SAS® 9.4 Open Metadata Interface: Reference and Usage, Third Edition

SAS® Documentation
December 4, 2019

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 Open Metadata Interface: Reference and Usage, Third Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 Open Metadata Interface: Reference and Usage, Third Edition

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

December 2019

9.4-P5:omaref

Contents

What's New in SAS 9.4 Open Metadata Interface: Reference and Usage

Overview

Beginning in SAS 9.4M6, the SAS Java Metadata Interface requires Java 2 SDK, Standard Edition, Version 1.8 (JDK 1.8.0) or higher.

Beginning in SAS 9.4M3, the Status method supports an XML element in the OPTIONS parameter: <CLUSTER/>. The Status method supports a new XML element in the INMETA parameter: <SynchCheck>.

Beginning in SAS 9.4M2, the IServer Status method supports new optional XML elements that enable you to configure additional alert email reminders for the condition `the journal commit task stopped running`. The method accepts a new <AlertConditions/> subelement within the <Scheduler/> element. The <XMLSELECT> documentation was enhanced.

SAS 9.4 enhancements to the SAS Open Metadata Interface are as follows:

- the IServer server interface, which contains methods that control and get information about the SAS Metadata Server, supports a clustered SAS Metadata Server configuration in addition to a single SAS Metadata Server configuration.
- the IOMI server interface, which contains methods for reading and writing metadata, has a new method and an enhanced method.

SAS 9.4M6

The current release of the Java client software requires Java 2 SDK, Standard Edition, Version 1.8 (JDK 1.8.0) or higher. Earlier SAS 9.4 releases of the Java client software allow JDK 1.6.0 or higher.

SAS 9.4M3

Beginning in SAS 9.4M3, the Status method supports an XML element in the OPTIONS parameter.

<CLUSTER/>
specifies to execute the request in the INMETA parameter on the master server. In a clustered metadata server configuration, a slave node processes Status requests by default. A load-balancing algorithm that operates outside of the SAS Open Metadata Interface controls which node gets the request. Some cluster XML elements, such as <CLUSTER List=" "/> and <CLUSTER CurrentNodes= " "/>, return meaningful information only when they are processed by the master server. When specified in a Status request, the <CLUSTER/> element submits the contents of the INMETA parameter directly to the master server. There is no need to know or specify server connection parameters. In previous releases of SAS 9.4, you had to use PROC METADATA with METHOD=STATUS, specify the NOREDIRECT argument, and specify connection parameters for the master server to connect to the master server. This parameter has no effect in a single metadata server configuration.

The SAS Metadata Server cluster has a synchronization check feature. This feature is invoked through the SAS Management Console Metadata Manager Analyze/ Repair wizard or the sas-analyze-metadata batch tool. The Status method supports the following XML element in the INMETA parameter to return information about synchronization check:

<SynchCheck><Results *OptionalAttribute(s)=" "*/></SynchCheck>
reports the results of the last synchronization check on the slave node that received the Status query. By default, the query returns the Id value and Name value of all repositories that were examined and a <Container/> XML element that specifies the name of any metadata type container in which an error was found. Optional attributes are available to request comprehensive output and to request information for a named repository.

For information about <CLUSTER/>, see "Status Method" on page 255. For more information about <SynchCheck/>, see the following topic for the Status method: "Cluster Information Elements" on page 259.

SAS 9.4M2

Beginning in SAS 9.4M2, the SAS Metadata Server sends alert email reminders in addition to the initial alert email message for the alert condition `the journal commit task stopped running`. The SAS Metadata Server terminates itself if the alert condition is not addressed within a defined grace period. The IServer Status method provides the following optional XML elements in the INMETA parameter to return information about the omaconfig.xml options that configure that functionality.

<OMA ALERT_CONDITION_FREQUENCY=" "/>
Returns the amount of time that will elapse before the initial and subsequent alert email reminders about the alert condition will be sent. The time value is returned in seconds. The default value is 21,600 seconds (six hours).

<OMA ALERT_CONDITION_GRACE_PERIOD=" "/>
Returns the amount of time that the alert condition will be allowed to persist before the SAS Metadata Server shuts itself down. The time value is returned in seconds. The default value is 259,200 seconds (three days).

The Status method also accepts a new subelement within the <Scheduler/> XML element to return information about alert conditions:

<Scheduler><AlertConditions/></Scheduler>
Specify the <AlertConditions/> subelement to determine whether an alert condition exists on the specified SAS Metadata Server. If an alert condition exists, the subelement returns an <AlertCondition/> XML element and an <ExpirationTime/> XML element. The <AlertCondition/> element includes the error and a datetime value representing the time at which the error occurred. The <ExpirationTime/> element includes the server's scheduled termination time.

For more information, see "Status Method" on page 255.

The XMLSELECT documentation was enhanced to include the new topic: "Controlling XMLSELECT Query Size" on page 387.

SAS 9.4 Enhancements to the IServer Server Interface

In SAS 9.4, the Pause, Refresh, Resume, and Stop methods support new XML elements in the OPTIONS parameter that enable users to specify whether the methods should be executed only on the connected SAS Metadata Server or on all SAS Metadata Servers in the cluster:

<CLUSTER/>
Specifies to execute the request on all SAS Metadata Servers in the cluster.

<SINGLE_SERVER/>
Specifies to execute the request only on the connected SAS Metadata Server.

For more information about these optional elements, see "Pause()" on page 241, "Refresh Method" on page 244, "Resume Method" on page 253, and "Stop Method" on page 271.

The IServer Status method supports new XML elements in the INMETA parameter to return information about a server cluster and the cluster's status in relationship to the quorum:

<CLUSTER *attributes*/>
Returns values for specified cluster attributes. Valid attributes are CLUSTERGUID=, DEFINED_NODES=, CURRENT_NODES=, HAS_FIRST_NODE=, HAS_QUORUM=, and LIST=.

<CLUSTERSTATE/>
Returns information about the cluster's availability. Valid return values are STARTING, QUORUM, or LOSTQUORUM.

<OMA MAXIMUM_CLUSTER_NODES=" "/>
Returns the maximum number of servers that are supported in the cluster as configured in the omaconfig.xml file. Fewer servers are initially defined than are supported. A customer can later add servers up to this number plus one (the additional slot is for the master server).

For more information about the new Status elements for clustering, see the following topic for the Status method: "Cluster Information Elements" on page 259. For more information about server clustering, see "Metadata Server Clustering" on page 240.

In SAS 9.4, the Status method supports new XML elements in the INMETA parameter that return performance metrics:

<OMA SERVER_STARTED=" "/>
Returns the date and time the metadata server was started as a SAS datetime value (the number of seconds since January 1, 1960).

<OMA SERVER_UPTIME=" "/>
Returns the amount of time that has elapsed since the metadata server was started in seconds.

For more information, see the following topic for the Status method: "Server Process Statistics Elements" on page 265.

SAS 9.4 Enhancements to the IOMI Server Interface

SAS 9.4 provides a new method:

ChangePassPhrase
Re-encrypts all passwords in Login and SASPassword objects that are stored in the SAS Metadata Server. For more information, see "ChangePassPhrase Method" on page 100.

The GetRepositories method OMI_ALL (1) flag is enhanced to return additional attributes. When the OMI_ALL (1) flag is set, the GetRepositories method returns values for the Engine=, MetadataCreated=, MetadataUpdated=, and Options= attributes for each repository, in addition to values for the Access=, CurrentAccess=, Path=, PauseState=, RepositoryFormat=, and RepositoryType= attributes. For more information, see "GetRepositories Method" on page 122.

PART 1

Concepts

1

Introduction

About This Book

This book provides reference and usage information about the SAS 9.4 Open Metadata Interface. It also provides usage information about the SAS 9.4 Java Metadata Interface.

The SAS Open Metadata Interface is the application programming interface (API) underlying the SAS Open Metadata Architecture. The SAS Open Metadata Architecture is a client/server architecture that uses XML as its transport language. The SAS Open Metadata Interface provides the basic server interfaces for connecting to the SAS Metadata Server, creating and accessing metadata on the server, securing metadata on the server, and managing the server.

The SAS Java Metadata Interface is a Java API that provides an object interface to the metadata access functionality that is available through the SAS Open Metadata Interface. It enables developers to create and access metadata on the SAS Metadata Server without having to understand how to format XML requests.

Using these APIs with the SAS Metadata Model, which is described in *SAS Metadata Model: Reference*, developers can produce SAS Open Metadata Architecture clients that create and manage metadata in metadata repositories, secure the metadata, and manage the SAS Metadata Server.

We encourage the use of the server interfaces in a Java or a SAS environment. Instead of using SAS Open Metadata Interface metadata access methods directly, we encourage Java developers to use the SAS Java Metadata Interface to produce clients that create, read, and update metadata on the SAS Metadata Server. Direct use of the server interfaces should be reserved for tasks that cannot be performed with the SAS Java Metadata Interface.

