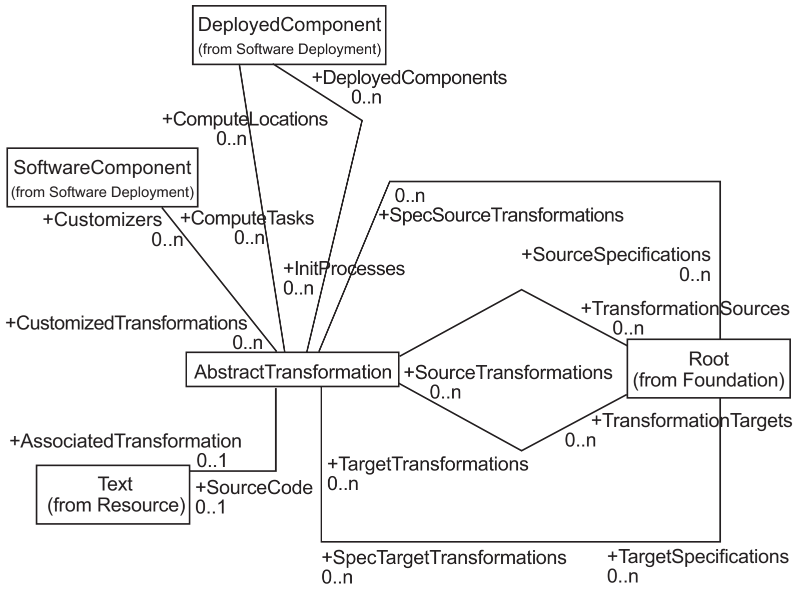
**Figure 5.52** Transformation Hierarchy Diagram



## Root, Transformation Associations

The Root and Transformation Associations Diagram illustrates the associations between the Root, AbstractTransformation, DeployedComponent, SoftwareComponent, and Text metadata types.

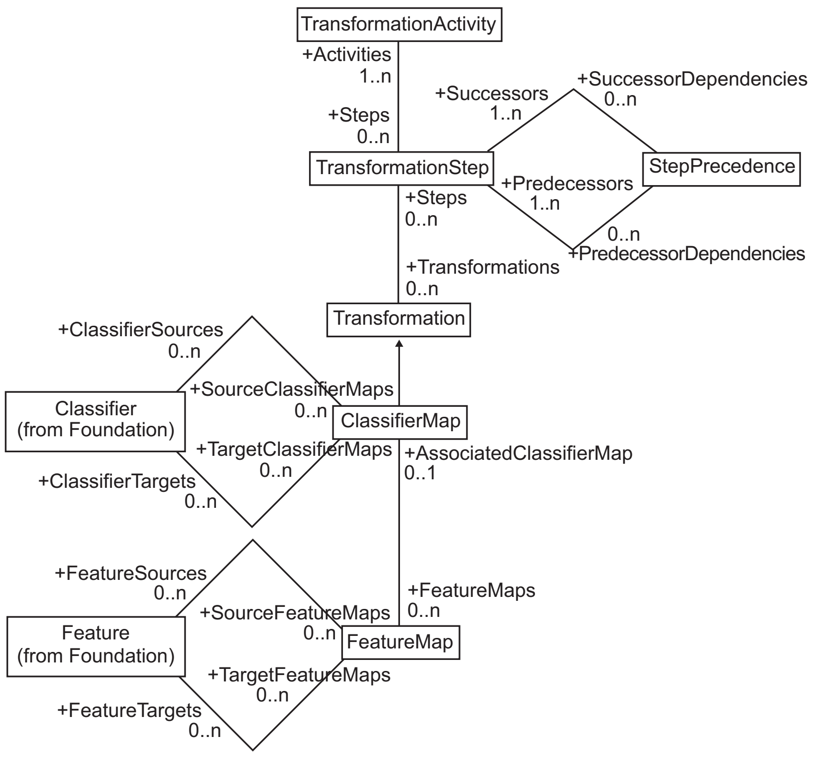
**Figure 5.53** Root and Transformation Associations Diagram



## Transformation Associations Overview

The Transformation Associations Overview Diagram illustrates the associations between the TransformationActivity, TransformationStep, StepPrecedence, Transformation, ClassifierMap, Classifier, FeatureMap, and Feature metadata types.

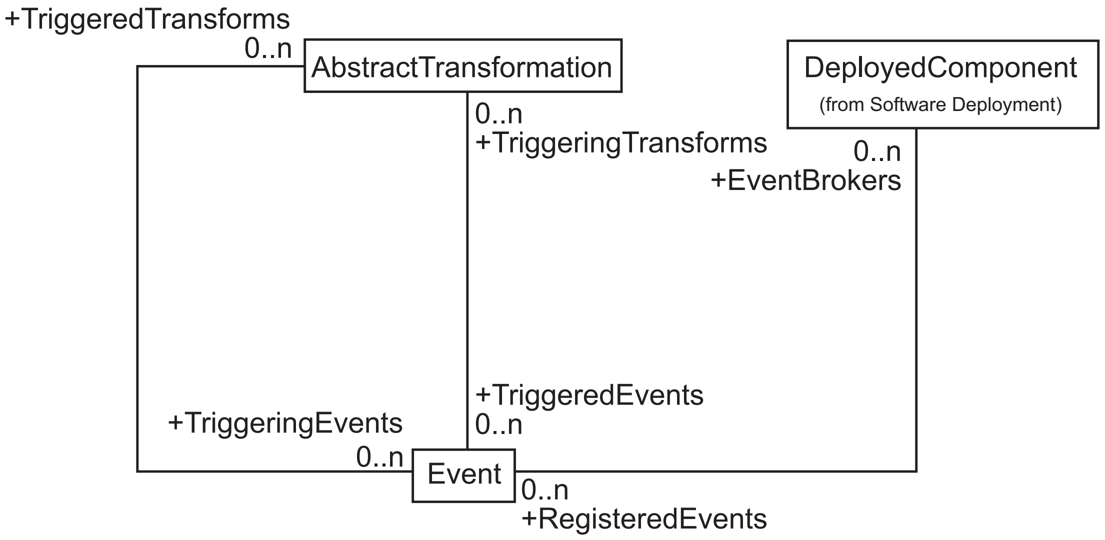
**Figure 5.54** Transformation Associations Overview Diagram



## Event Associations

The Event Associations Diagram illustrates the associations between the AbstractTransformation, DeployedComponent, and Event metadata types.

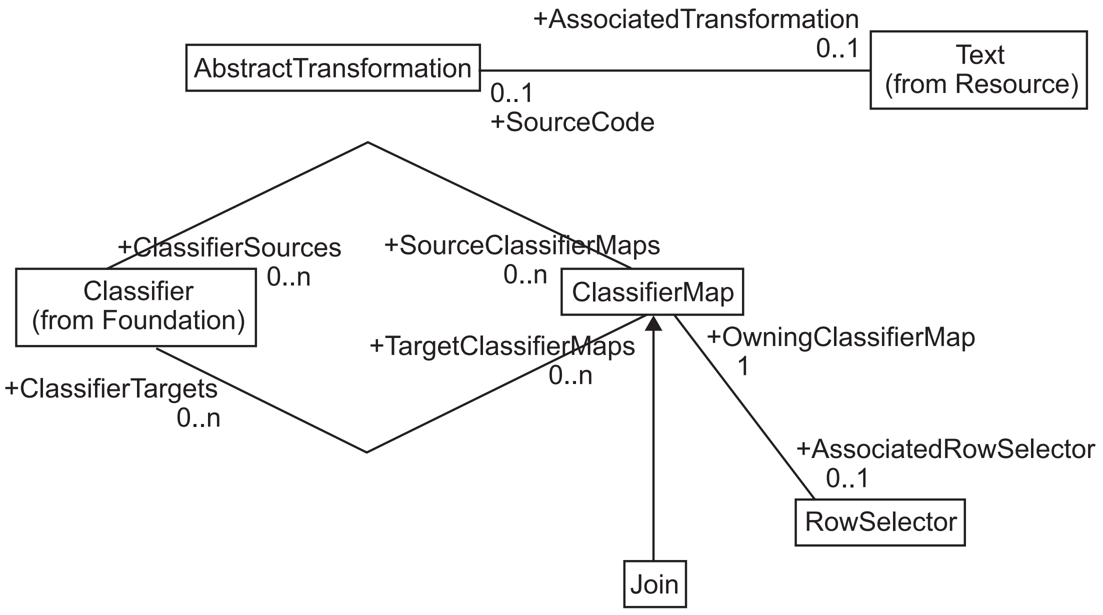
**Figure 5.55** Event Associations Diagram



## ClassifierMap Associations

The ClassifierMap Associations Diagram illustrates the associations between the AbstractTransformation and Text, and Classifier, ClassifierMap, RowSelector, and Join metadata types.

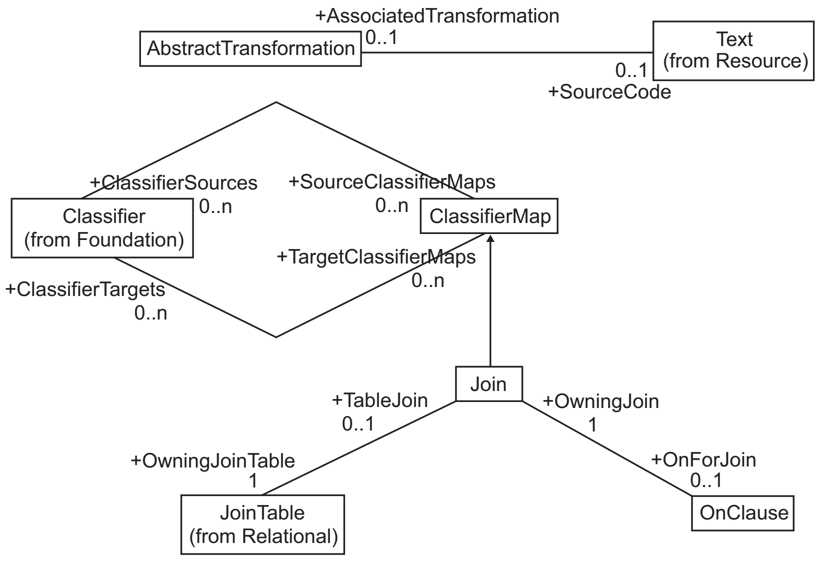
**Figure 5.56** ClassifierMap Associations Diagram



## Join Associations

The Join Associations Diagram illustrates the associations between the AbstractTransformation and Text, and Classifier, ClassifierMap, Join, JoinTable, and OnClause metadata types.

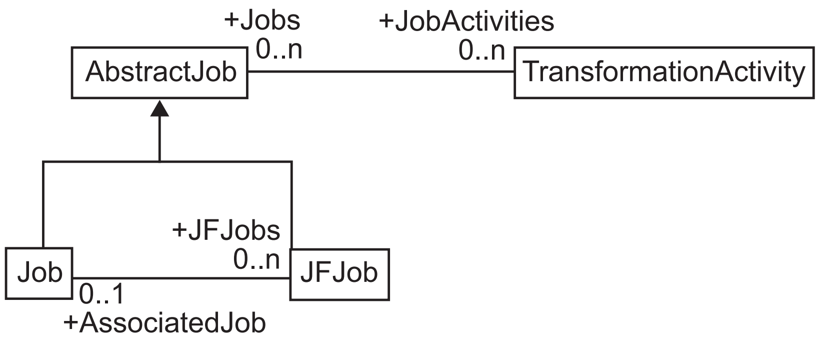
**Figure 5.57** Join Associations Diagram



## Job Associations

The Job Associations Diagram illustrates the associations between the AbstractJob, TransformationActivity, Job, and JFJob metadata types.

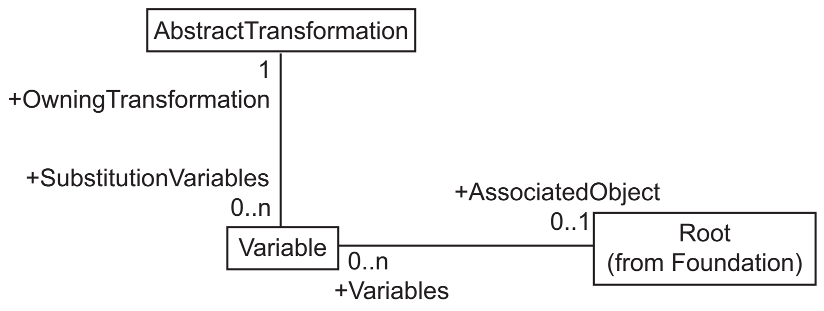
**Figure 5.58** Job Associations Diagram



## Variable Associations

The Variable Associations Diagram illustrates the associations between the Root, Variable, and AbstractTransformation metadata types.

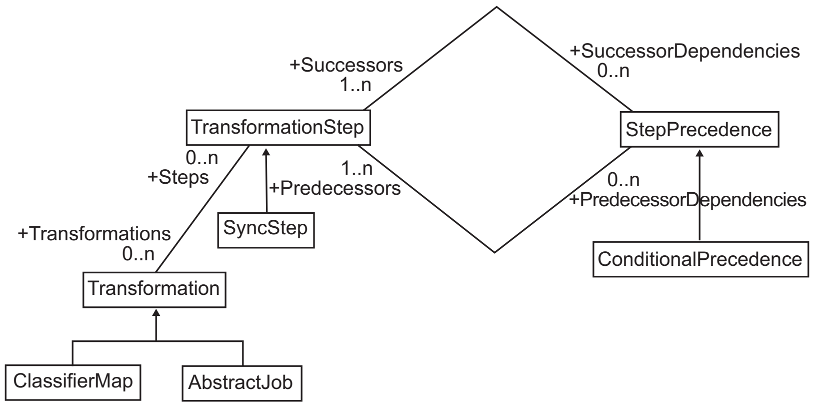
**Figure 5.59** Variable Associations Diagram



## 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.

**Figure 5.60** Workflow Associations Diagram

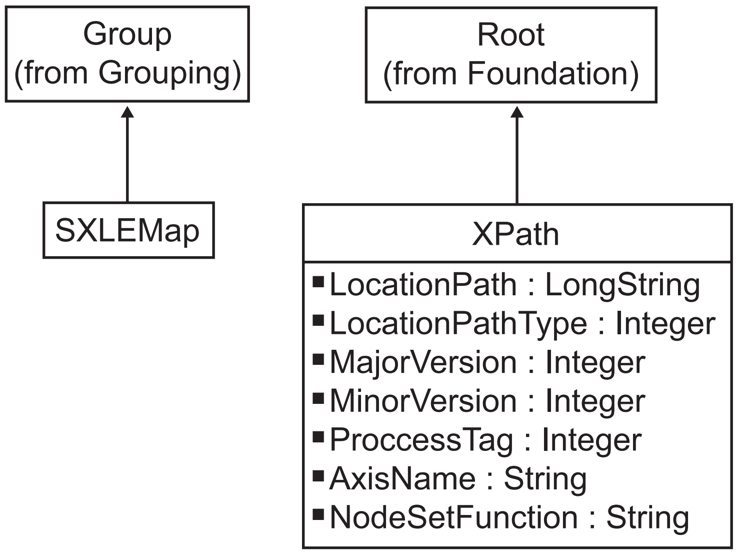


# Diagrams for XML Metadata Types

## XML Hierarchy

The XML Hierarchy Diagram depicts the hierarchy of the metadata types that are defined to represent XML structures.