SAS provides SAS language interfaces to metadata, such as PROC METADATA and SAS metadata DATA step functions, to enable SAS programmers to submit SAS Open Metadata Interface method requests either directly or indirectly within SAS.

This book is organized in four parts:

- Part 1, Concepts, provides an overview of the SAS Open Metadata Architecture and the SAS Open Metadata Interface server interfaces. It provides information that is needed by all clients to connect to and communicate with the SAS Metadata Server.
- Part 2, SAS Java Metadata Interface, describes how to use the SAS Java Metadata Interface to produce clients that create, read, and update metadata. Reference information about SAS Java Metadata Interface methods is provided as Java class documentation. You can view a Web-enabled version of the Java class documentation at support.sas.com/94api.
- Part 3, Server Interfaces, contains reference information about SAS Open Metadata Interface server interfaces. There are four server interfaces:
  - IOMI — metadata access interface
  - ISecurity — metadata authorization interface
  - ISecurityAdmin — security administration interface
  - IServer — server control interface

  IOMI information is provided for PROC METADATA users. PROC METADATA enables users to submit IOMI method calls that are formatted for the DoRequest method through its interface.
- Part 4, IOMI Server Interface Usage, contains IOMI usage information that is helpful to all clients issuing metadata access method calls, whether clients are using the SAS Java Metadata Interface, IOMI methods directly, or one of the SAS language interfaces to metadata.

Installation Requirements

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

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

- SAS Versioned Jar Repository (VJR)
- SAS Integration Technologies software appropriate for the client
- software for the intended programming environment

See “JRE and JAR Requirements” on page 28 for information about SAS Java Metadata Interface requirements.

Both the SAS Open Metadata Interface server interfaces and SAS Java Metadata Interface are contained in the VJR. The VJR is installed when the SAS Intelligence Platform Object Framework or SAS Management Console is installed. For easy

access to the VJR, we recommend the SAS AppDev Studio development environment. For more information about the VJR, see the *SAS AppDev StudioUser's Guide* at support.sas.com/rnd/appdev.

SAS language interfaces to metadata such as PROC METADATA and SAS metadata DATA step functions simply need access to Base SAS 9.4 software.

Prerequisites

- You must have access to a properly configured SAS Metadata Server to create and access metadata. A properly configured metadata server has a foundation repository that contains standard SAS metadata. The SAS Deployment Wizard installs and configures a metadata server for you. An existing SAS 9.2 Metadata Server or SAS 9.3 Metadata Server can be migrated to the SAS 9.4 environment by using the SAS Migration Utility and SAS Deployment Wizard. For more information, see the *SAS Intelligence Platform: Migration Guide*.
- To connect to the SAS Metadata Server, you must be able to authenticate to the server. To create and update metadata, you must have proper authorization to metadata repositories. For more information, see the *SAS Intelligence Platform: Security Administration Guide*.

Audience

This book provides information for developers who are producing or maintaining clients that access metadata, secure metadata, or manage the SAS Metadata Server.

It is the primary source of information for developers who are producing SAS Java Metadata Interface and SAS Open Metadata Interface clients.

It is a secondary source of information for users of the SAS language interfaces to metadata. The SAS language interfaces to metadata are described in *SAS Language Interfaces to Metadata*. Users of the SAS language interfaces to metadata need information from this book to be able to format the XML method calls that can be submitted with PROC METADATA.

What Is the SAS Open Metadata Architecture?

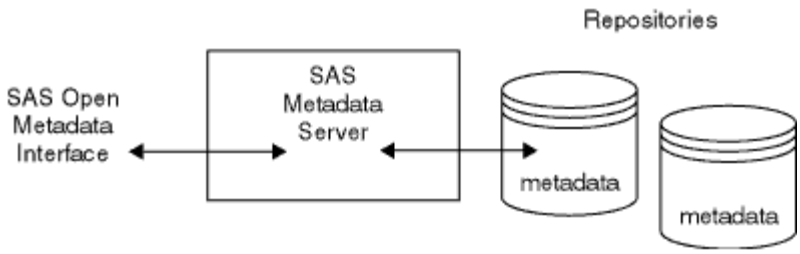
The SAS Open Metadata Architecture is a general-purpose metadata management facility that provides common metadata services to SAS applications. Using the metadata architecture, separate SAS applications can exchange metadata, which makes it easier for these applications to work together. The metadata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

The metadata architecture includes a metadata model, an API, and a metadata server.

- The metadata model, called the SAS Metadata Model, provides classes and objects that define repositories, the SAS Repository Manager, and different types of application metadata.

  The SAS Metadata Server uses information stored in repository objects to access the metadata repositories. It uses the SAS Repository Manager to manage the metadata repositories. The application metadata types are used in various combinations by clients to create metadata that describes application data or entities that are used by an application. The metadata model defines valid relationships between metadata types, uses the inheritance of attributes and associations to affect common behaviors, and uses subclassing to extend behaviors.
- The SAS Open Metadata Interface provides methods for reading and writing metadata objects in repositories. It also provides methods for administering repositories and the SAS Metadata Server, for defining and administering access controls on application metadata objects and repositories, and for getting authorizations based on the metadata access controls.
- The SAS Metadata Server is a server that surfaces metadata from one or more repositories through 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 and enables you to use industry-standard languages, programming tools, and communication protocols to develop clients that access Base SAS features on IOM servers. The purpose of the SAS Metadata Server is to provide a central, shared location for accessing metadata.

Figure 1.1 *SAS Open Metadata Architecture*



What Can I Do with the SAS Open Metadata Interface?

The SAS Open Metadata Interface enables clients to read and write the metadata of applications that comply with the metadata architecture. It also supports the development of clients to maintain repositories and to control the SAS Metadata Server, but these tasks are secondary. Mostly, clients use the SAS Open Metadata Interface to read or write the metadata of applications. For example, a client might use the SAS Open Metadata Interface to perform the following tasks:

- Store LIBNAME information that is needed to access data stores so that the information is available centrally and can be maintained independently of the client.
- Store details about data sources, such as SAS and database table and column names, formats, data types, and information about responsible parties, so that the information can be centrally searched and used to locate tables that meet specified criteria.
- Return a list of available SAS application servers and use their definitions to maintain their configuration and manage the servers. For example, the definitions could be used to start, stop, pause, and resume the servers.
- Define access controls on resources and request authorization decisions from the SAS Open Metadata Architecture authorization facility.

Authentication

The SAS Metadata Server supports a variety of authentication providers to determine who can access the SAS Metadata Server. It also defines privileged users. Only a user who has been granted unrestricted user status on the SAS Metadata Server has unrestricted access to metadata on the SAS Metadata Server. Only a user who has been granted either unrestricted user status or administrative user status on the SAS Metadata Server can create and delete repositories, modify a repository's registrations, change the state of a repository, and register users. For more information about metadata server authentication and privileged users, see the *SAS Intelligence Platform: Security Administration Guide*.

Authorization Facility

The SAS Metadata Server uses an authorization facility to control access to metadata repositories and to specific metadata in the metadata repositories. Authorization processes are insulated from metadata-related processes in the SAS Metadata Server. 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 uses the authorization facility to make queries about ReadMetadata and WriteMetadata permissions on metadata and enforces the decisions that are returned by the authorization facility. It is not necessary for SAS Open Metadata Interface clients to enforce authorization decisions regarding the ReadMetadata and WriteMetadata permissions.

SAS Open Metadata Interface clients can use the authorization facility to request authorization decisions on other types of access (for example, to request authorization decisions on data that is represented by SAS metadata). For example, other SAS IOM servers define and enforce Read, Write, Create, and Delete permissions on data that is represented by metadata. Applications that use the authorization facility to request authorization decisions on application-defined actions and objects must enforce the authorization decisions themselves.

The authorization facility's interface consists of a set of methods that are available in the ISecurity server interface. For more information, see Chapter 8, “Authorization (ISecurity Interface),” on page 143. Security administration methods are available in the ISecurityAdmin server interface. For more information, see Chapter 9, “Security Administration (ISecurityAdmin Interface),” on page 195.

For information about the types of access controls supported by the authorization facility and how the authorization facility makes authorization decisions, see the *SAS Intelligence Platform: Security Administration Guide*.

2

Client Requirements

Types of SAS Open Metadata Interface Clients

A SAS Open Metadata Interface client is a program that communicates with the SAS Metadata Server. The SAS Open Metadata Interface provides methods to perform the following tasks on the SAS Metadata Server:

- Create, read, and update repository objects.
- Create, read, and update application metadata objects.
- Control access to the SAS Metadata Server.
- Define access controls on application resources and repositories, request authorizations based on access controls, and manage access controls.
- Define and manage internal user accounts.
- Back up and recover the SAS Metadata Server.

Most clients create, read, and update application metadata.

Clients use repository objects to register repositories in the SAS Repository Manager, to modify a repository's registered access mode, or to get information about repository availability.

A client that controls access to the SAS Metadata Server does so to interrupt client activity so that maintenance tasks can be performed. Examples of maintenance tasks are recovering memory, running metadata analysis and repair tools, or changing certain server configuration and invocation options while the server is offline. For SAS 9.4, which supports clustering of metadata servers to provide redundancy and high availability, options are available to execute server control requests on specific servers or uniformly on all servers in the cluster.

The SAS authorization facility supports resource-based authorization and role-based authorization.

A client that defines resource-based authorization enables administrators to define and manage access controls on the metadata that describes the resources. Access controls can be defined directly on the metadata that describes a resource, or they can be defined in an access control template (ACT) that is associated with many resources. A client that manages access controls enables administrators to list identities that have permissions on a resource. Administrators can also list permissions that are defined directly on a resource, list permissions that are defined in an ACT, and apply and remove ACTs from a resource. Administrators can create an ACT, modify the attributes of an ACT, and destroy an ACT.

A client that requests authorizations based on resource-authorization settings queries the SAS Open Metadata Architecture authorization facility to determine whether the specified user has appropriate permission to a requested resource based on active access controls. Then, depending on the decision, the SAS authorization facility either enforces the decision or allows the SAS Metadata Server to enforce the decision. The SAS Metadata Server enforces ReadMetadata and WriteMetadata permissions to a resource. A client that wants to enforce other permissions on a resource must do so itself. For information about the default access controls supported by the SAS authorization facility, and how the SAS authorization facility works, see the *SAS Intelligence Platform: Security Administration Guide*.

A client that defines role-based authorization identifies application actions that will be controlled as metadata. Administrators can assign identities to the roles. The GetApplicationActionsAuthorizations method enables clients to request decisions based on role membership.

A client that creates and manages internal user accounts creates internal logins, and modifies their authentication settings for the task.

Appropriate identity, permission, resource, ApplicationAction, and Role objects must be defined in the SAS Metadata Server for authorizations to be meaningful. For detailed information about the security features that are available through the SAS Open Metadata Architecture authorization facility, see the *SAS Intelligence Platform: Security Administration Guide*.

For information about methods that can be used to back up and recover the SAS Metadata Server, see “Backing Up and Recovering the SAS Metadata Server” on page 17.

Important Terms

To create a metadata client, you must be familiar with the following terms:

metadata type
: specifies a SAS Metadata Model metadata type that models the metadata for a resource. 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. The SAS Metadata Model defines approximately 170 metadata types.

namespace
: specifies a group of related metadata types. A namespace is 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. The REPOS namespace contains metadata types that describe repositories.

metadata object
: specifies an instance of a metadata type.

metadata type property
: is a term that collectively refers to the attributes and associations that are defined for a metadata type in the SAS Metadata Model. An attribute describes a characteristic of the metadata type. An association describes a relationship between an object of this metadata type and an object of another metadata type.

Creating Repositories

Before you can create application metadata in a SAS Metadata Repository, you must create metadata that defines at least one SAS Metadata Repository. The SAS Open Metadata Interface can be used to create a SAS Metadata Repository, but this is not the recommended way. If you perform a SAS planned installation to set up your SAS Metadata Server, the SAS Deployment Wizard creates the first repository — a foundation repository — for you. If additional repositories are needed, we recommend that you create them with SAS Management Console because it creates default metadata in the repositories for you. For more information about defining and managing SAS metadata repositories, see the *SAS Intelligence Platform: System Administration Guide*.

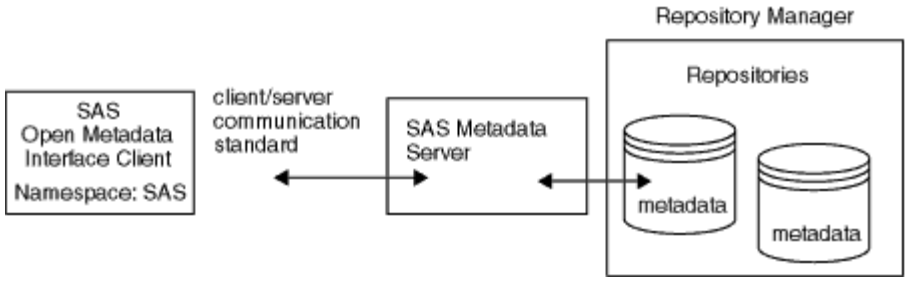
Creating and Accessing Metadata That Describes Application Elements

A SAS Open Metadata Interface client that accesses application metadata has the following characteristics:

- The client connects to the SAS Metadata Server with a communication standard that is appropriate for the client and the IOM-based server.
- The client specifies the SAS namespace to access the metadata types for application elements, such as servers, tables and columns.

- The client issues SAS Open Metadata Interface method calls to create or access instances of the metadata types that are stored in metadata repositories.

Figure 2.1 *Accessing Metadata Defined in the SAS Namespace*



For server connection information, see “Connecting to the SAS Metadata Server” on page 12.

For a description of the metadata types in the SAS namespace, see “Alphabetical Listing of SAS Namespace Types” in the *SAS Metadata Model: Reference*.

The SAS Open Metadata Interface provides the IOMI server interface for reading and writing metadata objects. For more information about IOMI, see Chapter 7, “Metadata Access (IOMI Interface),” on page 71.

We recommend that clients use the SAS Java Metadata Interface to read and write metadata instead of using IOMI methods directly. For reference information about the SAS Java Metadata Interface, see the documentation at support.sas.com/94api. For usage information, see Chapter 6, “Using the SAS Java Metadata Interface,” on page 41.

Beginning in SAS 9.3, a SAS type dictionary affects the SAS Metadata Model metadata types that clients should select to represent common and shared application entities. Objects need to store specific attribute values to ensure that the dictionary is used in metadata queries. For more information about the SAS type dictionary, see Chapter 3, “Using Interfaces That Read and Write Metadata in SAS 9.4,” on page 19.

Connecting to the SAS Metadata Server

Available Interfaces

The SAS Metadata Server is an object server; it uses the Integrated Object Model (IOM) provided by SAS Integration Technologies to communicate with clients. SAS Integration Technologies provides interfaces that enable clients to connect to the SAS Metadata Server generically as an IOM server. When you use these interfaces, you must be familiar with the interfaces and classes that define the SAS Metadata Server and the SAS Open Metadata Interface server interfaces. In addition, you must know how to read and write an XML document to use the metadata access functionality of the IOMI server interface.

As an alternative to the SAS Integration Technologies interfaces, SAS provides the SAS Java Metadata Interface. The SAS Java Metadata Interface hides the details of IOM servers and the steps of how to create a connection to a SAS Metadata Server from the client. It provides a Java object interface to the metadata access functionality of the SAS Open Metadata Interface. This object interface defines an interface for each SAS Metadata Model metadata type. These interfaces enable clients to use getter and setter methods to read and write metadata attributes and associations, instead of requiring clients to submit XML metadata property strings that define values, like the IOMI methods do. In addition, these interfaces provide methods for connecting to the SAS Metadata Server with the ISecurity, ISecurityAdmin, and IServer server interfaces, so that clients do not need to know the details of their implementation.

Because of its ease of use, the SAS Java Metadata Interface is recommended over the SAS Integration Technologies interfaces, both for performing metadata access tasks and for connecting to the SAS Metadata Server with the non-metadata server interfaces.

Connecting with SAS Integration Technologies Interfaces

SAS Integration Technologies provides the SAS Object Manager for Windows client development and the Java Connection Factory for Java client development. For more information about the interfaces, see the *SAS Integration Technologies: Windows Client Developer's Guide* and the *SAS Integration Technologies: Java Client Developer's Guide*.

Connecting with the SAS Java Metadata Interface

The SAS Java Metadata Interface and SAS Open Metadata Interface are contained in JAR files in the SAS VJR. For information about the Java Runtime Environment (JRE) and JAR files required by the SAS Java Metadata Interface, see “JRE and JAR Requirements” on page 28. The server interfaces are provided in the sas.oma.omi.jar file.

A Java client accesses the APIs by importing the appropriate packages, instantiating an object factory, and connecting to the SAS Metadata Server with a handle to the interface that is appropriate for the task that it wants to perform. A SAS Java Metadata Interface client that will perform metadata access tasks imports the com.sas.metadata.remote package. A client that accesses the ISecurity, ISecurityAdmin, or IServer server interface imports the com.sas.metadata.remote package and appropriate interfaces from the com.sas.meta.SASOMI package. For information about the specific com.sas.meta.SASOMI interfaces that are required, see the server interface documentation.