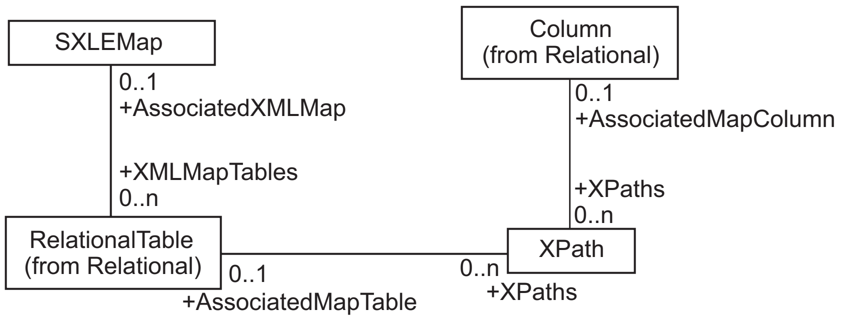
**Figure 5.61** XML Hierarchy Diagram



## XML Associations

The XML Associations Diagram illustrates the associations between the SXLEMap, RelationalTable, XPath, and Column metadata types.

**Figure 5.62** XML Associations Diagram



# CHAPTER 6 REPOS Namespace Metadata Types

## Overview of REPOS Namespace Metadata Types

The REPOS namespace contains metadata types that 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 by using IOMI class methods, just like you can those of any application-related metadata object. You can also issue IServer class methods to "control" a repository, for example, to temporarily change a repository's state in preparation for a backup.

A relationship between repositories must exist before users of the repositories can create cross-repository references between the objects in those repositories. A relationship between repositories is also 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 Open Metadata Interface: User's Guide*. For information about the change management facility, see "Using the Change Management Facility" in the user's guide. For information about repository auditing, see "Invoking a Repository Audit Trail" in the user's guide.

When using IOMI class methods on repository objects, 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.

This is the hierarchy of the repository types:

- "Repository" on page 108 (Abstract Class for Metadata Repositories)
- "RepositoryBase" on page 109 (Metadata Type for SAS Repositories)

# Repository

Description: 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. In the current release, the only subtype is RepositoryBase.

## Attributes

| AttributeName | Description | Type | Length |
|---|---|---|---|
| Desc | A user-defined description of the repository | String | 200 |
| Id | The repository's unique, system-generated identifier | String | 15 |
| MetadataCreated | The date and time the metadata was created | Double | |
| MetadataUpdated | The date and time the metadata was last updated | Double | |
| Name | A user-defined, unique, logical name assigned to the repository | 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>
```

# RepositoryBase

Description: The 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 unique, system-generated identifier. | String | 17 | No | No |
| Name | A user-defined, unique, logical name assigned to the repository. | String | 60 | Yes | Yes |

| Attribute Name | Description | Type | Length | Reqd for Add | Update Allowed |
|---|---|---|---|---|---|
| Desc | A user-defined description of the repository. | String | 200 | No | Yes |
| Access | The repository'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, and DB2. | String | 10 | No | Yes |
| Options | SAS library options for this repository. See "Creating a Repository on an External DBMS" in the *SAS Open Metadata Interface: User's Guide* for a list of required options. | 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 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. | String | 15 | No | Yes |
| AuditType | Turns auditing on and off and specifies the type of auditing that will be performed. 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 of all transaction 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 |

| Attribute Name | Description | Type | Length | Reqd for Add | Update Allowed |
|---|---|---|---|---|---|
| AuditOptions | SAS library options for the audit trail. This option is reserved for future use. | String | 200 | No | No |
| MetadataCreated | The date and time the repository was created. | Double | | No | No |
| MetadataUpdated | The date and time the repository's properties were 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 Partner Name: RepositoryBase | The repositories that depend on this repository. | 0 to * | No | No | Yes |
| Name: DependencyUses Subelements: Repository Partner Name: RepositoryBase | The repositories on which this repository depends. | 0 to * | No | No | Yes |

## RepositoryBase 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 the default access mode of a repository read-only, set Access = 1. To temporarily change a repository's access mode, issue the Pause method to modify its PauseState attribute.

To grant specific users ReadMetadata and WriteMetadata permission to a repository, specify values in the repository's default access control template by using SAS Management Console. For more information, see the Help for SAS Management Console.

## Using DeleteMetadata

You can delete the metadata in a repository or the metadata and the repository's registration. For more information, see "DeleteMetadata" on page 141. Also see "Clearing or Deleting a Repository" in "Repository Maintenance Tasks" in the *SAS 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. PauseState is a transient value that is not retained across instances of the server. 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. Use the GetRepositories method to determine a repository's current Access and PauseState attribute values. For more information, see "GetRepositories" on page 156, "Pause" on page 182, and "Resume" on page 186.

P A R T 3

# Method Classes

# CHAPTER 7
# Methods for Reading and Writing Metadata (IOMI Class)

## Overview of the IOMI Class Methods

The SAS Open Metadata Interface defines a set of methods that read and write metadata types (the IOMI class), a set of methods for administering the SAS Metadata Server (the IServer class), and a set of methods for requesting authorization decisions from the authorization facility (the ISecurity class). This section describes the methods for reading and writing metadata.

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. Except for 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 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 specify flags and additional options to use when processing the request. The IOMI class supports two call interfaces for passing method parameters to the metadata server. These interfaces are described in "Call Interfaces" on page 13.

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" on page 118.

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
: are nested subelements that describe the relationship between the metadata type named in the main XML element and another metadata type. For example:

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

In the example, the first nested element, Columns, is the *association name*. The association name is a label that describes the relationship between the main element (PhysicalTable) and the second nested element (Column). The second nested element is known as 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 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:
'<' = &lt;

’>’ = &gt;
’&’ = &amp;

## Identifying Metadata

The documentation for many of the metadata types mentions “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 user-defined name assigned to the metadata object. In the previous example, the Name of the table column is "New Column". |
| Id | refers to the unique identifier of the 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 system-generated value that is assigned to a particular metadata repository when it is created. Each application using the SAS Open Metadata Architecture has one or more repositories. The *reposid* is a unique character string that identifies the metadata repository that stores the object.

The *instanceid* is system-generated value that is assigned to a particular metadata 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. △

## Functional Index to IOMI Methods

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

| Category | Method | Description |
|---|---|---|
| Change Management | 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 |
| General Management | 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 | GetMetadata | Reads specified metadata from a repository |
| | GetMetadataObjects | Lists metadata objects when passed the repository and type |
| Repository | GetRepositories | Lists all metadata repositories available through this server |
| Write | AddMetadata | Adds specified metadata to a repository |
| | DeleteMetadata | Deletes specified metadata from a repository |
| | UpdateMetadata | Updates specified metadata in a repository |
| Messaging | DoRequest | Executes XML-encoded method calls |

## 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 filter metadata retrieval requests. See “Summary Table of IOMI Flags” on page 122 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 (1) flag in a GetMetadata call, specify the number 1 in the *Flags* parameter. To specify more than one flag, add their numeric values together and specify the sum in the *Flags* parameter. For example, OMI_ALL (1) + OMI_SUCCINCT (2048)=2049. This flag combination retrieves 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. When a flag does require it, you must supply this input 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, requires you to specify the search criteria in an <XMLSelect> element in the *Options* parameter. The GetMetadata method OMI_TEMPLATES flag, which enables you to request additional properties for metadata objects, requires that you submit a string identifying those properties in a <Templates> element in the *Options* parameter. Several methods also support an OMI_LOCK_TEMPLATES flag, which enables you to specify the associated metadata objects that are checked out along with a requested metadata object. When this flag is used, you must submit a metadata property string that identifies the associated objects in a <LockTemplates> element in the *Options* parameter. See "Summary Table of IOMI Options" on page 129 for a list of these special XML elements.

## Flag Behavior When Multiple Flags Are Used

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

The properties and any search criteria specified in a template are always applied *in addition to* any properties requested by other GetMetadata parameters.

## Summary Table of IOMI Flags

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

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_ALL | 1 | "GetMetadata" on page 148<br>"GetRepositories" on page 156 | 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" on page 158 | Gets the descendants of the returned subtypes in addition to the subtypes. |
| OMI_ALL_SIMPLE | 8 | "GetMetadata" on page 148 | Gets all of the attributes of the requested object. |
| OMI_CI_DELETE_ALL | 33 | "CheckinMetadata" on page 135 | 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" on page 135 | Keeps copies of checked, fetched, and new metadata objects in a project repository after check-in to a primary repository. |
| OMI_DELETE | 32 | "DeleteMetadata" on page 141 | Deletes the contents of a repository in addition to the repository's registration. |

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_DEPENDENCY_USED_BY | 16384 | "GetMetadata" on page 148<br>"GetMetadataObjects" on page 152 | 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 Open Metadata Interface: User's Guide*. |
| OMI_DEPENDENCY_USES | 8192 | "GetMetadataObjects" on page 152 | 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 Open Metadata Interface: User's Guide*. |
| OMI_GET_METADATA | 256 | "GetMetadataObjects" on page 152 | Executes a GetMetadata call for each object that is returned by the GetMetadataObjects method. |

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_IGNORE_NOTFOUND | 134217728 | "DeleteMetadata" on page 141<br>"UpdateMetadata" on page 167 | 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" on page 148<br>"GetMetadataObjects" on page 152 | 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" on page 148 | Locks the specified object and any associated objects selected by GetMetadata flags and options from update by other users. |
| OMI_LOCK_TEMPLATE | 65536 | "CheckoutMetadata" on page 137<br>"CopyMetadata" on page 139<br>"FetchMetadata" on page 146<br>"GetMetadata" on page 148<br>"UndoCheckoutMetadata" on page 165 | 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. |

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_MATCH_CASE | 512 | "GetMetadataObjects" on page 152 | 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" on page 152 | 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" on page 148 | 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" on page 141 | Deletes the contents of a repository but does not remove the repository's registration from the repository manager. |

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_RETURN_LIST | 1024 | "CheckinMetadata" on page 135<br>"CheckoutMetadata" on page 137<br>"CopyMetadata" on page 139<br>"DeleteMetadata" on page 141<br>"FetchMetadata" on page 146<br>"UndoCheckoutMetadata" on page 165<br>"UpdateMetadata" on page 167 | 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" on page 148<br>"GetTypes" on page 162 | 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 "Overview" of querying metadata in the *SAS Open Metadata Interface: User's Guide*. |

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_TEMPLATE | 4 | "GetMetadata" on page 148 | 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 "Querying Specific Metadata Objects" in the *SAS Open Metadata Interface: User's Guide*. |
| OMI_TRUNCATE | 4194304 | "DeleteMetadata" on page 141 | 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" on page 132<br>"UpdateMetadata" on page 167<br>"DeleteMetadata" on page 141 | Determines whether the client can call this method. |
| OMI_UNLOCK | 131072 | "UpdateMetadata" on page 167 | Unlocks an object lock that is held by the calling user. |

| Flag name | Constant | Method | Description |
|---|---|---|---|
| OMI_UNLOCK_FORCE | 262144 | "UpdateMetadata" on page 167 | Unlocks an object lock that is held by another user. |
| OMI_XMLSELECT | 128 | "GetMetadataObjects" on page 152 | 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 "Querying All Metadata of a Specified Type" in the *SAS Open Metadata Interface: User's Guide*. |

## Summary Table of IOMI Options

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

| Option | Flag | Method | Description |
|---|---|---|---|
| <DOAS> | None | "AddMetadata" on page 132<br>"DeleteMetadata" on page 141<br>"GetMetadata" on page 148<br>"GetMetadataObjects" on page 152<br>"GetSubtypes" on page 158<br>"GetTypeProperties" on page 160<br>"IsSubtypeOf" on page 164<br>"UpdateMetadata" on page 167 | Specifies an alternate calling identity for a metadata request. For more information, see "<DOAS> Option" on page 131. |
| <LockTemplates> | OMI_LOCK_TEMPLATE<br>(65536) | "CheckoutMetadata" on page 137<br>"FetchMetadata" on page 146<br>"CopyMetadata" on page 139<br>"GetMetadata" on page 148<br>"UndoCheckoutMetadata" on page 165 | For GetMetadata, specifies associated objects to be locked instead of those requested by method flags and options. For other methods, overrides the default lock template with a user-supplied lock template. |
| <Reposid> | OMI_SUCCINCT (2048) | GetTypes | Specifies a repository ID for which to list metadata types. See "Listing the Metadata Types in a Repository" in "Overview" of querying metadata in the *SAS Open Metadata Interface: User's Guide*. |

| Option | Flag | Method | Description |
|---|---|---|---|
| <Templates> | OMI_TEMPLATE (4) | "GetMetadata" on page 148 | Specifies properties to retrieve for an object, in addition to those specified in the *inMetadata* parameter. See "Using Templates" in "Querying Specific Metadata Objects" in the *SAS Open Metadata Interface: User's Guide*. |
| <XMLSelect> | OMI_XMLSELECT (128) and optionally OMI_MATCH_CASE (512) | "GetMetadataObjects" on page 152 | Specifies a search string to filter the objects that are retrieved. See "Filtering a GetMetadataObjects Request" in "Querying All Metadata of a Specified Type" in the *SAS Open Metadata Interface: User's Guide*. |

## <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 user. 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 user. It is the responsibility of the client to authenticate the user.

### 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 client must have trusted user status on the metadata server in order to issue the ISecurity getCredentials method.* 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 "User and Group Management" in the *SAS Intelligence Platform: Security Administration Guide*.

## Example 1: Standard Interface

The following is an example of a GetMetadataObjects request 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.

## 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 | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata property string that defines the properties of the metadata object. |
| reposid | C | in | The ID of the repository to which the metadata is to be added. |
| outMetadata | C | out | The metadata string returned by the metadata server. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for the method. This method supports one flag, which is required: OMI_TRUSTED_CLIENT=268435456 Specifies that the client can call this method. |
| options | C | in | The indicator for options. This method currently supports one option: <DOAS Credential="*credHandle*"/> Optionally specifies an alternate calling identity for the request. For more information, see "<DOAS> Option" on page 131. |

## 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" on page 118 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 define the objects indicate the type of operation that is to be performed. For more information, see "Adding Metadata" in the *SAS Open Metadata Interface: User's Guide*.

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

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 program-specific examples, see "Program-Specific AddMetadata Examples" on page 193.

## 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" on page 139
- □ "UpdateMetadata" on page 167
- □ "GetRepositories" on page 156

# 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 | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata string that identifies the metadata objects to check in. Reserved for future use. |
| outMetadata | C | out | The metadata string returned from the server. |
| projectReposid | C | in | The ID of the project repository. |
| changeName | C | in | A name for the output Change object. |
| changeDesc | C | in | A description of the metadata updates. |
| changeId | C | in | The identifier of an existing Change object. Reserved for future use. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed.<br>OMI_CI_DELETE_ALL=33<br>Deletes fetched as well as checked and new metadata objects from a project repository on check-in to a primary repository.<br>OMI_CI_NODELETE=524288<br>Keeps copies of checked, fetched, and new metadata objects in a project repository after check-in to a primary repository.<br>OMI_RETURN_LIST=1024<br>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:* In the current release, CheckinMetadata does not support check-in of individual objects. A CheckinMetadata method 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 enable 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.

CheckinMetadata does not place a limit on the number of characters that you can specify in the *changeDesc* parameter. The method stores the first 200 characters in the Desc= attribute of the Change object and stores the complete description text in a TextStore object. A GetMetadata request on the Desc= attribute of the Change object will not automatically retrieve the text in the TextStore. In order to retrieve the complete description text, you must issue a GetMetadata request directly on the TextStore object. The TextStore object has the same Name as the Change object, and is associated to the Change object via the Notes association.

The default behavior of the CheckinMetadata method is to delete checked and new metadata objects from the project repository when the objects are checked into the primary repository. Fetched objects are not deleted from the project repository. Set OMI_CI_DELETE_ALL (33) to additionally delete fetched objects. Set the OMI_CI_NODELETE (524288) flag to keep copies of the all objects in the project repository.

Be sure to check the return code of a CheckinMetadata 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.

For more information about primary repositories, project repositories, and Change objects, see “Using the Change Management Facility” in the *SAS 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" on page 137
- "UndoCheckoutMetadata" on page 165
- "FetchMetadata" on page 146

## 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 | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata string identifying an object to be locked and copied. |
| outMetadata | C | out | The metadata string returned from the server. |
| projectReposid | C | in | The ID of the target project repository. |
| ns | C | in | The namespace to use as the context for the request. |

| Parameter | Type | Direction | Description |
|---|---|---|---|
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed.<br>OMI_LOCK_TEMPLATE=65536<br>Overrides the default lock template with a user-defined template submitted in the *Options* parameter.<br>OMI_RETURN_LIST=1024<br>Includes the object identifiers of any associated objects that were checked out in the *outMetadata* parameter. |
| options | C | in | The indicator for options. This method supports one option:<br>&lt;LockTemplates&gt; Specifies a user-defined lock template. The &lt;LockTemplates&gt; 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 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, see “Using the Change Management Facility” in the *SAS Open Metadata Interface: 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 in the *outMetadata* parameter, set the OMI_RETURN_LIST (1024) flag.

Be sure to check the return code of a CheckoutMetadata 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.

## 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" on page 165
- □ "CheckinMetadata" on page 135
- □ "FetchMetadata" on page 146

## CopyMetadata

Copies metadata objects from one metadata repository to another
Category: Change management

## Syntax

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

## Parameters

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata string that identifies the metadata object to be copied. |
| outMetadata | C | out | The metadata string returned from the metadata server. |
| targetReposid | C | in | The ID of the target repository. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed.<br>OMI_LOCK_TEMPLATE=65536<br>Overrides the default lock template with a user-supplied template submitted in the <LockTemplates> element in the *Options* parameter.<br>OMI_RETURN_LIST=1024<br>Returns the object identifiers of any associated objects that were copied in the *outMetadata* parameter. |
| options | C | in | An indicator for options. This method supports one option:<br><LockTemplates><br>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. CopyMetadata 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 Open Metadata Interface: User's Guide*. You can override the default lock template and copy a different set of associated objects by setting the OMI_LOCK_TEMPLATE (65536) flag and supplying an alternate lock template in a <LockTemplates> element in the *Options* parameter. For more information, see "Using the Change Management Facility" in the *SAS Open Metadata Interface: User's Guide*.

Set the OMI_RETURN_LIST (1024) flag to include the associated objects that were copied in the output returned in 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.

Be sure to check the return code of a CopyMetadata 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.

## 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” on page 132

# 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 | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata property string that identifies the metadata object to be deleted. |
| outMetadata | C | out | The metadata property string returned from the metadata server. For this method, the output string 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 | The flags supported for this method. The OMI_TRUSTED_CLIENT flag is required; all other flags are optional.<br><br>OMI_DELETE=32<br>Deletes the contents of a repository in addition to the repository's registration.<br><br>OMI_IGNORE_NOTFOUND=134217728<br>Prevents a delete operation from being aborted when a request specifies to delete an object that does not exist.<br><br>OMI_PURGE=1048576<br>Removes previously deleted metadata from a repository without disturbing the current metadata objects.<br><br>OMI_REINIT=2097152<br>Deletes the contents of a repository but does not remove the repository's registration from the repository manager.<br><br>OMI_RETURN_LIST=1024<br>Returns a list of deleted object IDs, as well as any cascading object IDs that were deleted.<br><br>OMI_TRUNCATE=4194304<br>Deletes all metadata objects but does not remove the metadata object containers from a repository or change a repository's registration.<br><br>OMI_TRUSTED_CLIENT=268435456<br>Specifies that the client can call this method. |
| options | C | in | Passed indicator for options.<br><br>&lt;DOAS Credential="*credHandle*"/&gt;<br>Specifies an alternate calling identity for the request. For more information, see "&lt;DOAS&gt; Option" on page 131. |

## Details

The DeleteMetadata method handles requests to remove metadata 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 perform these actions by issuing the DeleteMetadata method on a RepositoryBase object in the REPOS namespace. 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. Specifying more than one of these flags will potentially yield undesirable results. For additional information about using REPOS namespace flags, see “Clearing or Deleting a Repository” in “Repository Maintenance Tasks” in the *SAS Open Metadata Interface: User’s Guide*.

Delete requests for dependent 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 with the specified object. There is no need to specify the dependent objects in the deletion request. This is called a cascading delete. If the dependent objects are specified in the DeleteMetadata request, then 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 (134217728) flag, but it is recommended that you avoid specifying dependent objects altogether.

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 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 are 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. Also see “Program-Specific DeleteMetadata Examples” on page 197.

## 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>
<!--- OMI_TRUSTED_CLIENT flag ------>
  <Flags>268436480</Flags>
  <Options/>
</DeleteMetadata>
```