The SAS Java Metadata Interface provides the MdFactory interface to instantiate an object factory for the SAS Metadata Server, and provides the MdOMRConnection interface to connect to the SAS Metadata Server. The MdOMRConnection interface includes the following methods to enable you to connect to the SAS Metadata Server:

- makeOMRConnection—connects to the SAS Metadata Server with the SAS Java Metadata Interface. A client uses the SAS Java Metadata Interface to read and write metadata.

- makeISecurityConnection—connects to the SAS Metadata Server with the ISecurity server interface. ISecurity contains metadata authorization methods. A client uses the ISecurity server interface to request user-defined authorization decisions on access controls that are stored as metadata.
- makeISecurityAdminConnection—connects to the SAS Metadata Server with the ISecurityAdmin server interface. ISecurityAdmin contains security administration methods. A client uses the ISecurityAdmin server interface to administer access controls that are defined directly on resources and to manage ACTs.
- makeIServerConnection—connects to the SAS Metadata Server with the IServer server interface. IServer contains server control methods. A client uses the IServer server interface to pause and resume, refresh, get the status of, and stop the SAS Metadata Server.

For examples of the statements that are required to establish a connection to the SAS Metadata Server with the SAS Java Metadata Interface, see "Sample Program" on page 55. The sample program is a metadata access client.

For more information about connecting with the non-metadata server interfaces, see Chapter 8, "Authorization (ISecurity Interface)," on page 143, Chapter 9, "Security Administration (ISecurityAdmin Interface)," on page 195, and Chapter 10, "Server Control (IServer Interface)," on page 237.

Server Connection Properties

A client must specify the following server connection properties to connect to a SAS Metadata Server. Optional properties are described in the SAS Integration Technologies documentation.

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

port=*number*
: The TCP port to which the SAS Metadata Server listens for requests, In addition, clients will use this port to connect to the SAS Metadata Server. The *number* value must be a unique number from 0 to 65,535. The default port number is 8561.

user name
: A valid user name on the host machine, or a SAS internal account. For information about internal authentication, see *SAS Intelligence Platform: Security Administration Guide*.

password
: the password for the user name.

Communicating with the SAS Metadata Server

The SAS Open Metadata Interface supports two ways to submit requests to the SAS Metadata Server—the standard interface and the DoRequest interface.

Standard Interface

A request is submitted by declaring object variables that represent the method's parameters in the client, and then referencing the object variables in the method request. In this documentation, we refer to this way to submit requests as the "standard interface." The standard interface is supported for the SAS Java Metadata Interface and all of the SAS Open Metadata Interface server interfaces.

When you submit a request using the standard interface, the SAS Metadata Server does not require you to use the published parameter names for the object variables. However, if you use different parameter names, the names in the object variable declarations must also be used to represent the method's parameters in the method request. For example, consider the GetMetadata method.

In the standard interface, a GetMetadata request is submitted as follows:

```
inMetadata= "<PhysicalTable Id="A5345678.A5000001" Name="" Desc="">
                            <Columns/>
                           </PhysicalTable>";
ns="SAS";
flags=0;
options="";

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

You do not have to use the parameter names INMETADATA, OUTMETADATA, NS, FLAGS, and OPTIONS in the object variable declarations. However, the same names that you use in the object variable declarations must be used in the method syntax. The names must be specified in the order in which they are presented in the method's documentation.

DoRequest Interface

The SAS Open Metadata Interface IOMI server interface and the Status method from the IServer server interface can be submitted to the SAS Metadata Server using the DoRequest interface. The DoRequest interface is based on the IOMI DoRequest method. The DoRequest method is a messaging method whose sole purpose is to submit another method to the SAS Metadata Server. A client declares object variables for the DoRequest method's parameters in the client. Then, the client submits another method in the DoRequest method's INMETADATA parameter. This other method's parameters are formatted in an XML string. For example, consider the GetMetadata method from the previous example, reformatted for the DoRequest method:

```
outMetadata=" ";
inMetadata="<GetMetadata>
 <Metadata>
   <PhysicalTable Id="A5345678.A5000001" Name="" Desc="">
     <Columns/>
   </PhysicalTable>
 </Metadata>
 <NS>SAS</NS>
 <Flags>0</Flags>
```

```
 <Options/>
</GetMetadata>";

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

The DoRequest interface provides a standard way for a client to submit method requests to the SAS Metadata Server. Instead of the client parsing the submitted method's parameters, the SAS Metadata Server parses them. The format of the XML string is described in "DoRequest Method" on page 107. The IOMI reference documentation includes examples of how to format each method for the DoRequest interface.

Because we recommend the SAS Java Metadata Interface to read and write metadata (instead of using the IOMI server interface directly), a Java client would not use the DoRequest interface. However, PROC METADATA accepts IOMI methods that are formatted for the DoRequest method's INMETADATA parameter as its input. For an example of how to submit, from PROC METADATA, an XML string that is formatted for the DoRequest method, see PROC METADATA documentation in *SAS Language Interfaces to Metadata*.

When creating an XML string for the DoRequest method or PROC METADATA, you must use published parameter names in the XML elements to represent the method parameters (<NS>, <FLAGS>, <OPTIONS>), with one exception. The submitted method's INMETADATA parameter should be represented by a <METADATA> element, as shown in the example. The method parameters do not need to be specified in the order in which they are presented in the syntax.

Controlling the SAS Metadata Server

A SAS Metadata Server must be running before any client can access metadata repositories. At many sites, an administrator starts the SAS Metadata Server, and then SAS Open Metadata Interface clients simply connect to that server. However, there are times when the administrator might want to refresh the SAS Metadata Server to change configuration or invocation options, or pause the server to temporarily downgrade the SAS Metadata Server's state. For example, the administrator might want to downgrade the server from an ONLINE state to an ADMINISTRATION state, so that only administrative users can access the SAS Metadata Server. Or, the administrator might want to take the server OFFLINE, which halts client activity while maintaining client connections, or stop the SAS Metadata Server, which halts client activity and terminates client connections.

The SAS Open Metadata Interface provides the IServer server interface for controlling the SAS Metadata Server. IServer includes Pause, Refresh, Resume, Status, and Stop methods. For more information about IServer methods, see Chapter 10, "Server Control (IServer Interface)," on page 237.

A user must have administrative user status on the SAS Metadata Server to issue all IServer methods, except the Status method. For more information about the administrative user status, see *SAS Intelligence Platform: Security Administration Guide*.

Administrative users are encouraged to use SAS Management Console to control the SAS Metadata Server. For information about controlling the server using SAS Management Console, see the *SAS Intelligence Platform: System Administration Guide*.

Backing Up and Recovering the SAS Metadata Server

It is important to make regular backups of your SAS metadata repositories. The SAS Metadata Server locks repository files, so you cannot use operating system commands to back up a running SAS Metadata Server unless you stop the server or change it to an OFFLINE state before you back up the files.

The SAS Metadata Server includes a server-based facility that can be used to perform unassisted, scheduled SAS Metadata Server backups. Because the facility is run by the server (in a separate thread), the following is true:

- Backups can run without interrupting the regular operation of the SAS Metadata Server.
- There is no need to obtain operating system access to SAS Metadata Server directories or to configure a special account for backup processing.
- There is no need to obtain appropriate authorization to metadata to back up the SAS Metadata Server.

The facility can be used to perform ad hoc backups, restores, and roll-forward recovery of the SAS Metadata Server from the metadata server journal. A client must have administrative access to the SAS Metadata Server to perform these tasks.

This facility replaces the %OMABAKUP macro that was provided to back up and restore the SAS Metadata Server in SAS 9.1 and SAS 9.2.

The SAS Deployment Wizard uses this facility to configure a default schedule of unassisted metadata server backups for all new and migrated SAS systems during installation. The SAS Management Console Metadata Manager provides a graphical user interface to enable customers to monitor and manipulate backups, modify the default backup configuration and backup schedule, perform ad hoc backups, and perform metadata server recovery. A MetadataServer script in the **`SASMeta/MetadataServer`** directory provides a batch interface for the functionality available in SAS Management Console.

Lower-level access can be obtained by using metadata procedures, such as PROC METAOPERATE and PROC METADATA, or by issuing IServer Refresh and Status method requests. However, using metadata procedures and issuing method requests are not necessary. Directly using the IServer Refresh or Status method is discouraged. The recommended interface for manipulating, administering, and monitoring server backups is SAS Management Console.

For information about manipulating, administering, and monitoring server backups with SAS Management Console, including best practices for incorporating server backups into the overall backup plan for the SAS Intelligence Platform, see the *SAS Intelligence Platform: System Administration Guide*. The MetadataServer script is also documented in the *SAS Intelligence Platform: System Administration Guide*.

For information about administering server backups with PROC METAOPERATE and monitoring server backups with PROC METADATA, see the *SAS Language Interfaces to Metadata*. IServer methods are described in Chapter 10, “Server Control (IServer Interface),” on page 237.

3

Using Interfaces That Read and Write Metadata in SAS 9.4

Overview of Using Interfaces That Read and Write Metadata

SAS provides the IOMI server interface and SAS Java Metadata Interface for reading and writing metadata. The IOMI server interface and SAS Java Metadata Interface create and manipulate specific instances of SAS Metadata Model metadata types.

Most resources and information assets that are managed in a SAS Metadata Repository are described by a set of SAS Metadata Model metadata types, rather than by just one metadata type. For example, a SAS table is described by the PhysicalTable, Column, Index, UniqueKey, and ForeignKey metadata types. The set of metadata types that describe a resource or information asset is referred to as the object's *logical metadata definition*. A logical metadata definition typically includes a primary metadata type and several associated secondary metadata types.

When using the SAS Open Metadata Interface or SAS Java Metadata Interface to create metadata, it is your responsibility to define an object's logical metadata definition. You do this by examining the SAS Metadata Model, and then selecting the metadata types that best describe the components of the resource or information asset that you are defining. Then, you make one or more requests to the SAS Metadata Server that define the objects and associations between the objects to create the logical metadata definition.

A SAS Open Metadata Interface and SAS Java Metadata Interface query method gets a specific SAS Metadata Model object instance, or it gets all object instances of

a specified SAS Metadata Model metadata type, and it gets specified attributes and associations. In the past, if you wanted to get an entity's full logical metadata definition in a query method request, you had to create and submit templates that identified the association names and secondary metadata types, which compose the logical metadata definition, in the request.

The SAS Metadata Model is structured to distinguish PrimaryType subtypes from SecondaryType subtypes to facilitate the metadata type selection process for creating logical metadata definitions.

Beginning in SAS 9.2, SAS took steps to improve consistency in the look and feel of SAS applications through the introduction of the SAS Folders tree and a SAS type dictionary. The SAS Folders tree is a feature of SAS Management Console, SAS Data Integration Studio, SAS OLAP Cube Studio, and other SAS applications.

In order for an object to appear in the SAS Folder tree, it must be of an object type that is registered in the dictionary. For object types that are persisted in metadata, an aspect of the dictionary is that it standardizes the primary metadata type and association names used to retrieve their logical metadata definition, and makes it easier to retrieve a logical metadata definition.

This section describes the dictionary and how it affects Read and Write operations with the interfaces described in this book.

What Is the SAS Type Dictionary?

The SAS type dictionary is a set of object type definitions that describe the common and shared objects that are used by SAS applications. A type definition is metadata that is represented in the SAS Metadata Repository by an object of the TypeDefinition metadata type. To determine the object types that are in the SAS type dictionary, view the dictionary in the SAS Folders tree. The type definitions are in the `/System/Types` folder.

The type definition contains information that is necessary to display and operate on instances of that object type in a SAS application. For example, instead of describing the content of an object (like a table's columns, indexes, and keys), the type definition defines the icon that is used to represent the object type in a SAS application, the metadata template that is used to retrieve the object's logical metadata definition, the type of object that should contain the object in the Folders tree (folder or other object), and the classes used to provide copy, paste, move, delete, import, and export functionality for the object.

Note: Not all object types support import and export. And, not all object types appear in the Folders tree.

Content of a Type Definition

The following attributes of a type definition are important when creating and querying metadata. This information is available on the **Advanced** tab of a type definition's Properties window. In addition, you can get this information by querying the appropriate TypeDefinition metadata object.

MetadataType
: specifies the primary SAS Metadata Model metadata type in the object type's logical metadata definition. This is always a PrimaryType subtype in the SAS Metadata Model.

TypeName
: specifies a name that uniquely identifies the type definition in the SAS type dictionary.

  Many type definitions use the same SAS Metadata Model metadata type as the primary object in their logical metadata definition. For example, the type definitions for both the SASReport and InformationMap object types specify Transformation as their metadata type. To ensure that the appropriate type definition is used to retrieve an object type's logical metadata definition, SAS wizards and procedures that create metadata store the TypeName value that is appropriate for each object type in the primary object's PublicType attribute. When listing objects, the PublicType value is used with the MetadataType to identify the object type.

ContainerType
: specifies the object type that contains the object. This is usually Folder, but it can be another object type from the SAS type dictionary. For example, a Column object is contained by a Table object.

ContainerAssociation
: specifies the association name that links the primary object in a logical metadata definition to its container. Object types that are contained by a folder have a Trees association to the folder. (A folder is represented by an object of the Tree metadata type in the SAS Metadata Model.) A Column object has a Table association to the table that contains it.

DefinitionVersion
: specifies a double value in the form `MMMmmbbbb.0` that indicates the type definition's current usage version. `M`=major version, `m`=minor version, and `b`=build.

SupportedObjectVersionMin
: specifies a double value in the form `MMMmmbbbb.0` that indicates the minimum usage version supported for this type definition.

SupportedObjectVersionMax
: specifies a double value in the form `MMMmmbbbb.0` that indicates the maximum usage version supported for this type definition.

Clients can specify a type definition version for an object by storing a version value in the primary object's UsageVersion attribute. The DefinitionVersion, SupportedObjectVersionMin, and SupportedObjectVersionMax attributes indicate

the range of versions that are available for a type definition. For more information about the TypeDefinition attributes, see SAS Metadata Model: Reference.

Requirements for Using the Type Dictionary

Creating Metadata

SAS provides wizards and procedures to create metadata for the resources and information assets that are most commonly used and shared among SAS applications. We recommend that you use these wizards and procedures to create metadata, instead of coding metadata definitions directly.

If a wizard or procedure does not exist for the metadata that you want to create, and you have to create a logical metadata definition, the definition must meet the following requirements:

- Use the metadata type indicated in the appropriate type definition's MetadataType attribute for its primary object.
- The primary object should specify the following:
  - the type definition's TypeName value in the PublicType attribute.
  - a valid usage version value in the UsageVersion attribute. Most objects are versioned as 1000000.
  - an association to a valid container object, if it is specified in the type definition.
  - associations to appropriate secondary objects.

A goal of the SAS type dictionary is to save customers from having to know the details of an object type's logical metadata definition. As a result, the type definition internalizes information that is necessary to retrieve the logical metadata definition. (For example, the metadata template is not shown in a type definition's Properties window.) To determine the association names and secondary metadata types used in an object type's logical metadata definition, issue a GetMetadata request on an existing object of the object type in which you are interested. Set the OMI_FULL_OBJECT (2) flag. OMI_FULL_OBJECT is a new flag that instructs the SAS Metadata Server to use the metadata template from an object's type definition to process the request. The metadata server will return the full logical metadata definition.

A logical metadata definition that is created with the SAS Open Metadata Interface or SAS Java Metadata Interface is considered a custom logical metadata definition. SAS does not guarantee that custom logical metadata definitions will take advantage of the full functionality of SAS. To ensure that full functionality is available, use SAS wizards and procedures to create metadata.

SAS does not support custom type definitions.

Querying Metadata

A GetMetadata request retrieves specified attributes and associations of a SAS Metadata Model metadata object. If you want to get an object's full logical metadata definition, specify the appropriate PrimaryType subtype in the INMETADATA parameter, and set the OMI_FULL_OBJECT (2) flag. The SAS Metadata Server reads the value in the object's PublicType attribute to fulfill the request. If the object does not store a value in the PublicType attribute, then GetMetadata returns specified information for the SAS Metadata Model metadata object only. For more information about the OMI_FULL_OBJECT flag, see "Using the OMI_FULL_OBJECT Flag" on page 334.

To list all instances of an object type that is in the type dictionary with a GetMetadataObjects request, perform the following steps.

1. Specify the MetadataType value from the type definition in the TYPE parameter.
2. Set the OMI_XMLSELECT (128) flag.
3. Submit an <XMLSELECT> element that specifies the object's TypeName value in the OPTIONS parameter, as follows:

   ```
   <XMLSELECT search="@PublicType='TypeName'"/>
   ```

   Using the TypeName value ensures that the correct type definition is used to process the GetMetadataObjects request.

For more information about querying objects with the SAS Java Metadata Interface, see "Getting and Updating Existing Objects" on page 50.

For more information about querying objects with the SAS Open Metadata Interface, see Chapter 14, "Getting the Properties of a Specified Metadata Object," on page 315 and Chapter 16, "Getting All Metadata of a Specified Metadata Type," on page 355.

Deleting Objects

A DeleteMetadata request that specifies a PrimaryType subtype always uses the type dictionary to process the request unless you specify a user-defined template.

For more information about deleting objects with the SAS Java Metadata Interface, see "Deleting Objects" on page 53.

For more information about deleting objects with the SAS Open Metadata Interface, see Chapter 19, "Deleting Metadata Objects," on page 395.