## Related Methods

- "AddMetadata" on page 132
- "UndoCheckoutMetadata" on page 165
- "UpdateMetadata" on page 167

# DoRequest

Executes XML encoded method calls
Category: Message Methods

## Syntax

rc = DoRequest(inMetadata, outMetadata);

## Parameters

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata property 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 | A metadata property 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>
   <Parametern>Value</Parametern>
...
</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>
   <Parametern>Value</Parametern>
</MethodName1>
<MethodName2>
   <Parameter1>Value</Parameter1>
   <Parameter2>Value</Parameter2>
   <Parametern>Value</Parametern>
</MethodName2>
</Multiple_Requests>
```

The published IOMI method 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 in all IOMI method calls that have an *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 output from the request is returned in the DoRequest method's *outMetadata* parameter. 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. The method also supports requests to metadata objects in both the SAS namespace and the REPOS namespace, although both namespaces should not be referenced in the same DoRequest.

Be sure to check the return code of a DoRequest method call. A nonzero return code indicates that a failure occurred while trying to write metadata. When a nonzero return code is returned, none of the changes by any of the methods in the DoRequest will be made.

## 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);
```

# 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 | The return code for the method. For more information, see “Return Code” on page 118. |
| inMetadata | C | in | A metadata property string that identifies the metadata object to be fetched. |
| outMetadata | C | out | The metadata property string returned from the server. |
| projectReposid | C | in | The ID of the target project repository. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed.<br>OMI_LOCK_TEMPLATE= 65536<br>Overrides the default lock template with a user-defined template submitted in the *Options* parameter.<br>OMI_RETURN_LIST=1024<br>Returns the object identifiers of any associated objects that were fetched in the *outMetadata* parameter. |
| options | C | in | An indicator for options. This method supports one option:<br><LockTemplates><br>Specifies a user-defined lock template that overrides the default lock template for the specified metadata object. 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. You can override the default lock template and fetch a different set of associated objects by setting the OMI_LOCK_TEMPLATE (65536) flag and supplying an alternate lock template in a <LockTemplates> element in the *Options* parameter. For more information, see “Using the Change Management Facility” in the *SAS Open Metadata Interface: 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.

Be sure to check the return code of a FetchMetadata 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 will be made.

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" on page 137
- "CheckinMetadata" on page 135
- "UndoCheckoutMetadata" on page 165

## GetMetadata

### Syntax

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

## Parameters

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | A metadata property string that identifies the metadata object whose properties are to be read. |
| outMetadata | C | out | The metadata property values returned from the metadata server. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed. |

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 "Querying Cross-Repository References" in "Creating Relationships Between Repositories" in the *SAS 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.

| Parameter | Type | Direction | Description |
|---|---|---|---|
| | | | OMI_TEMPLATE=4<br>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 indicator for options. This method supports the following options:<br><DOAS Credential="*credHandle*"/><br>Specifies an alternate calling identity for the request. For more information, see "<DOAS> Option" on page 131.<br><LockTemplates><br>Specifies associated objects to lock. The <LockTemplates> element must be specified in conjunction with the OMI_LOCK_TEMPLATE flag.<br><Templates><br>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 four 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 the properties requested in the *inMetadata* and *Flags* parameters. Templates also enable you to retrieve properties for associated objects.
- □ You can specify search criteria on the association name subelements that are passed to the server in either or both of the *inMetadata* parameter and in the <Templates> element to filter the associated objects that are retrieved.

For detailed usage information about the GetMetadata method, see "Querying Specific Metadata Objects" in the *SAS 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.

The OMI_LOCK (32768) flag is one of several multi-user concurrency controls supported by the SAS Open Metadata Interface. 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, you can additionally set the OMI_LOCK_TEMPLATE (65536) flag and supply one or more user-defined templates in a <LockTemplates> 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.) See “Metadata Locking Options” in the *SAS Open Metadata Interface: User’s Guide* for an overview of the concurrency controls supported by the SAS Open Metadata Interface.

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.

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” on page 121.

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” on page 201.

## 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" on page 152

## 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 | The return code for the method. For more information, see Return Code"Return Code" on page 118. |
| reposid | C | in | The ID of the target repository. |
| type | C | in | The metadata type name of the objects to retrieve. |
| objects | C | out | The returned list of metadata objects. |
| ns | C | in | The namespace to use as the context for the request. |

| Parameter | Type | Direction | Description |
|---|---|---|---|
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed.<br>OMI_DEPENDENCY_USED_BY=16384<br>Specifies to retrieve objects of the passed type from all repositories that have a DependencyUsedBy dependency in the repository chain.<br>OMI_DEPENDENCY_USES=8192<br>Specifies to retrieve objects of the passed type from all repositories that have a DependencyUses dependency in the repository chain.<br>OMI_NO_DEPENDENCY_CHAIN=16777216<br>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.<br>OMI_GET_METADATA=256<br>Specifies to execute a GetMetadata call for each object that is returned by the GetMetadataObjects request.<br>OMI_INCLUDE_SUBTYPES=16<br>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.<br>OMI_MATCH_CASE=512<br>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.<br>OMI_XMLSELECT=128<br>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 “Querying All Metadata of a Specified Type” in the *SAS Open Metadata Interface: User’s Guide* for more information. |
| options | C | in | An indicator for options. This method supports two options:<br><DOAS Credential="*credHandle*"/><br>Specifies an alternate calling identity for the request. For more information, see “<DOAS> Option” on page 131.<br><XMLSELECT><br>Specifies an XML search string. |

## Details

The GetMetadataObjects method retrieves a list of all objects of the specified type in the specified repository. The default behavior is to return “Identifying Metadata” on page 120 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 of the specified metadata type.
- OMI_GET_METADATA enables you to issue a GetMetadata request from within the GetMetadataObjects method 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 specify search criteria to filter the objects that are retrieved.

For additional information, including examples of using GetMetadataObjects flags, see “Querying All Metadata of a Specified Type” in the *SAS Open Metadata Interface: User’s Guide*.

General information about flags, including interdependencies between them, is provided in Using IOMI Flags“Using IOMI Flags” on page 121.

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” on page 205.

## 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" on page 148

## GetNamespaces

Lists namespaces
Category: Management Methods

### Syntax

rc = GetNamespaces(ns, flags, options);

### Parameters

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. For more information, see "Return Code" on page 118. |
| ns | C | out | The returned list of all namespaces |
| flags | L | in | This method does not support any flags. Reserved for future use.<br>For no flags, a 0 should be passed. |
| options | C | in | An indicator for options. This method does not support any options. |

### 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 provides 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" on page 209.

## 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" on page 162
- "GetSubtypes" on page 158

## 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 | The return code for the method. For more information, see “Return Code” on page 118. |
| repositories | C | out | The returned list of all repositories that are available to the current server. |
| flags | L | in | The flags supported for this method. This flag is optional. For no flags, a 0 should be passed.<br>OMI_ALL=1<br>Gets the Access and PauseState attributes of all repositories and the repository manager. |
| options | C | in | An indicator for options. This method does not support any options. |

## 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 alist of the repositories that are registered in the repository manager, as well as the description, default namespace, and “Identifying Metadata” on page 120 for each repository.

Setting OMI_ALL (1) 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 “RepositoryBase” on page 109.

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 program-specific examples, see “Program-Specific GetRepositories Examples” on page 213.

### Standard Interface

The following is an example of a GetRepositories method that is formatted for the standard interface and that is issued without flags.

```
flags=0;
options= "";
```

```
rc = GetRepositories(repositories,flags,options);
```

This is an example of the output returned by the metadata server:

```
<!-- 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 formatted for the *inMetadata* parameter of the DoRequest method and is issued with the OMI_ALL (1) flag set.

```
<!-- XML string for inMetadata= parameter of DoRequest method call -->
<GetRepositories>
   <Repositories/>
   <Flags>1</Flags>
   <Options/>
</GetRepositories>
```

This is an example of the output returned by the metadata server:

```
<!-- 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 | The return code for the method. For more information, see "Return Code" on page 118. |
| supertype | C | in | The name of the metadata type for which you want a list of subtypes. |
| subtypes | C | out | The 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 | The flags supported for this method. This flag is optional. For no flags, a 0 should be passed. OMI_ALL_DESCENDANTS=64 Returns all subtypes along with all of their descendants. |
| options | C | in | An indicator for options. This method supports one option: <DOAS Credential="*credHandle*"/> Specifies an alternate calling identity for the request. For more information, see "<DOAS> Option" on page 131. |

## 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 additional examples, see "Program-Specific GetSubtypes Examples" on page 216.

## 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);
```

The following is an example of the output returned by the metadata server:

```
<!-- 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” on page 162
- “IsSubtypeOf” on page 164

## 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 | The return code for the method. For more information, see “Return Code” on page 118. |
| type | C | in | The name of the metadata type for which you want a list of properties. |
| properties | C | out | The returned XML list of all properties for the specified metadata type. |
| ns | C | in | The namespace to use as the context for the request. |

| Parameter | Type | Direction | Description |
|---|---|---|---|
| flags | L | in | This method does not support any flags. Reserved for future use. For no flags, a 0 should be passed. |
| options | C | in | An indicator for options. This method supports one option: <DOAS Credential="*credHandle*"/> Specifies an alternate calling identity for the request. For more information, see "<DOAS> Option" on page 131. |

## 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 additional examples, see "Program-Specific GetTypeProperties Examples" on page 220.

## Example 1: Standard Interface

The following GetTypeProperties method call, formatted for the standard interface, requests the properties of the Column metadata type.

```
type="Column";
ns="SAS";
flags=0;
options="";

rc = GetTypeProperties(type, properties, ns, flags, options);
```

This is an example of the output returned by the metadata server:

```
<!-- 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” on page 162
- “GetSubtypes” on page 158

## 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 | The return code for the method. For more information, see “Return Code” on page 118. |
| types | C | out | The returned XML list of metadata types. |
| ns | C | in | The namespace to use as the context for the request. Use a " (double-quotation mark) to indicate the default namespace of the repository. |

| Parameter | Type | Direction | Description |
|---|---|---|---|
| flags | L | in | The flags that are supported for this method. This flag is optional. For no flags, a 0 should be passed.<br><br>OMI_SUCCINCT=2048<br>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 indicator for options. This method supports one option:<br><br><Reposid><br>Specifies a unique repository identifier, in the form<br><br>`A0000001.RepositoryId`<br><br>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 (2048) flag and its corresponding <Reposid> element are specified.

- Used without the flag, the method returns an XML string that lists all of the metadata types defined in the specified 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 additional examples, see “Program-Specific GetTypes Examples” on page 224.

For an example of a GetTypes request that sets the OMI_SUCCINCT (2048) flag, see “Listing the Types in a Repository” in “Overview” of querying metadata in the *SAS Open Metadata Interface: User’s Guide*.

## Example 1: Standard Interface

The following GetTypes request, formatted for the standard interface, 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 returned by the metadata 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” on page 155
- □ “GetSubtypes” on page 158

## 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 | The return code for the method. For more information, see “Return Code” on page 118. |
| type | C | in | The name of the metadata type that might be a subtype of *supertype*. |
| supertype | C | in | The name of the metadata type that might be a supertype of *type*. |
| result | N | out | The 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. |

| Parameter | Type | Direction | Description |
|---|---|---|---|
| flags | L | in | This method does not support any flags. Reserved for future use.<br>For no flags, a 0 should be passed. |
| options | B | in | An indicator for options. This method supports one option:<br>&lt;DOAS Credential="*credHandle*"/&gt;<br>Specifies an alternate calling identity for the request. For more information, see "&lt;DOAS&gt; Option" on page 131. |

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 additional examples, see "Program-Specific IsSubtypeOf Examples" on page 227.

## Example 1: Standard Interface

The following IsSubtypeOf request, formatted for the standard interface, 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" on page 158

## 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 | The return code for the method. For more information, see "Return Code" on page 118. |
| inMetadata | C | in | The metadata property string that identifies the metadata objects to be deleted and unlocked. |
| outMetadata | C | out | The property string returned from the metadata server. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for this method. All of the flags are optional. For no flags, a 0 should be passed.<br>OMI_LOCK_TEMPLATE= 65536<br>Overrides the default lock template with a user-defined template submitted in the *Options* parameter.<br>OMI_RETURN_LIST=1024<br>Includes the object identifiers of any associated objects that were unchecked in the method output. |
| options | C | in | An indicator for options. This method supports one option:<br>&lt;LockTemplates&gt;<br>Specifies a user-defined lock template that overrides the default lock template associated with the specified object. The &lt;LockTemplates&gt; 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, see "Using the Change Management Facility" in the *SAS 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.

Be sure to check the return code of an UndoCheckoutMetadata 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.

### Example 1: Standard Interface

The following UndoCheckoutMetadata request, formatted for the standard interface, deletes PhysicalTable object B700005N from project repository A3VTX83H and any associated objects specified in the default lock template. Any corresponding objects in the primary repository that were locked are unlocked as well. The OMI_RETURN_LIST (1024) flag specifies to list the associated objects that are 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" on page 137
- □ "CheckinMetadata" on page 135
- □ "FetchMetadata" on page 146

## 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 | The return code for the method. For more information, see “Return Code” on page 118. |
| inMetadata | C | in | A metadata property string that identifies the object to be updated. For the general format of this string, see “Constructing a Metadata Property String” on page 118. |
| outMetadata | C | out | The property string returned from the metadata server. |
| ns | C | in | The namespace to use as the context for the request. |
| flags | L | in | The flags supported for this method. The OMI_TRUSTED_CLIENT flag is required; all other flags are optional.<br><br>OMI_IGNORE_NOTFOUND = 134217728<br>Prevents an update operation from being aborted when a request specifies to update an object that does not exist.<br><br>OMI_RETURN_LIST = 1024<br>Returns the object identifiers of any dependent objects that were deleted as a result of the update operation in the *outMetadata* parameter.<br><br>OMI_TRUSTED_CLIENT = 268435456<br>Determines if the client can call this method. This flag is required.<br><br>OMI_UNLOCK=131072<br>Unlocks a lock that is held by the calling user.<br><br>OMI_UNLOCK_FORCE=262144<br>Unlocks a lock that is held by another user. |
| options | C | in | An indicator for options. This method supports one option:<br><br>&lt;DOAS Credential="*credHandle*"/&gt;<br>Specifies an alternate calling identity for the request. For more information, see “&lt;DOAS&gt; Option” on page 131. |

## 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 (134217728) 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 element in the input metadata property string 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 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. For an overview of the multiuser concurrency controls supported by the SAS Open Metadata Interface, see “Locking Metadata Objects” in the *SAS Open Metadata Interface: User’s Guide*. When OMI_UNLOCK (131072) or OMI_UNLOCK_FORCE (262144) is set, any associated objects that were locked are automatically unlocked as well.

Be sure to check the return code of an UpdateMetadata 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 UpdateMetadata method call without regard to the programming environment. Examples are given that use the standard interface and the DoRequest method. For additional examples, see “Program-Specific UpdateMetadata Examples” on page 231.

## Example 1: Standard Interface

The following is an example of an UpdateMetadata method call that is formatted for the standard interface. The specified attributes and values will replace those stored for the object of the specified metadata type 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);
```

The following is an example of the output returned by the metadata server:

```
<!-- 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 first quarter"/>
   </Metadata>
   <NS>SAS</NS>
   <!- OMI_TRUSTED_CLIENT flag ->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

## Related Methods

- □ "DeleteMetadata" on page 141
- □ "GetMetadata" on page 148

CHAPTER

8

# Security Methods (ISecurity Class)

## Overview of the ISecurity Class

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. For more information, see “Call Interfaces” on page 13.

The following information applies to all of the ISecurity methods:

- The parameter *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
  ```

- In the current release, 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 that is stored in SAS metadata repositories. Appropriate metadata must be defined in order for authorization decisions to be made. User metadata is defined by using the SAS Management Console User Manager plug-in or by extracting user and group definitions from an enterprise source using macros. Access control metadata is defined using the Authorization Manager plug-in to SAS Management Console. For information about the plug-ins, see the SAS Management Console documentation. For information about the access controls supported by the SAS Open Metadata Architecture authorization facility and the complete collection of tools to manage them, see the *SAS Intelligence Platform: Security Administration Guide*.

## 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 ISecurity methods.

## FreeCredentials

Frees the credential handle obtained by GetCredentials
Category: Authorization Methods

### Syntax

rc = FreeCredentials(credHandle);

### Parameters

| Parameter | Type | Direction | Description |
|---|---|---|---|
| credHandle | string | in | The 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 method call. △

```
FreeCredentials(credHandle_value);
```

### Related Methods

- “GetCredentials” on page 175

## 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 | A credential handle identifying a user identity or an empty string. |
| resource | string | in | A resource identifier. |

| Parameter | Type | Direction | Description |
|---|---|---|---|
| permission | string | in | A 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 | A 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" on page 177 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" on page 177

## 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 | The returned credential handle representing the subject. |
| subject | string | in | The 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” on page 172
- “GetIdentity” on page 176

# GetIdentity

Gets identity metadata for the specified user
Category: Authorization Methods

## Syntax

rc = GetIdentity(credHandle,identity);

## Parameters

| Parameter | Type | Direction | Description |
|---|---|---|---|
| credHandle | string | in | The credential handle identifying a user identity, an empty string, or a user ID in the form "LOGINID:*userid*”. |
| identity | string | out | The 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" on page 175

# 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 | The credential handle identifying a user identity or an empty string. |
| resource | string | in | A resource identifier. |
| permission | string | in | The name of a metadata permission(s). |
| permissionCondition | string | out | The 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 the current release, PermissionCondition objects are supported only 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” on page 173

CHAPTER
9

# Repository and Server Control Methods (IServer Class)

## Overview of the IServer Class

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.

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 parameter rc captures the return code of the method.

*Note:* A user must have *unrestricted user* or *administrative user* status on the metadata server in order to pause, refresh, and resume repositories and the repository manager, and to stop the metadata server. Anyone can issue the Status method. For more information about the administrative privileges supported on the metadata server, see the *SAS Intelligence Platform: Security Administration Guide*. △

## Using the IServer Class

The IServer class contains methods that enable you to perform the following tasks:

□ pause and resume the SAS Metadata Server. Client activity on the SAS Metadata Server must be paused prior to updating or deleting a repository object or updating the repository manager. After repository and repository manager updates are completed, client activity must be resumed by using a Resume method.

□ refresh the SAS Metadata Server to recover memory or to change certain server configuration options.

□ poll the SAS metadata server to determine its status.

□ shut down (stop) the SAS Metadata Server.

## Pause

Temporarily limits the availability of a repository, the repository manager, or all repositories

Category: Repository Control

### Syntax

rc=Pause(options);

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1). |
| options | C | in | An indicator for options. This method supports one option:<br><REPOSITORY><br>specifies an XML property string that identifies 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" on page 109. To determine the access mode in force for a given repository, issue a GetRepositories method, setting the OMI_ALL (1) flag. The GetRepositories method will return a list of all repositories registered on the metadata 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 executed without options, the Pause method quiesces client activity in all repositories on the metadata 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" on page 186 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 those defined on a repository's Default ACT.

The Pause method is supported only in the standard interface. For more information, see "Call Interfaces" on page 13.

A user must have *unrestricted user* or *administrative user* status on the metadata server in order to pause the metadata server.

### Example

The following example pauses a repository for READONLY.

```
<!--Pause repository A5H9YT45 for READONLY-->
options='<Repository Id="A5H9YT45" State="READONLY"/>';

rc=Pause(options);
```

## Refresh

Changes SAS Metadata 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);

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1). |
| options | C | in | An indicator for options. This method supports two options:<br>&lt;ARM&gt;<br>an XML element that specifies one or more ARM system options.<br>&lt;REPOSITORY&gt;<br>an XML element that identifies a repository to pause and resume. |

### Details

Refresh is issued on a running metadata 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 the *SAS Intelligence Platform: System Administration Guide* for more information about SAS OMA ARM logging, including 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" on page 182 method.

When used with the <Repository> element, the Refresh method clears a repository's PauseState attribute.

Executing 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. For more information, see "Call Interfaces" on page 13.

A user must have *unrestricted user* or *administrative user* status on the metadata server in order to refresh the metadata server.

## 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);
```

## Resume

Restores client activity in one or more repositories
*Category:* Repository Control

### Syntax

rc=Resume(options);

| Parameter | Type | Direction | Description |
|---|---|---|---|
| rc | N | out | The return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1). |
| options | C | in | An indicator for options. This method supports one option:<br><REPOSITORY><br>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. When executed 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 "RepositoryBase" on page 109 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. For more information, see "Call Interfaces" on page 13.

A user must have *unrestricted user* or *administrative user* status on the metadata server in order to resume the metadata server.

## 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);
```

## Status

Polls the SAS Metadata Server and returns SAS Metadata Server and SAS Metadata Model version information
Category: Server Control

### Syntax

rc=Status(inmeta, outmeta, options);

| Parameter | Type | Direction | Description | |
|---|---|---|---|---|
| rc | N | out | The return code for the method. Indicates whether the server ran the method call (rc=0) or not (rc=1). | |
| inmeta | string | in | An XML string that requests additional information to be returned from the metadata server. | |
| | | | <ModelVersion/> | requests the SAS Metadata Model version number. |
| | | | <PlatformVersion/> | requests the SAS Metadata Server version number. |
| | | | <State/> | requests the server's availability. |
| | | | <Version/> | deprecated in SAS 9.1.3, Service Pack 3. |
| outmeta | string | out | An output string that mirrors the content of the inmeta parameter except that it has return values filled in. | |
| options | C | in | An indicator for options. No options are supported at this time. | |

## Details

The *inmeta* parameter is a string that contains one or more XML elements that request information.

The *outmeta* parameter mirrors the *inmeta* parameter and returns the following values:

<ModelVersion/>
: returns the SAS Metadata Model version number in the form X.XX, for example, 5.01. The model version is incremented when there is a change to the metadata model. A change includes the addition, modification, or removal of metadata types, attributes, and associations. The integer part of the version number is the "repository format" number. When this number is incremented, it is an indication that the underlying data structure has changed and a conversion of the tables should be performed. It is possible that a server written for format 5 to use repositories that are format 4 without conversion; however, there will likely be a performance penalty and some features will not be available. The decimal part of the version number indicates that a model change was made, but there is no need for conversion. This would be the case if a new metadata type or association was added to the model.

<PlatformVersion/>
: returns the SAS version number of the metadata server in the form X.X.X.X. For example, for a metadata server that is running SAS 9.1.3, Service Pack 3, the platform version number is 9.1.3.3. When the server installation is updated to use SAS 9.2, the platform version number will be 9.2.0.0.

<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/>
: the use of this element is deprecated beginning with SAS 9.1.3, Service Pack 3. In previous releases of SAS software, it returned the SAS Metadata Model version number.

The Status method does not check the availability of SAS Metadata Repositories. To determine the availability of the repositories registered on the metadata server, you must use the GetRepositories method. See "GetRepositories" on page 156 for more information.

### Standard Interface Example

```
<!--Determine availability and version numbers of the
SAS Metadata Model and metadata server-->
inmeta='<State/> <ModelVersion/> <PlatformVersion/>'
outmeta=''
options=''

rc=Status(inmeta,outmeta,options);
```

### DoRequest Example

```
<!--Determine availability and version numbers of the
SAS Metadata Model and metadata server-->
```

```
<Status>
<Metadata>
 <State/>
 <ModelVersion/>
 <PlatformVersion/>
</Metadata>
<Options/>
</Status>
```

## Stop

Shuts down the SAS Metadata Server
Category: Server Control

### Syntax

serverObject.Stop(options);

| Parameter | Type | Direction | Description |
|---|---|---|---|
| options | C | in | An indicator for options. 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.

A user must have *unrestricted user* or *administrative user* status on the metadata server in order to stop the server. For more information about these privileges, see the *SAS Intelligence Platform: Security Administration Guide*.

### Example

```
<!--Stops the metadata server-->
options=''

rc=Stop(options);
```

CHAPTER

# 10 Program-Specific Method Examples

## Overview to Program-Specific Method Examples

This section contains Java, Visual Basic, and Visual C++ examples of the method calls described in "Overview of the IOMI Class Methods" on page 117. The IOMI samples are code fragments that can be used in conjunction with the sample IOMI connection code presented in "Sample Java IOMI Client" on page 24, "Sample Visual Basic OMI Client" on page 28, and "Sample Visual C++ IOMI Client" on page 31 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.

## Program-Specific AddMetadata Examples

The following code fragments create a PhysicalTable object by using first the standard interface and then the DoRequest method.

### 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;);
}
```

## Program-Specific DeleteMetadata Examples

The following code fragments delete a SASLibrary object and a PhysicalTable object by using first the standard interface and then the DoRequest method.

### 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;);
}
```

## 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, by using first the standard interface and then using the DoRequest method.