Benefits of the SAS Type Dictionary

Here are the advantages of using logical metadata definitions that conform to the SAS type dictionary:

- The objects can be shared with other SAS applications and have consistent functionality.
- The metadata objects are displayed in the SAS Folders tree, and they can be operated on using the functionality available through the SAS Folders tree. This functionality includes the following:
  - update, copy, paste, move, and delete actions on all object types
  - import, export, and impact analysis actions on some object types
  - analyze and repair metadata and upgrade metadata actions on all object types, as appropriate

  For more information about the SAS Folders tree, see the *SAS Intelligence Platform: System Administration Guide*.
- The container gives context to the object in the SAS Metadata Repository.

PART 2

SAS Java Metadata Interface

4

Understanding the SAS Java Metadata Interface

About This Section

This section describes how to create, read, and update metadata in metadata repositories with the SAS Java Metadata Interface. Reference information about the SAS Java Metadata Interface is provided as Java class documentation. You can view a Web-enabled version of the Java class documentation at support.sas.com/94api.

The SAS Java Metadata Interface is a Java API that provides an object interface to the metadata access functionality that is available through the SAS Open Metadata Interface IOMI server interface. In addition, it provides methods for connecting to the SAS Metadata Server with the non-metadata SAS Open Metadata Interface server interfaces (IServer, ISecurity, and ISecurityAdmin).

SAS Java Metadata Interface Overview

The SAS Java Metadata Interface provides a way to access metadata repositories through the use of client Java objects that represent server metadata. This enables users to perform metadata access tasks without having to have a deep understanding of the format of each XML method.

The API contains functionality used for the following:

- connecting to the SAS Metadata Server
- creating, reading, and writing Java object instances that represent SAS Metadata Model objects on the client, and propagating additions and changes to the SAS Metadata Server

The API is contained in the com.sas.metadata.remote package. The com.sas.metadata.remote package is typically used with the com.sas.services.information package included with SAS Foundation Services software. The com.sas.services.information package provides a generic interface for interacting with heterogeneous data repositories from client applications. Heterogeneous data repositories include SAS metadata repositories, Lightweight Directory Access Protocol (LDAP) repositories, and WebDAV repositories. Using the com.sas.services.information package, a client can create objects that provide a layer of abstraction. As a result, users do not need to know all of the details of the SAS Metadata Model. These objects typically wrap the SAS Metadata Model objects created by the SAS Java Metadata Interface. The com.sas.services.information package is described in the SAS Foundation Services Java class documentation. SAS Foundation Services is a component of SAS Integration Technologies. The Java class documentation is available at support.sas.com/94api.

JRE and JAR Requirements

The current release of the Java client software requires Java 2 SDK Standard Edition, Version 1.8 (JDK 1.8.0) or higher.

The SAS Java Metadata Interface depends on several JAR files. These JAR files can be obtained from the SAS Versioned JAR Repository (VJR), which is typically located in the*`SAS-installation-directory`*`\SASVersionedJarRepository\9.4\eclipse\plugins` directory.

Here are the JAR files:

- sas.oma.joma.jar
- sas.oma.joma.rmt.jar
- sas.oma.omi.jar
- sas.svc.connection.jar
- sas.core.jar
- sas.entities.jar
- sas.security.sspi.jar

If you are using SAS AppDev Studio as your development environment, the JAR files are automatically made available in the project's build path from the project's SAS repository. If you are using another Java development environment, you need to copy the JAR files from the VJR to the project's build path, and set their location in the CLASSPATH variable.

How the SAS Java Metadata Interface Works

The SAS Java Metadata Interface consists of the following:

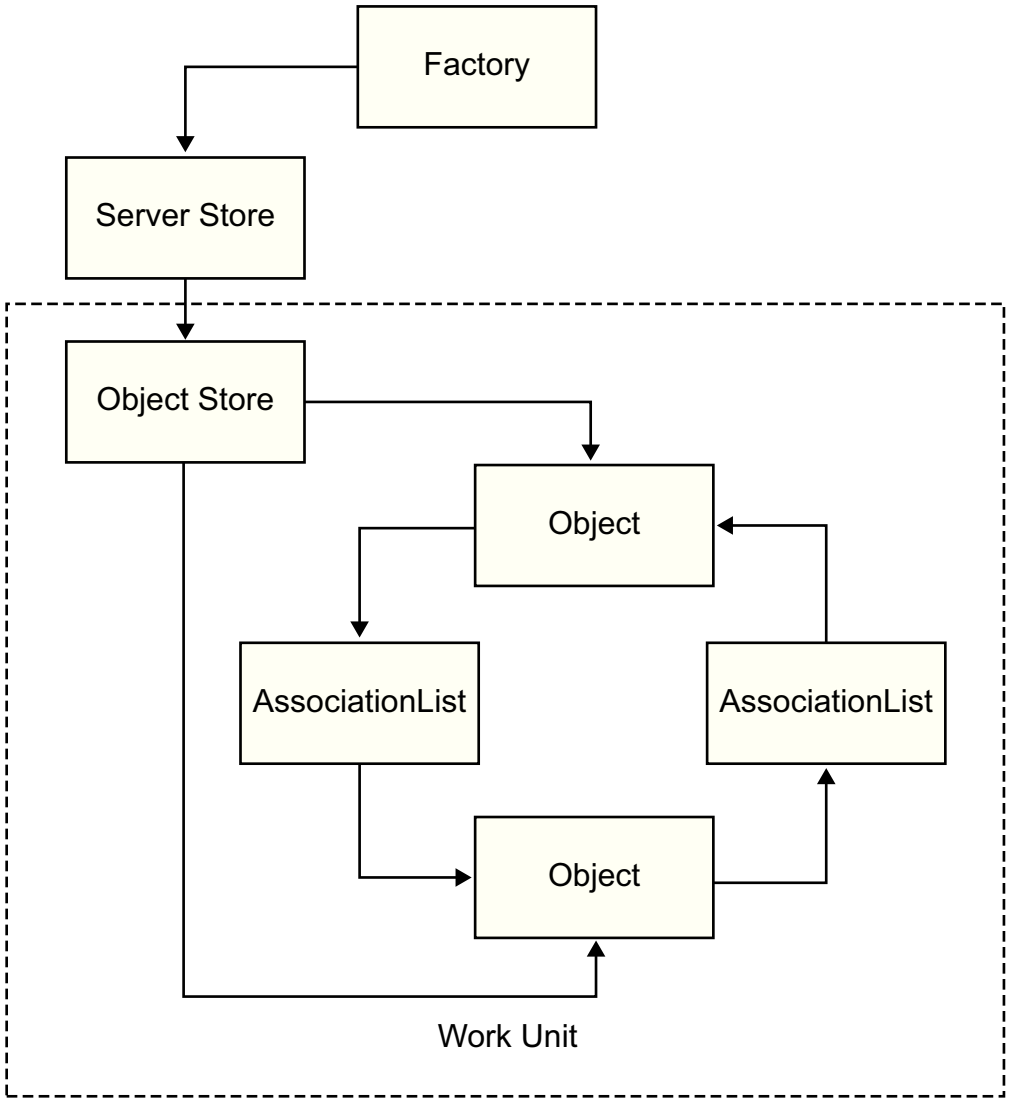
- Java interfaces that correspond to objects in the SAS Metadata Model

- an object factory for creating and controlling the life cycles of objects in the client
- object stores that serve as work unit containers for storing object instances and for grouping object instances that need to be persisted to the SAS Metadata Server as a unit

The object factory provides an environment for managing Java objects that represent SAS metadata object instances.

The object store serves as a container for Java objects that users create to add or modify metadata objects in the SAS Metadata Server. The following figure illustrates the relationship between the objects in an object store.

Figure 4.1 *Relationship between Objects in an Object Store*



A SAS Open Metadata Interface metadata object is defined by two types of properties:

- a set of attributes that describe the characteristics of the metadata object instance, including its name, description, date it was created, and any unique characteristics
- associations that describe its relationships with other metadata objects

Using the SAS Java Metadata Interface, you create a metadata object on the SAS Metadata Server, or you modify an existing metadata object's attributes, by creating a Java object representing its SAS metadata type. You then persist the new or modified Java object to the SAS Metadata Server. A metadata type refers to one of the metadata types defined in the SAS namespace of the SAS Metadata Model. Metadata objects live in the SAS Metadata Server. The Java objects in the object store act as proxies for the metadata objects in the SAS Metadata Server.

Information about associations is managed separately from information about attributes. Associations are managed by creating AssociationList objects. An AssociationList object stores information about how two metadata objects are related to each other through an association name. To determine the associations defined for a specific metadata type, see the "Alphabetical Listing of SAS Namespace Metadata Types" in the SAS Metadata Model: Reference.