### 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;);
}
```

## Program-Specific GetMetadataObjects Examples

The following code fragments retrieve all objects of type PhysicalTable by using first the standard interface and then the DoRequest method.

## 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;);
}
```

## Program-Specific GetNamespaces Examples

The following code fragments issue a GetNamespaces call by using first the standard interface and then the DoRequest method.

## 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;);
   }
```

## Program-Specific GetRepositories Examples

The following code fragments issue a GetRepositories method call by using first the standard interface and then the DoRequest method.

### 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;);
}
```

## Program-Specific GetSubtypes Examples

The following code fragments retrieve the Subtypes for supertype DataTable by using first the standard interface and then the DoRequest method.

## 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;);
}
```

## Program-Specific GetTypeProperties Examples

The following code fragments issue a GetTypeProperties call for metadata type Column by using first the standard interface and then the DoRequest method.

### 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;);
    }
```

## 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

### 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;);
}
```

## Program-Specific IsSubtypeOf Examples

The following code fragments issue an IsSubtypeOf method call by 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

### 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;);
}
```

## Program-Specific UpdateMetadata Examples

The following code fragments issue an UpdateMetadata method call by 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

### 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;);
}
```

# PART 4

# SAS Language Metadata Interfaces

CHAPTER
11

# Procedures

## METADATA Procedure

The METADATA procedure sends an XML string 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 submit the method call, and to optionally store the output XML to a file. You must know how to format a SAS Open Metadata Interface method call in order to use this procedure. The method calls must be formatted for the *inMetadata* parameter of the DoRequest method. For information about the DoRequest format, see "DoRequest" on page 144.

### Procedure Syntax

Use the following syntax to execute the METADATA Procedure. Note that unlike most SAS procedures, which end individual statements with a semicolon (;), the PROC METADATA statements are all submitted as one string. That is, only one semicolon is used, and this semicolon is placed after the last statement that is submitted before the RUN statement. See "Examples" on page 243 for specific usage.

**PROC METADATA**

Server Connection Statements

<**SERVER=** "*host-name*">
<**PORT=** *port–number*>
<**USERID=** "*userid*">
<**PASSWORD=** "*password*">
<**PROTOCOL=** BRIDGE>

Input Statement

**IN=** *fileref* | "*XML–formatted–method–call*"<;>

Output Statements

<**OUT=** *fileref*><;>
<**HEADER=** NONE | SIMPLE | FULL><;>

Informational Statements

<**REPOSITORY=** "*repository-identifier*"><;>
<**VERBOSE**><;>

Execution Statement

**RUN;**

## Server Connection Statements

The server connection statements are optional in the METADATA procedure but are required to connect to a 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" on page 305. 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.

*Note:* 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=
: specifies 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 metadata server on the same computer.

PORT=
: specifies the TCP port to which the SAS Metadata Server listens for connections. For example, port=8561. It should be the same port number that was used to start the SAS Metadata Server.

USERID=
: specifies an authenticated user ID. See the *SAS Intelligence Platform: Security Administration Guide* for information about authentication requirements.

PASSWORD=
: specifies the password corresponding to the authenticated user ID.

PROTOCOL=
: specifies 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.

For more information, see "Example of Specifying Server Connection Statements" on page 243.

## Input Statement

IN=
: specifies an XML-formatted SAS Open Metadata Interface method call or a fileref that points to an XML file that contains a SAS Open Metadata Interface method call.

PROC METADATA uses the SAS Open Metadata Interface DoRequest method to submit method calls to the metadata server. The DoRequest method provides a generic way of submitting metadata-related method calls to the server. For information about how to format a method request for the DoRequest method, see "DoRequest" on page 144. To view a list of metadata-related methods, see Chapter 7, "Methods for Reading and Writing Metadata (IOMI Class)," on page 115.

You also need to know how to construct a metadata property string that defines or queries metadata. For information to help you write a metadata property string, see "Constructing a Metadata Property String" on page 118. For a listing of the metadata types defined in the SAS Metadata Model, see the "Alphabetical Listing of SAS Namespace Metadata Types." This listing is available only in online versions of the *SAS Open Metadata Interface: Reference*. Look for it in *SAS Help and Documentation* or on *SAS OnlineDoc*.

For an example of specifying the IN= statement, see "Example of a Simple Method Call" on page 243 and "Example of Filerefs in the IN= and OUT= Statements" on page 244.

## Output Statements

OUT=
: optionally specifies a fileref in which to store the output returned by the SAS Metadata Server. In most cases, the server returns an output XML string that mirrors the input string, except that the requested values are filled in. If the OUT= statement is omitted, procedure output is written to the SAS Log.

HEADER=
: specifies an encoding declaration for the output XML. 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 that was used to identify the output XML file to the SAS system or from the ENCODING= system option.

For an example of specifying output statements, see "Example of the HEADER= Statement" on page 244.

## 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 repository named in 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 pre-processing.

For an example of specifying informational statements, see "Example of the VERBOSE Statement" on page 245.

## Execution Statement

RUN;
: executes the METADATA procedure. RUN is a required statement.

## Examples

### Example of a Simple Method Call

The following PROC METADATA example issues a GetTypes method call.

```
PROC METADATA

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

This example omits server connection statements, therefore, the software will use the system option values of METASERVER, METAPORT, METAUSER, METAPASS, METAREPOSITORY, 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.

### Example of Specifying Server Connection Statements

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/>
        </GetRepositories>";
   RUN;
```

## Example of Filerefs in the IN= and OUT= Statements

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.

## Example of the HEADER= Statement

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 National Language Support (NLS): User's Guide*.

## Example of the VERBOSE Statement

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>
```

# METALIB Procedure

The METALIB procedure synchronizes table definitions in a SAS metadata repository with current information from the physical library data source.

- For any tables in the physical library that have no metadata in the metadata repository, the procedure will add table metadata.
- For any tables in the physical library that do have metadata in the repository, the procedure will update the table definitions to match the corresponding physical

tables. The procedure will update information about a table's columns, indexes, unique keys, foreign keys, and key associations.

- For any table definitions that exist in the metadata repository but do not have a corresponding table in the physical library, the procedure will optionally delete the metadata.

By default, the procedure adds and updates metadata for all table definitions that are associated with the specified SASLibrary, including those that exist in dependent repositories. Optional statements enable you to modify the procedure's behavior as follows:

- to select or exclude specific tables from processing
- to suppress the metadata add action, the metadata update action, or both
- to delete obsolete and duplicate table definitions from the SAS metadata repository
- to generate a report of needed metadata changes without actually applying the changes

## Metadata Types Updated by PROC METALIB

The procedure adds, updates, and deletes metadata of the following metadata types: PhysicalTable, Column, Index, UniqueKey, ForeignKey, and KeyAssociation. For more information about these metadata types, see their descriptions in the "Alphabetical Listing of SAS Namespace Metadata Types" in online versions of this guide.

## Understanding Support for Dependent Repositories

A SASLibrary metadata object represents a SAS library statement. This library statement can be stored in one repository, and metadata objects describing the tables referenced by the library can be stored in the same or a different repository, as long as the repositories have a dependency relationship defined between them. A dependency relationship enables cross-references to be defined between the objects in repositories.

In a dependency relationship, one repository functions like a parent and the other like a child. In the SAS Business Intelligence environment, the foundation repository is always a parent to a custom repository. A custom repository can also be a parent to another custom or a project repository.

A SASLibrary object can exist in either a parent or a child repository. PROC METALIB will add new table definitions to the repository that contains the specified SASLibrary object, regardless of whether it is a parent or a child. When updating and deleting table definitions, however, the default behavior of the procedure is to evaluate associated table definitions that exist in the specified repository and all of its children repositories. When a table definition in a dependent repository is updated, any new column, index, or key definitions are added to the dependent repository.

## How PROC METALIB Works

By default, PROC METALIB uses the SAS names of the tables found in the physical location referenced by the SASLibrary metadata object to determine what metadata changes need to be made. A table definition in a SAS Metadata Repository stores the SAS name of the table that it describes in the SASTableName attribute. The table definitions that are associated with a given SASLibrary object are tracked in the repository's association list.

For each SAS name found in the referenced physical location, the procedure checks the association list to see if a matching table definition exists.

- □ If a matching table definition does not exist, one is created.
- □ When a matching table definition is found, the definition is updated.
- □ If more than one matching table definition exists in the association list, only the first definition is updated. When more than one table definition stores the same SAS name in the SASTableName attribute, the additional table definitions are considered to be duplicates.
- □ If a table definition exists that does not correspond to one of the SAS names, it is ignored, unless the UPDATE_RULE=(DELETE) statement is set in the procedure. When UPDATE_RULE=(DELETE) is set, table definitions that do not correspond to the SAS name of a physical table are deleted, including duplicate definitions.

When the SELECT statement is set, the procedure can behave in either of two ways. The SELECT statement enables you to select tables or table definitions for processing. Tables are identified by specifying their SAS name. Table definitions are identified by specifying the value stored in their SASTableName attribute or by specifying the table definition's unique 17–character metadata identifier.

- □ When you specify a table name or SASTableName value in the SELECT statement, the procedure functions similarly to the way it does in default mode:
  - □ It checks the SASLibrary object's association list to see if a matching table definition exists.
  - □ If a table definition is not found, one is created in the repository that contains the SASLibrary object.
  - □ When a matching table definition is found, it is updated. If duplicate definitions exist, the duplicates are ignored.
  - □ If a matching table definition exists, but a physical table with that SAS name does not exist, then the table definition is ignored, unless UPDATE_RULE=(DELETE) is set. When UPDATE_RULE=(DELETE) is set, all table definitions that contain the specified name in their SASTableName attribute, including duplicates, are deleted.
- □ When you specify a table definition's metadata identifier in the SELECT statement, only the specified table definition is processed, as follows:
  - □ The specified table definition is updated, rather than the first table definition found in the association list.
  - □ When UPDATE_RULE=(DELETE) is set, duplicate table definitions are not deleted.

For syntax information, see "Table Selection Statements" on page 251.

## Procedure Syntax

**PROC METALIB;**

Server Connection and Library Identification Statement

**OMR** <=> (**LIBID** = "*library-identifier*"
| **LIBRARY** = "*library-name*"
| **LIBURI** = "*URI-format*"

<**USER** = "*authorized-userid*">
<**PASSWORD** = "*associated-password*">

<**METASERVER** = "*hostname*">

<**REPID** = "*repository-identifier*" | **REPNAME** = "*repository-name*">
<**PROTOCOL**= BRIDGE>
<**PORT**="*port-number*">);

Statement for Specifying Metadata Changes

<**UPDATE_RULE** <=> (<NOADD> <DELETE> <NODELDUP>
<NOUPDATE>);>

Table Selection Statements

<**EXCLUDE** <=> (<">*table-name-1*<"> <<">*table-name-n*<">>);>
<**SELECT** <=> (<">*table-name-1*<"> <<">*table-name-n*<">>
| *table-identifier-1* <*table-identifier-n*>);>

Reporting Statement

<**REPORT**;>

Statements That Control Change Execution

<**NOEXEC;**>
<**QUIT | EXIT;**>
**RUN;**

## Server Connection and Library Identification Statement

The OMR statement supports metadata server connection parameters and identifies the physical library that will be examined. The server connection parameters are optional. When the server connection parameters are omitted from the OMR statement, the values of the METASERVER, METAPORT, METAUSER, METAPASS, and METAREPOSITORY system options are used to connect to the metadata server. For details about these system options, see the *SAS Language Reference: Dictionary*.

*Tip:* To determine what system option settings are active in your SAS session, submit the following procedure statement:

```
proc options group=meta; run;
```

To override the settings with new ones, submit an OPTIONS statement.

If these system options are not set or if they provide incomplete information, then a dialog box is posted to acquire the necessary server connection values.

The library identification parameter (LIBID=, LIBRARY=, or LIBURI=) is required. Refer to the following descriptions for more information about the required and optional OMR statement parameters.

LIBID= | LIBRARY= | LIBURI=
LIBID=, LIBRARY=, or LIBURI= identify the physical library that will be examined by referencing the SASLibrary metadata object that represents the physical library in a SAS Metadata Repository. Use only one of these parameters in the OMR statement.

LIBID=
: specifies the 8-character metadata identifier of the SASLibrary object that represents the physical library that you want to examine.

LIBRARY=
: specifies the value stored in a SASLibrary metadata object's Name attribute.

LIBURI=
: specifies a SASLibrary metadata object by using one of the SAS Open Metadata Architecture Uniform Resource Identifier (URI) formats. The URI formats are described in Table 11.1 on page 249.

**Table 11.1** URI Formats

| Format type | Description |
|---|---|
| id | specifies the 8-character metadata identifier of the SASLibrary object that you want to access. This format is similar to specifying LIBID=. An example of specifying LIBURI with an id value is LIBURI="id=A9000001". Use of single- or double-quotation marks to enclose the identifier is supported, but not required. |
| type/id | is the metadata type name and metadata object identifier of the SASLibrary object that you want to access, for example, LIBURI="SASLibrary/A9000001". Because the metadata type required by PROC METALIB is always SASLibrary, this format is basically the same as specifying LIBID=. Use of single or double quotation marks to enclose the identifier is optional. |
| type-@search-criteria | is the object's metadata type name followed by a search string, which is in the form of an attribute=value pair, for example, LIBURI="SASLibrary-@name='oralib'" or LIBURI="SASLibrary-@engine='base'". The -@ ( dash and "at" sign) characters are required. This URI format must be enclosed in double-quotation marks, and the attribute value must be enclosed in single-quotation marks. |

LIBID=, LIBRARY=, or LIBURI= is typically specified along with the REPID= or REPNAME= argument to indicate the active repository. The active repository is the one that contains the specified SASLibrary object. If you omit a repository identification parameter from the OMR statement, then PROC METALIB will search for the specified library in the repository identified in the METAREPOSITORY system option.

For information about how to obtain these SASLibrary and repository identifiers, see "Obtaining Metadata Identifiers" on page 253.

USER=
: is an optional parameter that specifies an authorized user ID on the metadata server. An authorized user ID is one that has been authenticated by the metadata server and has ReadMetadata and WriteMetadata permission to the specified SASLibrary. METAUSER= is an alias for USER=.

PASSWORD=
: is an optional parameter that specifies the password stored for the authorized user ID on the metadata server. PW= and METAPASS= are aliases for PASSWORD=.

METASERVER=
: is an optional parameter that specifies the IP address or host name of the computer that is hosting the metadata server. IPADDR= and SERVER= are aliases for METASERVER=.

REPID= | REPNAME=
: are optional parameters that identify the repository that contains the specified SASLibrary metadata object.

REPID= supports an 8–character repository identifier as a value.

REPNAME= specifies the name stored in the repository's Name attribute. METAREPOSITORY= is an alias for REPNAME=.

Use either REPID= or REPNAME=. If you specify both, REPID= takes precedence over REPNAME=.

For information about how to obtain these repository identifiers, see "Obtaining Metadata Identifiers" on page 253.

PORT=XXXX
: is an optional parameter that specifies the TCP port to which the SAS Metadata Server listens for requests and that clients use to connect to the server. XXXX is a unique number from 0-64K. METAPORT= is an alias for PORT=.

PROTOCOL=
: is an optional parameter that specifies the network protocol for connecting to the metadata server. The valid value is BRIDGE. METAPROTOCOL= is an alias for PROTOCOL=.