In the figure, Figure 4.1 on page 29, the squares named Object represent metadata objects, and the squares named AssociationList represent the associations between the metadata objects. Every relationship in the SAS Metadata Model is a two-way association. That is, there are two sides to each relationship, and each side has a name. For example, if the metadata objects in the figure represented a PhysicalTable and a Column, the PhysicalTable object would have a Columns association to the Column object. The Column object would have a Table association to the PhysicalTable object. For more information about associations, see "Understanding Associations" in *SAS Metadata Model: Reference*.

For an overview of the interfaces used to create the factory, stores, and other objects, see "Interfaces and Classes Summary" on page 31.

For information about how to write a SAS Java Metadata Interface client that reads and writes metadata, see "Overview of Creating a SAS Java Metadata Interface Client" on page 41.

For documentation about specific classes and methods, see the SAS Java Metadata Interface at support.sas.com/93api.

5

Overview of Interfaces and Classes

Interfaces and Classes Summary

A SAS Java Metadata Interface client that reads and writes metadata objects references the following interfaces and classes from the com.sas.metadata.remote package.

Table 5.1 *com.sas.metadata.remote Interfaces and Classes for Reading and Writing Metadata*

Class or Interface Name	Description
MdFactory	The starting point for all Java clients. This interface is used to control the creation of metadata objects and object stores, as well as to maintain the connections to the SAS Metadata Server for a user or session.

Class or Interface Name	Description
MdOMRConnection	Contains methods for connecting to the SAS Metadata Server.
CMetadata	Specifies the base interface that is used to describe SAS Metadata Model metadata objects.
MdOMIUtil	Contains utility methods for communicating with the SAS Metadata Server.
AssociationList	Contains methods for defining and maintaining associations.
MdObjectStore	Specifies the container for created or modified metadata objects.
MdUtil	Contains generic utility methods.

Working with the MdFactory Interface

Overview of Tasks That Can Be Performed with the MdFactory Interface

The MdFactory interface provides methods for creating and deleting SAS Metadata Model metadata objects and for invoking the SAS Java Metadata Interface event-handling interface and messaging mechanisms.

Instantiating the Object Factory

The object factory should be instantiated before any other tasks are performed.

```
MdFactory factory = new MdFactoryImpl();
```

If the object factory does not need to be used in a remote environment—that is, it does not need to be available to remote Java Virtual Machines (JVMs)—you can pass in a false value to the constructor. As a result, the factory behaves as if it is running in a local, single JVM environment.

The object factory is instantiated once for each user. If the application is intended to support multiple users, such as a Web application, a separate object factory needs to be created for each user.

Creating Metadata Objects

After you have instantiated the object factory and connected to the SAS Metadata Server (using the makeOMRConnection method of the MdOMRConnection interface), you can then use the methods in the MdFactory interface to create SAS Metadata Model object instances on the client. MdFactory provides the createComplexMetadataObject method for creating objects. The createComplexMetadataObject method creates an object that stores information about a metadata object's attributes and its potential associations. You can use this method to create an object that represents a new or existing object.

The following are examples of the createComplexMetadataObject method. To create an object that represents a new metadata object, specify:

```
MdFactory.createComplexMetadataObject(myNewObjectName,
                                      metadata_type,
                                      8char_target_repository_identifier)
```

To create an object that represents an existing metadata object on the SAS Metadata Server, specify:

```
MdFactory.createComplexMetadataObject(ObjectName,
                                     metadata_type,
                                     identifier_of_existing_metadata_object)
```

An alternate — and preferred — approach for creating objects for an existing metadata object is to issue a getMetadata or getMetadataObjects request. The getMetadata and getMetadataObject methods are available within the MdOMIUtil interface. The createComplexMetadataObject method creates an empty metadata object; it has no attributes or associations set. Use of one of the GetMetadata methods will then be needed to fully populate the object.

You can get the identifiers of all registered repositories on the SAS Metadata Server by using the getRepositories method of the MdOMIUtil interface. You can get the identifier of an existing object instance by using one of the getMetadataObjects methods of the MdOMIUtil interface. For more information about repository and object instance identifiers, see “Identifying Metadata” on page 78.

Invoking the Event Handling Interface

The MdFactoryListener interface notifies other users of the factory every time a metadata object is created, updated, or deleted on the SAS Metadata Server. Notifications are sent when the changes are persisted to the SAS Metadata Server. Users of the factory can use the information to make other changes (for example, to refresh their displays to include the new, modified, or deleted objects).

The MdFactoryListener interface includes the MdFactoryEvent class and the addMdFactoryListener method. You can have multiple listeners in a factory. The addMdFactoryListener method can be instantiated either directly before or after the server connection is made.

A listener should be removed when it is no longer needed. All listeners are automatically removed at the end of the factory’s session.

Deleting Objects

To delete an existing metadata object from the SAS Metadata Server, you must create an object that represents it in the SAS Java Metadata Interface client. Then, you delete both the server and client metadata objects by calling the deleteMetadataObjects method of the MdFactory interface. Calling this method removes the metadata object from both the server and client.

An alternate way to delete existing metadata objects is to use the delete method that is available with each object. Using this method simply marks the object as deleted. The updateMetadataAll method needs to be called to persist this change to the SAS Metadata Server.

Disposing of the Object Factory

To remove the object factory from memory, use the MdFactory dispose method before closing the client application. The dispose method removes the object factory and any remaining object stores. The dispose method should be used whenever the factory is no longer needed.

Working with the MdOMRConnection Interface

The MdOMRConnection interface contains methods for connecting to and disconnecting from the SAS Metadata Server. The MdOMRConnection interface can be retrieved as follows:

```
MdOMRConnection connection = factory.getConnection();
connection.makeOMRConnection(serverName, serverPort, serverUser,
  serverPassword);
```

A client that reads and writes metadata uses the makeOMRConnection method to connect to the SAS Metadata Server. The client disconnects from the server by using the closeOMRConnection method.

The MdOMRConnection interface also provides methods for connecting to the server with the SAS Open Metadata Interface IServer, ISecurity, and ISecurityAdmin server interfaces, and for getting information about the SAS Metadata Server connection. The following table summarizes the methods in the MdOMRConnection interface.

Table 5.2 *Basic MdOMRConnection Methods*

Method Name	Description
closeOMRConnection	Disconnects from the SAS Metadata Server.

Method Name	Description
getIdentityofUserConnected	Gets the Identity object of the connected user.
getPlatformVersion	Gets the SAS version of the SAS Metadata Server as an integer, which when printed as a decimal number has four digits. For example, a SAS Metadata Server running SAS 9.4 returns the value 9400, which represents version 9.4.0.0.
getServerModelVersion	Gets the SAS Metadata Model version number in the form XX.XX. (For example, 12.04.)
makeISecurityConnection	Obtains a handle to the ISecurity server interface, which contains SAS Open Metadata Interface authorization methods.
makeISecurityAdminConnection	Obtains a handle to the ISecurityAdmin server interface, which contains SAS Open Metadata Interface security administration methods.
makeIServerConnection	Obtains a handle to the IServer server interface, which contains SAS Open Metadata Interface server control methods.
makeOMRConnection	Obtains a handle to the IOMI server interface, which contains SAS Open Metadata Interface metadata access methods.

Working with the CMetadata Interface

The CMetadata interface is the parent interface that is used to describe all metadata objects, such as a PhysicalTable, Column, Person, or LogicalServer. The CMetadata interface contains the basic attributes for all metadata objects, such as Name, Description, Id, MetadataCreated time, and MetadataUpdated time. All metadata objects inherit these attributes. They also inherit the routines that are used to get and set these attributes. For example, routines such as getName and setName or getDesc and setDesc are all inherited from CMetadata.

Other frequently used CMetadata methods are summarized in the following table.

Table 5.3 *Frequently Used CMetadata Methods*

Method Name	Description
delete	Marks an object as deleted in its parent object store. The object remains available until the object store is persisted to the SAS Metadata Server with the updateMetadataAll() method.
dispose	Removes the object and all links to an object and clears it from memory.
getCMetadataType	Returns the metadata type of an object.
getObjectStore	Gets the object store for an object.
getRepositoryID	Returns an object's repository identifier.
updateMetadataAll	Persists new and modified objects to the SAS Metadata Server.

Working with the MdOMIUtil Interface

Overview of the Tasks That Can Be Performed with the MdOMIUtil Interface

The MdOMIUtil interface provides wrapper methods for methods in the SAS Open Metadata Interface IOMI server interface. Many of the methods in the MdOMIUtil interface enable you to retrieve existing metadata objects from the SAS Metadata Server.

The MdOMIUtil interface also includes AddMetadata, UpdateMetadata, and DoRequest methods. The AddMetadata and UpdateMetadata methods enable you to pass XML metadata property strings that define SAS Metadata Model objects to add and update metadata objects directly on the SAS Metadata Server, instead of having to create Java objects. The DoRequest method enables you to pass XML-formatted IOMI method calls to the server. Use of these methods is not recommended. They duplicate functionality provided by Java object interfaces provided by the SAS Java Metadata Interface.