*Note:* Both the library identification parameter and the optional server connection parameters must be submitted to the OMR statement within parentheses. △

*Note:* The equal sign (=) between the OMR keyword and the parenthesized list is optional. The quotation marks around parameter values are also optional, except for PORT= and PROTOCOL=. The PORT= value must be quoted. The PROTOCOL= value should not be quoted. △

## Statement for Specifying Metadata Changes

UPDATE_RULE is an optional statement that enables you to override one or both of the METALIB procedure's default add and update actions and to specify additional or optional actions that you want to perform. Valid arguments are:

NOADD
: specifies not to add table metadata to the metadata repository for physical tables that have no metadata.

NOUPDATE
: specifies not to update existing table metadata to resolve discrepancies with the corresponding physical tables.

DELETE
: specifies to delete table metadata if a corresponding physical table is not found in the specified library.

NODELDUP
: specifies not to delete duplicate table definitions in the SAS metadata repository. A duplicate table definition is one that has the same SASTableName value as the table being processed. Duplicate table definitions are deleted by default when DELETE is specified. NODELDUP is valid only when DELETE is also specified.

For more information about add, update, and delete processing, see "Table Addition Report Description" on page 257, "Table Update Report Description" on page 257, and "Table Deletion Report Description" on page 259.

*Note:* Arguments must be submitted within parentheses. The equal sign (=) between the UPDATE_RULE keyword and the parenthesized list of arguments is optional. △

*Note:* An error is returned if you specify both NOADD and NOUPDATE and omit DELETE. The procedure must be given an action to perform if both of the default actions are suppressed. △

## Table Selection Statements

The default behavior of the METALIB procedure is to update all of the table definitions associated with a SASLibrary metadata object in SAS Metadata Repositories with information from all of the tables found in the physical location referenced by the SASLibrary metadata object.

SELECT <=> (<">*table-name-1*<"> <<">*table-name-n*<">>
| *table-identifier-1* <*table-identifier-n*>) ;
: is an optional statement that specifies a specific table or table definition to process. You can specify multiple tables or table definitions. However, do not specify both a table name and a metadata identifier in the SELECT statement. The SELECT statement will fail if you submit both a table name and a metadata identifier.

EXCLUDE <=> (<">*table-name-1*<"> <<">*table-name-n*<">>) ;
: is an optional statement that specifies a single table or a list of tables to exclude from processing.

*Table-name* is the SAS name of a table in the physical location referenced by the specified SASLibrary metadata object. It can also be the value stored in the SASTableName attribute of a table definition in a SAS Metadata Repository. A table definition in a SAS Metadata Repository stores the SAS name of a physical table in the SASTableName attribute.

*Note:* Do not confuse the table definition's Name= attribute with the SASTableName= attribute. The Name= attribute stores a user-defined name that was assigned to the metadata object when it was created. This is sometimes the SAS name of the physical table that it describes but it does not have to be. △

*Table-identifier* is a table definition's 17–character metadata identifier, in the form **`Reposid.Objectid`**.

For information about how PROC METALIB processes table names and metadata identifiers, see "How PROC METALIB Works" on page 246. For instructions to obtain a table definition's metadata identifier, see "Obtaining Metadata Identifiers" on page 253.

*Note:* The SELECT and EXCLUDE statements are mutually exclusive; only one statement can be used in a single execution of the METALIB procedure. △

*Note:* Table names and identifiers must be submitted within parentheses, as shown. Use of quotation marks around table names is optional, unless the table name contains special or mixed-case characters. If table names that contain special or mixed-case characters are not quoted, the procedure will convert the names to uppercase letters. For example, if you submit the following statement:

```
SELECT (tab1 tab2 tab3 "Table4");
```

tab1, tab2, and tab3 will be uppercased because they are not quoted.

Do not quote a table definition's metadata identifier. The procedure expects a two-part identifier. If you use quotation marks, the identifier must be quoted as follows: "A0001234"."A9375BC45". △

*Note:* When selecting and excluding tables, be aware that the tables you select can affect the associated objects that are updated. For example, both the primary key and foreign key tables must be selected for foreign key metadata to be updated. The primary key and foreign key tables must also be in the same repository. △

## Reporting Statement

REPORT;

is an optional statement that creates a SAS output listing that summarizes metadata changes. The default report destination is a SAS output listing. The SAS Output Delivery System (ODS) can be used to print the output listing in different report formats, such as HTML and RTF.

When ODS is used, a SAS output listing will be produced in addition to the specified ODS output, unless you specifically suppress the output listing using ODS.

For information about the content of the REPORT output listing, see "Understanding REPORT Statement Output" on page 256.

## Statements That Control Change Execution

NOEXEC;
: is an optional statement that prevents the indicated metadata changes from being applied.

  NOEXEC is typically used with the REPORT statement to enable you to identify any needed metadata changes before they are made. Running PROC METALIB with the NOEXEC and REPORT statements prior to running PROC METALIB in default mode can help prevent unwanted metadata changes from being made.

QUIT | EXIT;
: are optional statements that enable you to terminate execution of the METALIB procedure when the procedure statements are submitted in line mode.These statements terminate the procedure immediately and abort any processing that might have occurred.

RUN;
: executes the METALIB procedure.

## Obtaining Metadata Identifiers

The SAS Metadata Server uses a 17-character identifier to identify any given metadata resource (metadata object) on the metadata server. The 17–character identifier, which looks like A32V87R9.A9000001, is composed of three parts:

- The first eight characters (A32V87R9, in the example) identify the repository in which the metadata resource is stored
- The ninth character is always a period
- The second set of eight characters (A9000001, in the example) uniquely identifies the resource in the named repository.

The METALIB procedure requires you to specify identifiers for metadata objects of the following types:

Repository
: identifies a particular metadata repository on the SAS Metadata Server.

SASLibrary
: describes a physical library that is accessed by a SAS LIBNAME engine

You can also optionally specify the 17–character identifier of table definition in the SELECT statement.

When supplying input to the METALIB procedure, use the appropriate portion of the resource's 17–character identifier to identify the item. For example, if a SASLibrary object has the 17–character metadata object identifier 'A32V87R9.A9000001', use the first eight characters ('A32V87R9') in the REPID= argument to identify the repository and use the second set of eight characters (A9000001) in the LIBID= argument to identify the SASLibrary.

A table definition is identified by its full 17-character metadata identifier.

You can determine the metadata identifiers of library and table definitions that exist in the same repository by using either the METABROWSE command, or by using the SAS Management Console Data Library Manager. For instructions to use these tools, see "Obtaining Metadata Identifiers Using the METABROWSE Command" on page 254 and "Obtaining Metadata Identifiers in SAS Management Console" on page 255.

For information about how to obtain the metadata identifiers of table definitions that exist in dependent repositories, see "Listing Table Definitions in Dependent Repositories" on page 255.

## Obtaining Metadata Identifiers Using the METABROWSE Command

The METABROWSE command is a SAS language interface that enables authorized users to browse the details of metadata on the SAS Metadata Server.

To use the METABROWSE command to obtain a SASLibrary metadata object identifier:

1 Issue the METABROWSE command at a SAS command prompt.
2 When prompted, enter metadata server connection parameters.
3 In the Metadata Browser window, select and expand the node representing the repository that contains the library definition.
4 Scroll the list of metadata types until you find "SASLibrary" and expand the SASLibrary node. The available library definitions are listed alphabetically under the node.
5 Highlight the definition that represents the library that you want to examine. The library's properties will display in the righthand pane of the Metadata Browser window.
6 The library's 17–character metadata identifier is in the ID field of the properties pane.

## Obtaining Metadata Identifiers in SAS Management Console

SAS Management Console is a java application that provides a single point of control for managing resources that are used throughout the Intelligence Value Chain. It also enables you to browse the metadata on a SAS Metadata Server. To obtain a SASLibrary metadata object's identifier using SAS Management Console:

1 Open SAS Management Console: **Start ▶ Programs ▶ SAS ▶ SAS Management Console**.
2 When prompted, enter metadata server connection parameters.
3 Expand the Data Library Manager in the navigation tree.
4 Expand the SAS Libraries node in the navigation tree.
5 Right-click a library name with your mouse, and select Properties from the pop-up menu.
6 The 17–character identifier is displayed on the General tab of the Properties window in an ID field under the Name field.

## Listing Table Definitions in Dependent Repositories

To view table definitions that exist in dependent repositories, issue the following XML method call in PROC METADATA:

```
<GetMetadata>
   <Metadata>
      <SASLibrary Id="A5KI3PIS.AZ0000GP"
        <Tables/>
      </SASLibrary>
   </Metadata>
   <NS>SAS</NS>
   <! -- OMI_DEPENDENCY_USED_BY flag -- >
   <Flags>16384</Flags>
   <Options/>
</GetMetadata>
```

The GetMetadata method enables you to get all or specified properties of a specific metadata object. This request specifies to retrieve all metadata objects that have a Tables association to the SASLibrary object identified in the Id= attribute in the repository that contains the SASLibrary object. Setting the OMI_DEPENDENCY_USED_BY flag (16384) specifies to include dependent repositories in the search.

*Note:* Be sure to replace the Id= value shown above with a real 17–character metadata identifier. △

The SAS Metadata Server will return output similar to the following:

```
<GetMetadata>
<Metadata>
<SASLibrary Id="A5KI3PIS.AZ0000GP">
<
```

```
Tables>
<PhysicalTable Id="A5KI3PIS.B00000M9" Name="SameReposAsLib" Desc=""/>
<PhysicalTable Id="A5SGOWT6.AA000001" Name="InDepRepos" Desc=""/>
</Tables>
</SASLibrary></Metadata>
<NS>SAS</NS>
<Flags>16384</Flags>
<Options/>
</GetMetadata>
```

In this example, SASLibrary A5KI3PIS.AZ0000GP has two table definitions associated with it: one exists in the same repository as the library definition (indicated by the A5KI3PIS in the first part of the table definition's two-part metadata identifier). The second table definition, which is preceded by a different identifier (A5SGOWT6), is in a dependent repository. (Note that the Name= attribute stores the metadata name of each table definition, not the value in the SASTableName attribute.)

Sometimes more than one table definition exists that has the same name. To identify the repositories to which the definitions belong, issue the following method call:

```
<GetRepositories>
<Repositories>
<Flags>0</Flags>
</GetRepositories>
```

The GetRepositories method lists the Id=, Name=, and Desc= attributes of all repositories registered on a SAS Metadata Server, as follows:

```
<GetRepositories>
<Repositories>
<Repository Id="A0000001.A5KI3PIS" Name="Foundation"
    Desc="Main repository" DefaultNS="SAS"/>
<Repository Id="A0000001.A5SGOWT6" Name="Test"
    Desc="Test repository" DefaultNS="SAS"/>
<Flags>0</Flags>
</GetRepositories>
```

For information about how to issue the XML calls using PROC METADATA, see "METADATA Procedure" on page 240.

## Understanding REPORT Statement Output

The default behavior of the METALIB procedure is to print the following summary information in the SAS log:

- the total number of tables that were analyzed
- the number of tables that would be or were updated
- the number of tables that would be or were added
- if UPDATE_RULE=(DELETE) is set, the number of table definitions that would be or were deleted

The REPORT statement is supported to additionally create an output listing that summarizes the metadata additions, deletions, and updates that would be or were made. When REPORT is specified with NOEXEC, the output listing summarizes needed metadata changes, but the procedure does not apply any of the changes to the SAS Metadata Repository. When REPORT is specified alone, the listing summarizes changes that were actually made to the SAS Metadata Repository.

The following sections describe the output listings that are created when the METALIB procedure adds, updates, and deletes table and associated metadata.

## Table Addition Report Description

The following is an example of the output listing that is created when PROC METALIB is executed with the REPORT statement and one or more table definitions are added to the SAS Metadata Repository.

```
                                The METALIB Procedure

                           Summary Report for Library sas91 lib2
                                Repository Meta Proc repos
                                         17MAR2005

                                 Metadata Summary Statistics

                              Total tables analyzed           2
                              Tables Updated                  0
                              Tables Added                    2
                              Tables matching data source     0
                              Tables not found                0


------------------------------------------------------------------------------------
                                       Tables Added
------------------------------------------------------------------------------------


Metadata Name                     Metadata ID          SAS Name

COUNTRY                           A5HJ58JU.AX001LPV    COUNTRY
POSTAL                            A5HJ58JU.AX001LPW    POSTAL
```

In the output listing:

Metadata Name
: is the metadata name of the table definition that was added. This value is stored in the table definition's Name= attribute.

Metadata ID
: is the table definition's 17-character unique object identifier.

SAS Name
: is the SAS table name of the physical table. This value stored in the table definition's SASTableName= attribute.

In this example, two table definitions were created, one for a SAS table named COUNTRY and another for a SAS table named POSTAL. Metadata about each table's columns, indexes, and keys is also created, as associated Column, Index, UniqueKey, ForeignKey, and KeyAssociation metadata objects, but these are not included in the listing. To see what associated metadata was created for a table, follow the steps for obtaining each table's SASTableName value using the METABROWSE command or SAS Management Console and refresh the display to show the new data. The property sheets for the new tables will contain information about associated metadata.

## Table Update Report Description

The METALIB procedure's update functionality updates table attribute values as well as the attribute values of associated objects to match those found in the specified physical library. The following is an example of the SAS output listing that is generated for a PROC METALIB update operation when REPORT is specified.

```
                             The METALIB Procedure

                            Summary Report for Library sas91 lib2
                                Repository Meta Proc repos
                                         25JAN2005

                                Metadata Summary Statistics

                             Total tables analyzed           3
                             Tables Updated                  3
                             Tables Deleted                  0
                             Tables Added                    0
                             Tables matching data source     0
                             Tables not found                0


-------------------------------------------------------------------------------------
                                  Tables Updated

-------------------------------------------------------------------------------------
       Table                   |    Updates
-------------------------------------------------------------------------------------


 Name        | Metadata ID     |Metadata | Metadata ID  | SAS Name | Metadata | Change
             |                 |Name     |              |          |  Type    |
-------------------------------------------------------------------------------------

NDX_MULTICOL|A5HJ58JU.AX001H35|multi_col|A5HJ58JU.B30011T5|Multi--col |Index |Column
                                                                             silver
                                                                             added
SCOTT    |A5HJ58JU.AX001FJN|a         |A5HJ58JU.AY001H0L|a      |Column|SASColumnLength
                           |b         |A5HJ58JU.AY001JBR|b      |Column|IsNullable

UKEYS    |A5HJ58JU.AX001D8C|UKEYS.stat_key|A5HJ58JU.B4000U3D|stat_key|UniqueKey|Added
```

In the output listing:

Name
  is the metadata name of the table definition that was modified.

Metadata ID
  is the table definition's unique 17-character metadata identifier.

Metadata Name
  is the metadata name of the affected associated object.

Metadata ID
  is the associated object's unique 17-character metadata identifier.

SAS Name
  is the SAS name of the item described by the metadata.

- For an index, this is the value stored in the IndexName= attribute.
- For a column, this is the value stored in the SASColumnName= attribute.
- For a non-primary unique key, this is a two-part identifier in the form *SASTableName.data-source-key-name*.
- For a primary unique key, this is a two-part identifier in the form *SASTableName*.Primary.
- For a foreign key, this is a two-part identifier in the form *primary-table-SASTableName.foreign-table-SASTableName*.

Metadata Type
  is the metadata type of the associated object.

Change
: is a system-generated description of the change that was made. This can be a one-word description, such as “Added” or “Deleted”, or an attribute name, to indicate that the attribute’s value was modified. It can also be a “Column” or a “Column Order” message, followed by the name of the column that was affected by a change. The METALIB procedure changes a table’s Columns association to make the metadata column order match the data source column order. Affected columns are listed separately in the listing. The column order in the listing indicates the new metadata column order.

The information in this particular example is read as follows:

1. An index associated with the table named NDX_MULTICOL, which has the ID A5HJ58JU.AX001H35, was updated to add a definition for column named “Silver.” The associated index has the SAS Name “multi_col” and the metadata ID A5HJ58JU.B30011T5.
2. The table named SCOTT had two updates:
   a. The associated column that has the SAS Name “a” and the metadata ID A5HJ58JU.AY001H0L had its SASColumnLength attribute updated.
   b. The associated column that has the SAS Name “b” and the metadata ID A5HJ58JU.AY001JBR had its IsNullable attribute updated.
3. A UniqueKey definition was added to the table named UKEYS. This UniqueKey has the metadata name UKEYS.stat_key, the metadata ID A5HJ58JU.B4000U3D, and the SAS Name “stat_key”.

## Table Deletion Report Description

The following is an example of the output listing that is created when PROC METALIB is executed by specifying UPDATE_RULE=(DELETE) without a SELECT statement. REPORT and NOEXEC are also specified, which means that the listing will report changes that are needed but have not yet been applied in the SAS Metadata Repository.

```
                                The METALIB Procedure

                  Summary Report of Potential Changes for Library sas91 lib2
                                Repository Meta Proc repos
                                         15FEB2005

                               Metadata Summary Statistics

                            Total tables analyzed           3
                            Tables to be Updated            0
                            Tables to be Deleted            3
                            Tables to be Added              0
                            Tables matching data source     0
                            Tables not found                0


-------------------------------------------------------------------------------------
                                 Tables to be Deleted
-------------------------------------------------------------------------------------


Metadata Name                       Metadata ID          SAS Name

SCOTT3                              A5HJ58JU.AX001IMQ    SCOTT
SCOTT2                              A5HJ58JU.AX001IMP    SCOTT
SCOTT                               A5HJ58JU.AX001FJN    SCOTT
```

In the output listing:

Metadata Name
: is the value in the obsolete table definition's Name attribute.

Metadata ID
: contains the obsolete table definition's unique 17-character metadata identifier.

SAS Name
: is the value in the obsolete table definition's SASTableName attribute.

Information about associated metadata that is deleted (Column, Index, and key metadata) is not included in the listing. To see what associated metadata would be deleted by the procedure, view the tables' property sheets using either the METABROWSE command or the SAS Management Console.

This particular output listing indicates that a physical table with a SAS name of "SCOTT" could not be found in the specified library, although a table definition exists for it. In fact, three table definitions have the value "SCOTT" in the SASTableName attribute: "SCOTT," "SCOTT2," and "SCOTT3," and all three exist in repository A5HJ58JU. To delete the table definitions and their associated metadata, execute PROC METALIB again omitting the NOEXEC option.

## Examples

### Synchronizing Metadata with the Physical Data Source

To periodically check for and correct any discrepancies between the table definitions stored for a specified SASLibrary in SAS Metadata Repositories with the tables at the physical data source, issue the METALIB procedure with the following statements:

```
ods html file="C:\update_rep.html";
ods listing close;

proc metalib;
  omr (libid="library-identifier"
        repid="repository-identifier");
  update_rule=(delete);
  report;
run;
```

The OMR statement identifies the library that will be examined and the repository where the library definition is stored. Because server connection statements are omitted from the OMR statement, the procedure will connect to the metadata server using the parameters set in the METASERVER, METAPORT, METAUSER, and METAPASS system options.

The UPDATE_RULE statement instructs the metadata server to delete obsolete table definitions (table definitions in the metadata repository that do not correspond to a physical table in the specified SAS library) in addition to the default actions of add and update.

The REPORT statement instructs the procedure to write the changes that were made to the metadata. The ODS statement specifies the file name and format for the report. Unless SAS is instructed otherwise, the report will also be written to a SAS output listing. In this example, the statement **`ods listing close;`** suppresses creation of the SAS output listing.

## Identifying Needed Metadata Updates Without Applying Changes

To check for needed updates without making any changes, submit the statements shown in "Synchronizing Metadata with the Physical Data Source" on page 260 and also include the NOEXEC statement as in the example below:

```
ods html file="C:\noexec_rep.html";

proc metalib;
  omr (libid="library-identifier"
       repid="repository-identifier");
  update_rule (delete);
  report;
  noexec;
run;
```

The NOEXEC statement instructs PROC METALIB not to apply the changes to the SAS Metadata Repository.

The REPORT statement writes information about needed changes both to the SAS log and to the file named in the ODS statement. The output will consist of three listings like the ones shown in "Understanding REPORT Statement Output" on page 256.

## Using PROC METALIB to Add or Update a Table

To use PROC METALIB to add or update metadata for a physical table that has the SAS name "mytable", issue the procedure as follows:

```
proc metalib;
  omr (libid="library-identifier"
       repid="repository-identifier");
  select (mytable);
  report;
run;
```

When you omit the UPDATE_RULE statement, PROC METALIB seeks to update or add the specified metadata. The procedure will first look for an existing table definition that stores the value "mytable" in its SASTableName attribute. If a table definition cannot be found, the procedure will create a new table definition.

## Using PROC METALIB to Update the Metadata for a Specific Table

To update the metadata for a specific table, it is recommended that you issue PROC METALIB with a SELECT statement that specifies the table definition's metadata identifier as follows:

```
proc metalib;
  omr (libid="library-identifier"
       repid="repository-identifier");
  select (A7892350.B00265DX);
  report;
run;
```

The first part of the two-part metadata identifier (A7892350) identifies the repository that contains the table definition; the second part (B00265DX) identifies the table definition in the repository. By using a metadata identifier, if duplicate table definitions exist, you are assured that you are updating the correct one. You can also directly specify table definitions that exist in dependent repositories using a metadata identifier.

When you use a metadata identifier to identify a table definition that is stored in a dependent repository, be sure to use the appropriate repository ID. The SAS

Management Console Data Library Manager plugin does not list table definitions that exist in dependent repositories. See "Listing Table Definitions in Dependent Repositories" on page 255 for instructions to obtain the metadata identifiers of table definitions that exist in dependent repositories.

### Deleting Obsolete Table Definitions

To delete obsolete table definitions that are associated with the specified SASLibrary, issue PROC METALIB with the UPDATE_RULE statement, as follows:

```
proc metalib;
  omr (libid="library-identifier"
       repid="repository-identifier");
  update_rule (delete noadd noupdate);
  report;
run;
```

Specifying DELETE invokes the delete metadata action. Specifying NOADD and NOUPDATE suppresses the default add and update actions. Omitting the SELECT statement instructs the procedure to examine all tables and table definitions. See "How PROC METALIB Works" on page 246 for a description of how obsolete table definitions are identified.

### Excluding Tables from Processing

By default, PROC METALIB seeks to update all table definitions associated with the specified SASLibrary object. It will also add table definitions for tables that do not have definitions in the SAS Metadata Repository. To exclude one or more tables from being processed, for example, if you do not want a metadata definition created for a particular physical table, issue the procedure with the following statements:

```
proc metalib;
  omr (libid="library-identifier"
       repid="repository-identifier");
  exclude (mytable);
  report;
run;
```

The EXCLUDE statement specifies to exclude the physical table that has the SAS name "mytable" from processing. If a table definition exists that has the value "mytable" in its SASTableName attribute, it will not be updated. If a table definition does not exist, one will not be created. However, definitions will continue to be updated and created for other physical tables found in the specified library.

## METAOPERATE Procedure

The METAOPERATE procedure enables you to perform administrative tasks associated with the SAS Metadata Server and SAS 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

Use the following syntax to execute the METAOPERATE Procedure. Note that unlike most SAS procedures, which end individual statements with a semicolon (;), the PROC METAOPERATE statements are all submitted as one string. That is, only one semicolon is used, and this semicolon is placed after the last statement that is submitted before the RUN statement. See the Examples section for specific usage.

**PROC METAOPERATE**

Server Connection Statements

<**SERVER=** "*host-name*">

<**PORT=** *port-number*>

<**USERID=** "*userid*">

<**PASSWORD=**"*password*">

<**PROTOCOL=** BRIDGE>

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*"><;>

Execution Statement

**RUN;**

## 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" on page 305. 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.

*Note:* 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=
: specifies 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=
: specifies the TCP port to which the SAS Metadata Server listens for connections. For example, port=8561. This is the port number that was used to start the SAS Metadata Server.

USERID=
: specifies an authenticated user ID. See the *SAS Intelligence Platform: Security Administration Guide* for information about authentication requirements.

PASSWORD=
: specifies the password that corresponds to the authenticated user ID.

PROTOCOL=
: specifies 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=
: specifies the action that you want to perform. ACTION is a required statement.

*Note:* A user must have *administrative user* status on the metadata server in order to execute all actions except STATUS. For more information about this privilege, see the *SAS Intelligence Platform: Security Administration 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 new state are identified in a <Repository> XML element that is passed to the metadata server in the OPTIONS statement.

When one or more repositories are paused by 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. Executing a PAUSE action without specifying a <Repository> element 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 the repository's PauseState attribute. For more information, see "GetRepositories" on page 156 and "METADATA Procedure" on page 240.

For examples of specifying a PAUSE action, see "Examples of ACTION=PAUSE" on page 267.

*Note:* A repository that is paused must be resumed to restore client activity to its normal state. A repository is resumed by executing PROC METAOPERATE and specifying ACTION=RESUME. △

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 "GetRepositories" on page 156 and "METADATA Procedure" on page 240. The repository to resume is identified in a <Repository> XML element that is passed to the metadata server in the OPTIONS statement. Specifying the RESUME action without a <Repository> element 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. For an example of specifying a RESUME action, see "Example of ACTION=RESUME" on page 268.

REFRESH
: The REFRESH action has two uses:

- □ It can pause client activity on the SAS Metadata Server long enough to invoke the server configuration option(s) specified in the OPTIONS statement 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 can be used to pause and resume client activity in one or more repositories in a single step. This is useful for recovering memory on the metadata 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.

*Note:* Do not use PROC METAOPERATE to delete a project repository. Project repositories should be deleted by using SAS Management Console. △

PURGE
: removes logically deleted metadata records from the specified repository without disturbing the current metadata records. For an example of specifying a PURGE action, see "Example of ACTION=PURGE" on page 267.

EMPTY
: deletes the metadata records in the specified repository without disturbing the repository's registration in the repository manager. For an example of specifying a EMPTY action, see "Example of ACTION=EMPTY" on page 267.

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. For an example of specifying a STOP action, see “Example of ACTION=STOP” on page 269.

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 <Repository> XML element is passed in the OPTIONS statement.

OPTIONS=
: Specifies 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"/>
```

ID=
: is required for each of the PAUSE, RESUME, and REFRESH actions, 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 access 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. For more information, see “Using the ARM_OMA Subsystem to Obtain Raw Performance Data for the SAS Metadata Server” in the *SAS Intelligence Platform: System Administration Guide*.

*Note:* To ensure that the options string is parsed correctly by the metadata server, all 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, PURGE, or EMPTY. REPOS= is an alias for REPOSITORY=.

## Execution Statement

RUN;
: executes the METAOPERATE procedure.

## Examples

### Example of ACTION=PURGE

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=8591
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=PURGE
   REPOS="MyRepos";

RUN;
```

### Example of ACTION=EMPTY

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;
```

### Examples of ACTION=PAUSE

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=8591
   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 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=8591
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

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

RUN;
```

## Example of ACTION=RESUME

The following example connects to the server and issues a RESUME action to restore client activity on all repositories and the repository manager:

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

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

RUN;
```

## Example of ACTION=STATUS

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=8591
   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.
```

## Example of ACTION=STOP

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=8591
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

   ACTION=STOP;

RUN;
```

CHAPTER
12

# DATA Step Functions

## SAS Metadata DATA Step Functions

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 "Overview of SAS Namespace Submodels" on page 46.

### 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 server 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" on page 305.

## 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 "Filtering a GetMetadataObjects Request" in "Querying All Metadata of a Specified Type" 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 function.

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.

| 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 |

## METADATA_DELASSN Function

Deletes all objects that make up the specified association

### Syntax

rc = METADATA_DELASSN(uri,asn);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| asn | in | Association name |

### Return Values

| Value | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -2 | The deletion was unsuccessful. See the SAS log for details |
| -3 | No objects match the URI |

### Example

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

data _null_;
    length uri $256
           curi $256
           curi1 $256
           curi2 $256;
    rc=0;

    /* Create a PhysicalTable object. */

    rc=metadata_newobj("PhysicalTable",
                       uri,
                       "My Table");
    put rc=;
    put uri=;

    /* Create a couple of columns on the new PhysicalTable object. */

    rc=metadata_newobj("Column",
                       curi,
                       "Column1",
                       "myrepos",
                       uri,
                       "Columns");

    put rc=;
    put curi=;

    rc=metadata_newobj("Column",
                       curi1,
                       "Column2",
                       "myrepos",
                       uri,
                       "Columns");

    put rc=;
    put curi1=;

    rc=metadata_newobj("Column",
                       curi2,
                       "Column3",
                       "myrepos",
                       uri,
                       "Columns");

    put rc=;
    put curi2=;

    rc=metadata_delassn(uri,"Columns");
    put rc=;
    rc=metadata_delobj(uri);
    put rc=;
run;
```

## Related Functions

- "METADATA_SETASSN Function" on page 299
- "METADATA_GETNASN Function" on page 282

## METADATA_DELOBJ Function

Deletes the first object specified by the input URI

### Syntax

rc = METADATA_DELOBJ(uri);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |

### Return Values

| Value | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -2 | The deletion was unsuccessful. See the SAS log for details |
| -3 | No objects match the given URI |

### Example

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

data _null_;
```

```
    rc=metadata_delobj("omsobj:Property?@Name='My Object'");
    put rc=;

run;
```

## Related Functions

- □ "METADATA_DELASSN Function" on page 276
- □ "METADATA_GETNOBJ Function" on page 286
- □ "METADATA_GETNTYP Function" on page 290
- □ "METADATA_NEWOBJ Function" on page 293

# METADATA_GETATTR Function

## Syntax

rc = METADATA_GETATTR(uri, attr, value);

## Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| attr | in | Attribute of the metadata object |
| value | out | Value of the specified attribute |

## Return Values

| Argument | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -2 | The attribute was not found |
| -3 | No objects matches the URI |

## Example

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

data _null_;

    length name $200
        desc $200
        modified $100;

    rc=metadata_getattr("omsobj:Machine?@Name='bluedog'","Name",name);
    put rc=;
    put name=;

    rc=metadata_getattr("omsobj:Machine?@Name='bluedog'","Desc",desc);
    put rc=;
    put desc=;

    rc=metadata_getattr("omsobj:Machine?@Name='bluedog'","MetaUpdated",modified);
    put rc=;
    put modified=;

run;
```

## Related Functions

- □ "METADATA_GETNATR Function" on page 284
- □ "METADATA_SETATTR Function" on page 301

## METADATA_GETNASL Function

Returns the *n*th named association for the object URI

## Syntax

rc = METADATA_GETNASL(uri, n, asn);

## Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| n | in | Numeric index value |
| asn | out | Association name |

## Return Values

| Value | Description |
|---|---|
| n | The number of objects that match the URI |
| -1 | Unable to connect to the metadata server |
| -3 | No objects match the given URI |
| -4 | n is out of range |

## Details

Use the METADATA_GETNASL function to iterate through all of the possible associations of an object.

## Example:

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

data _null_;
    length assoc $256;
    rc=1;
    n=1;
```

```
      do while(rc>0);

          /* Walk through all possible associations of this object. */

          rc=metadata_getnasl("omsobj:Machine?@Name='bluedog'",
                              n,
                              association);
          put assoc=;
          n=n+1;
      end;
  run;
```

## Related Functions

- “METADATA_GETNASN Function” on page 282
- “METADATA_SETASSN Function” on page 299

## METADATA_GETNASN Function

Returns the *n*th associated object of the association specified

### Syntax

rc = METADATA_GETNASN(uri, asn, n, nuri);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| asn | in | Association name |
| n | in | Numeric index value |
| nuri | out | The Universal Resource Identifier of the *n*th associated object |

## Return Values

| Value | Description |
|---|---|
| n | The number of associated objects |
| -1 | Unable to connect to the metadata server |
| -3 | No objects match the given URI |
| -4 | n is out of range |

## Details

Use the METADATA_GETNASN function to iterate through the associated objects on a metadata object.

## Example

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

data _null_;
    length uri $256
      text $256;
    rc=1;
    arc=0;
    n=1;

    do while(rc>0);

        /* Walk through all the notes on this machine object. */

        rc=metadata_getnasn("omsobj:Machine?@Name='bluedog'",
                            "Notes",
                            n,
                            uri);

        arc=1;
        if (rc>0) then arc=metadata_getattr(uri,"StoredText",text);
        if (arc=0) then put text=;
        n=n+1;
    end;
run;
```

## METADATA_GETNATR Function

Returns the *n*th attribute on the object specified by the URI

### Syntax

rc = METADATA_GETNATR(uri, n, attr, value);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| n | in | Numeric index value |
| attr | out | Attribute of the metadata object |
| value | out | Value of specified attribute |

### Return Values

| Value | Description |
|---|---|
| n | The number of attributes on the given URI |
| -1 | Unable to connect to the metadata server |
| -2 | No attributes are defined on the object |
| -3 | No objects match the given URI |
| -4 | n is out of range |

## Details

Use the METADATA_GETNATR function to iterate through all of the attributes defined for an object.

## Examples

### Example 1: Using an Object URI

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

data _null_;
    length attr $256
        value $256;
    rc=1;
    n=1;
    do while(rc>0);

        /* Walk through all the attributes on this machine object. */

        rc=metadata_getnatr("omsobj:Machine?@Name='bluedog'",
                            n,
                            attr,
                            value);

        if (rc>0) then put n=;
        if (rc>0) then put attr=;
        if (rc>0) then put value=;

        n=n+1;

    end;
run;
```

### Example 2: Using a Repository URI

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

data _null_;

    length id $20
    type $256
    attr $256
    value $256;

    rc=metadata_resolve("omsobj:RepositoryBase?@Name='myrepos'",type,id);

    put rc=;
    put id=;
    put type=;
    n=1;
    rc=1;
    do while(rc>=0);

        rc=metadata_getnatr("omsobj:RepositoryBase?@Name='myrepos'",n,attr,value);
        if (rc>=0) then put attr=;
        if (rc>=0) then put value=;
        n=n+1;
    end;
run;
```

## Related Functions

- □ "METADATA_GETATTR Function" on page 279
- □ "METADATA_SETATTR Function" on page 301

## METADATA_GETNOBJ Function

Returns the *n*th object matching the specified URI

### Syntax

rc = METADATA_GETNOBJ(uri, n, nuri);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| n | in | Numeric index value |
| nuri | out | n th object matching the given URI |

## Return Values

| Value | Description |
|---|---|
| n | The number of objects matching the input URI |
| -1 | Unable to connect to the metadata server |
| -3 | No objects match the given URI |
| -4 | n is out of range |

## Details

Use the METADATA_GETNOBJ function to iterate through all of the objects that match the given URI.

## Examples

### Example 1 : Determining How Many Machine Objects Exist

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

data _null_;
    length uri $256;
    nobj=0;
    n=1;

    /* Determine how many machine objects are in this repository. */

    nobj=metadata_getnobj("omsobj:Machine?@Id contains '.'",n,uri);
    put nobj=;   /* Number of machine objects found. */
    put uri=;    /* URI of the first machine object. */

run;
```

### Example 2 : Looping Through Each Repository on a Server

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

data _null_;
    length uri $256;
    nobj=1;
    n=1;

    /* Determine how many repositories are on this server. */

    do while(nobj >= 0);

        nobj=metadata_getnobj("omsobj:RepositoryBase?@Id contains '.'",n,uri);
        put nobj=;   /* Number of repository objects found. */
        put uri=;    /* Nth repository.                     */
        n=n+1;
    end;
run;
```

## Related Functions

- □ "METADATA_DELOBJ Function" on page 278
- □ "METADATA_NEWOBJ Function" on page 293

# METADATA_GETNPRP Function

Returns the *n* th property on the object specified by the input URI

## Syntax

rc = METADATA_GETNPRP(uri, n, prop, value);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| n | in | Numeric index value |
| prop | out | Abstract property string |
| value | out | Value of specified property string |

## Return Values

| Value | Description |
|---|---|
| n | The number of properties for the given URI |
| -1 | Unable to connect to the metadata server |
| -2 | No properties are defined for the object |
| -3 | No objects match the given URI |
| -4 | n is out of range |

## Details

Use the METADATA_GETNPRP function to iterate through all properties defined on an object.

## Example

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

data _null_;
    length prop $256
        value $256;
    rc=1;
    n=1;

    do while(rc>0);

        /* Walk through all the properties on this machine object. */

        rc=metadata_getnprp("omsobj:Machine?@Name='bluedog'",
                            n,
                            prop,
                            value);

        if (rc>0) then put n=;
        if (rc>0) then put prop=;
        if (rc>0) then put value=;
        n=n+1;
```

```
end;
run;
```

## Related Functions

- □ "METADATA_GETPROP Function" on page 291
- □ "METADATA_SETPROP Function" on page 302

## METADATA_GETNTYP Function

Returns the *n*th object type on the server

### Syntax

rc = METADATA_GETNTYP(n, type);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| n | in | Numeric index value |
| type | out | Metadata type |

### Return Values

| Value | Description |
|---|---|
| n | The number of objects matching the input URI |
| -1 | Unable to connect to the metadata server |
| -4 | n is out of range |

## Details

Use the METADATA_GETNTYP function to iterate through all possible object types on a server.

## Example

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

data _null_;
    length type $256;
    rc=1;
    n=1;

    do while(rc>0);

        /* Walk through all possible types on this server. */
        rc=metadata_getntyp(n,type);
        put type=;
        n=n+1;
    end;
run;
```

# METADATA_GETPROP Function

Returns the named property for the object specified by the input URI

## Syntax

rc = METADATA_GETPROP(uri, prop, value);

## Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| prop | in | Abstract property string |
| value | out | Value of specified property string |

## Return Values

| Value | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -2 | Named property is undefined |
| -3 | No objects match the given URI |

## Example

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

data _null_;
    length value $200;
    rc=metadata_getprop("omsobj:Machine?@Name='bluedog'","Property 1",value);
    if rc=0 then put value=;
run;
```

## Related Functions

- "METADATA_GETNPRP Function" on page 288
- "METADATA_SETPROP Function" on page 302

## METADATA_NEWOBJ Function

Creates a new metadata object

### Syntax

rc = METADATA_NEWOBJ(type, uri<,name><,repos><,parent><,asn>);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| type | in | The metadata type |
| uri | out | Universal Resource Identifier |
| name | in | Name attribute for the new metadata object |
| repos | in | Repository identifier of an existing repository; by default, the new object is created in the default repository |
| parent | out | Parent of the new metadata object |
| asn | in | Association name |

### Return Values

| Value | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -2 | Unable to create object, see the SAS log for details |

## Example

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

data _null_;
    length uri $256;
        curi $256;
    rc=0;

    /* Create a PhysicalTable object. */

    rc=metadata_newobj("PhysicalTable",
                       uri,
                       "My Table");
    put uri=;

    /* Create a couple of columns on the new PhysicalTable object. */

    rc=metadata_newobj("Column",
                       curi,
                       "Column1",
                       "myrepos",
                       uri,
                       "Columns");
    put curi=;

    rc=metadata_newobj("Column",
                       curi,
                       "Column2",
                       "myrepos",
                       uri,
                       "Columns");
    put curi=;

    rc=metadata_newobj("Column",
                       curi,
                       "Column3",
                       "myrepos",
                       uri,
                       "Columns");
    put curi=;
run;
```

### Related Functions

- “METADATA_DELOBJ Function” on page 278
- “METADATA_GETNOBJ Function” on page 286

## METADATA_PAUSED Function

Determines whether the server specified by the METASERVER system option is paused

### Syntax

rc = METADATA_PAUSED();

#### Return Values

| Value | Description |
|---|---|
| 0 | Server is not paused |
| 1 | Server is paused |
| -1 | Unable to connect to the metadata server |

### Example

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

data _null_;
   rc=metadata_paused();
   if rc eq 0 then put 'server is not paused';
   else if rc eq 1 then put 'server is paused';
run;
```

## METADATA_PURGE Function

Purges the specified URI

## Syntax

rc = METADATA_PURGE(<uri>);

## Argument

If no argument is specified, the entire connection is purged from the cache.

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |

## Return Value

| Value | Description |
|---|---|
| 0 | Object successfully purged |

## Details

For performance reasons, metadata objects are cached by URI within a DATA step. To refresh the metadata object with the latest data from the server, purge the URI with the METADATA_PURGE function.

## Example

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

data _null_;
    length association $256;
    rc=1;
    n=1;

    do while(rc>0);

        /* This will make this DATA step run much slower by        */
        /* purging the object cache, which requires the metadata   */
        /* server to be accessed again.                            */
        /* Compare run timings by commenting out the purge.       */

        rc=metadata_purge("omsobj:Machine?@Name='bluedog'");

        /* Walk through all possible associations of this object. */

        rc=metadata_getnasl("omsobj:Machine?@Name='bluedog'",
                            n,
                            association);
        put asn=;
        n=n+1;
    end;
run;
```

## METADATA_RESOLVE Function

Resolves a metadata URI into a specific object on the current metadata server

### Syntax

rc = METADATA_RESOLVE(uri, type, id);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| type | out | Metadata type |
| id | out | Unique identifier for an object |

### Return Values

| Value | Description |
|---|---|
| N | Number of objects that match the given URI |
| 0 | No objects match the given URI |
| -1 | Unable to connect to the metadata server |

## Examples

### Example 1: Using an Object URI

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

data _null_;
    length id $20
    type $256;
    rc=metadata_resolve("omsobj:Machine?@Name='bluedog'",type,id);
    put rc=;
    put id=;
    put type=;
run;
```

### Example 2: Using a Repository URI

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

data _null_;

    length id $20
    type $256
    attr $256
    value $256;

    rc=metadata_resolve("omsobj:RepositoryBase?@Name='myrepos'",type,id);

    put rc=;
    put id=;
    put type=;
    n=1;
    rc=1;
    do while(rc>=0);

        rc=metadata_getnatr("omsobj:RepositoryBase?@Name='myrepos'",n,attr,value);
        if (rc>=0) then put attr=;
        if (rc>=0) then put value=;
        n=n+1;

    end;
run;
```

## METADATA_SETASSN Function

Modifies an association list for an object

### Syntax

rc = METADATA_SETASSN(uri, asn, mod, auri1<,auri2,...aurin>);

### Arguments

| Argument | Direction | Description | |
|---|---|---|---|
| uri | in | Universal Resource Identifier | |
| asn | in | Association name | |
| mod | in | Modification to be performed on the metadata object; valid values are | |
| | | APPEND | appends the specified associations to the specified object's association element list without modifying any of the other associations on the list. |
| | | REMOVE | deletes the specified associations from the specified object's association element list without modifying any of the other associations on the list. |
| aurin | in | The Universal Resource Identifier of the associated object | |

### Return Values

Number of objects matching the input URI.

| Value | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -3 | No objects match the input URI |

| Value | Description |
|---|---|
| -4 | Unable to perform modification; see the SAS log for details |
| -5 | Invalid modification |
| -6 | Unable to resolve association list URIs |

## Example

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

data _null_;
    length uri $256;
    rc=0;

    /* Create a TextStore object. */

    rc=metadata_newobj("TextStore",
                       uri,
                       "My TextStore");
    put uri=;

    rc=metadata_setassn("omsobj:Machine?@Name='bluedog'",
                        "Notes",
                        "Append",
                        uri);
    put rc=;

    rc=metadata_setassn("omsobj:Machine?@Name='bluedog'",
                        "Notes",
                        "Remove",
                        uri);
    put rc=;

run;
```

### Related Functions

- □ "METADATA_DELASSN Function" on page 276
- □ "METADATA_GETNASN Function" on page 282

## METADATA_SETATTR Function

Sets the named attribute for the object specified by the input URI

### Syntax

rc = METADATA_SETATTR(uri, attr, value);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| attr | in | Attribute of the metadata object |
| value | in | Value of specified attribute |

### Return Values

| Value | Description |
|---|---|
| 0 | Successful completion |
| -1 | Unable to connect to the metadata server |
| -2 | Unable to set the attribute |
| -3 | No objects match the given URI |

## Example

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

data _null_;
    rc=metadata_setattr("omsobj:Machine?@Name='bluedog'","Desc",
"My New Description");     put rc=; run;
```

## Related Functions

□ "METADATA_GETATTR Function" on page 279
□ "METADATA_GETNATR Function" on page 284

## METADATA_SETPROP Function

Sets the named property for the object specified by the input URI

### Syntax

rc = METADATA_SETPROP(uri, prop, value);

### Arguments

| Argument | Direction | Description |
|---|---|---|
| uri | in | Universal Resource Identifier |
| prop | in | Abstract property string |
| value | in | Value of specified property string |

## Return Values

| Value | Description |
|---|---|
| 1 | New property was created and set |
| 0 | Existing property was successfully set |
| -1 | Unable to connect to the metadata server |
| -2 | Unable to set the attribute |
| -3 | No objects match the given URI |
| -4 | Unable to create a new property |

## Example

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

data _null_;
    rc=metadata_setprop("omsobj:Machine?@Name='bluedog'","New Property",
    "my value");     put rc=; run;
```

## Related Functions

- “METADATA_GETPROP Function” on page 291
- “METADATA_GETNPRP Function” on page 288

## METADATA_VERSION Function

Returns the server model version number

## Syntax

ver = METADATA_VERSION();

## Return Values

| Value | Description |
|---|---|
| ver | Server model version number |
| -1 | Unable to connect to the metadata server |

## Example

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

data _null_;
   ver=metadata_version();
   put ver=;
run;
```

# CHAPTER 13 System Options

## 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 the current release.

The following table summarizes the options required by each protocol.

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

| System option | COM | BRIDGE |
|---|---|---|
| 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 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 Language Reference: Dictionary*.

# Your Turn