The following table summarizes the basic methods in the MdOMIUtil interface.

Table 5.4 Basic MdOMIUtil Methods

Method Name	Description
getRepositories	Gets the ID and name of all repositories registered on the SAS Metadata Server.
getFoundationRepository	Returns the foundation repository for the connected SAS Metadata Server.
getFoundationReposID	Returns the ID of the foundation repository.
getMetadataAllDepths	Gets the properties (attributes and associations) of a specified metadata object.
getMetadataNoCache	Issues a GetMetadata request on the specified object, and then parses the output returned by the SAS Metadata Server so that it is stored in a HashMap. The HashMap contains attribute and association values in key=value pairs.
getMetadataObjectsNoCache	Issues a GetMetadataObjects request on a specified metadata type, and parses the output returned by the SAS Metadata Server so that each returned object's properties are stored in a HashMap.
getMetadataObjectsSubset	Gets metadata objects of the requested metadata type.
getObjectPath	Returns the path of an object that resides in the SAS folder tree.
getUserHomeFolder	Gets the user home folder for the specified user.
DoRequest	Passes an XML-formatted IOMI method call to the SAS Metadata Server.

For reference information about each method, see the SAS Java Metadata Interface documentation at support.sas.com/93api.

Using the Get Methods

The get methods enable you to query metadata.

Most of the get methods require you to specify SAS Open Metadata Interface flags and options to identify the information that you want to retrieve. For example, the getMetadataObjects methods support the OMI_XMLSELECT flag and the <XMLSELECT> element to pass a search string. All of the getMetadata* and getMetadataObjects methods support the OMI_TEMPLATE flag and the

<TEMPLATES> element to enable you to specify the attributes and associations to retrieve in a template.

The SAS Java Metadata Interface Get methods support all of the flags and options that are defined for the SAS Open Metadata Interface IOMI GetMetadataObjects and GetMetadata methods. For reference information about the IOMI methods, flags, and options, see Chapter 7, “Metadata Access (IOMI Interface),” on page 71. For usage information, see:

- Chapter 3, “Using Interfaces That Read and Write Metadata in SAS 9.4,” on page 19
- Chapter 14, “Getting the Properties of a Specified Metadata Object,” on page 315
- Chapter 16, “Getting All Metadata of a Specified Metadata Type,” on page 355
- Chapter 17, “Filtering a GetMetadataObjects Request,” on page 371
- Chapter 15, “Using Templates,” on page 337

To make the flags easier to use, the MdOMIUtil interface defines constant values for each of the flags. These constant values are in the documentation at support.sas.com/93api. Specify these constant values in your SAS Java Metadata Interface method calls instead of the numeric values that are documented for the IOMI server interface methods.

Working with the AssociationList Class

The AssociationList class provides methods to manage the associations between metadata objects. A SAS Open Metadata Interface metadata object instance is defined by its properties. Attributes describe the characteristics of the object instance, and associations describe the object’s relationships with other object instances. All associations on the SAS Metadata Server are bidirectional. An AssociationList object is required to represent each association name.

An AssociationList object is created by submitting a get*AssociationName* method on the metadata object on the SAS Metadata Server. When using this method, substitute a valid association name for *AssociationName*. The following is an example of a get*AssociationName* request:

```
AssociationList columns = tableObject.getColumns();
columns.add(columnObject);
```

- The first statement specifies to get from the SAS Metadata Server for object tableObject a list of all the objects in the Columns association. If a Columns association does not exist, then an empty AssociationList object is created on the client.
- The second statement adds object columnObject to the Columns AssociationList that was retrieved or created.

Whenever you create an AssociationList object for a specified association name, the SAS Java Metadata Interface automatically creates an AssociationList object that represents the reverse association. For example, for each column listed by the getColumns method, the SAS Java Metadata Interface creates AssociationList objects in the object store representing the reverse association. So, for the

preceding example, the columnObject.setTable(tableObject) is performed for the user by the SAS Java Metadata Interface.

If you want to clear the contents of an association, you can use the clear method from the AssociationList class. For the preceding example, you would issue the following:

```
columns.clear();
```

The clear method removes all SAS Metadata Model metadata objects representing both sides of the bidirectional association.

Working with the MdObjectStore Interface

All objects that you create to read metadata or to add or update metadata on the SAS Metadata Server must be contained in an MdObjectStore object. The MdObjectStore object serves as a working container for metadata objects. When you are ready to apply changes to the SAS Metadata Server, all of the new and modified metadata objects in the object store are persisted to the SAS Metadata Server as a group. The object store automatically maintains lists of new, updated, and deleted metadata objects. These lists are used to persist the updates to the SAS Metadata Server.

An object store is created with the statement:

```
MdObjectStore store = MdFactory.createObjectStore();
```

The dispose method should be used when the object store is no longer needed. The dispose method removes all objects in the object store from memory.

Working with the MdUtil Interface

The MdUtil interface has utility methods for defining logging messages within the SAS Java Metadata Interface. There are three different categories of information that can be logged—client/server XML information, debug messages, and performance times for measuring client/server communication. Actual logging is turned on or off with the MdFactory interface. The MdUtil interface controls the output of these log messages.

6

Using the SAS Java Metadata Interface

Overview of Creating a SAS Java Metadata Interface Client

The SAS Java Metadata Interface makes it as simple as possible to use the functionality of the SAS Metadata Server in a Java program. Using the SAS Java Metadata Interface, you can write Java client programs that create and update SAS Open Metadata Interface metadata objects as if they were Java objects. There is no need to learn SAS Open Metadata Interface method calls or XML, although users must be familiar with the metadata types in the SAS Metadata Model, and with flags and options that are supported by SAS Open Metadata Interface IOMI server interface methods. For more information, see “Summary Table of IOMI Flags” on page 81. Also see “Summary Table of IOMI Options” on page 87.

The SAS Java Metadata Interface follows Java distributed programming standards such as CORBA and JDBC. When you write a Java client program that uses the SAS Metadata Server—whether that program is an applet, a stand-alone application, a servlet, or an enterprise JavaBean—you can focus on exploiting the features of the SAS Metadata Server, rather than figuring out how to communicate with it.

The SAS Java Metadata Interface includes all of the tools that you need to work with the SAS Metadata Server from a Java client. Knowledge of distributed programming

standards is not required, and you are not required to license any third-party software.

Advantages over the IOMI Server Interface

The SAS Java Metadata Interface has the following advantages over the SAS Open Metadata Interface IOMI server interface:

- It provides a simpler interface for connecting and disconnecting from the SAS Metadata Server. Clients issue method calls to connect to the SAS Metadata Server, instead of specifying IOMI connection factory classes.
- The SAS Java Metadata Interface provides a set of generated Java interfaces and implementation classes that represent the SAS Metadata Model. Once an object is created, clients can define, read, and update its attributes and associations using getter or setter methods, instead of submitting XML requests. The SAS Java Metadata Interface seamlessly handles all XML creation and parsing.
- The SAS Java Metadata Interface uses the concept of an object store that acts like a work unit. Object stores enable clients to make and test multiple changes locally within the client. Then, object stores persist all of the changes to the SAS Metadata Server in a single request. IOMI server interface methods typically support smaller requests and update the SAS Metadata Server directly.

Getting Started

This section provides the steps to construct and execute a SAS Java Metadata Interface client that reads and writes metadata.

The first step in developing and running a client program is to make sure that you have access to a properly configured SAS Metadata Server. You should have a properly configured SAS Metadata Server if your site performed a SAS planned installation.

After the SAS Metadata Server has been configured, you can begin developing a SAS Java Metadata Interface client that uses it. All SAS Java Metadata Interface clients access a SAS Metadata Server using the following steps:

1. Instantiate an object factory.
2. Connect to the SAS Metadata Server.
3. Create Java object instances that represent SAS Metadata Model metadata objects and modify attributes and associations as needed.
4. Persist changes to the SAS Metadata Server.

Read the following topics for instructions about how to implement the preceding steps:

Example code fragments are given to illustrate each step. To see how the code examples are submitted in an actual program, see "Sample Program" on page 55.

An object factory is needed for each user who will use an application. Therefore, each user will have their own factory instance.

The examples given do not attempt to show how to create multiple object factories. Their goal is to show how a typical user connects to the SAS Metadata Server and issues SAS Java Metadata Interface method calls that create, read, and persist metadata objects on the SAS Metadata Server.

Instantiating an Object Factory and Connecting to the SAS Metadata Server

This section provides an example of the SAS Java Metadata Interface calls necessary to instantiate an object factory and to connect to the SAS Metadata Server.

When using the SAS Java Metadata Interface, you create an object factory by instantiating the MdFactory interface. This interface contains all of the methods to create Java metadata objects and to invoke Java event-handling and messaging mechanisms.

You create a connection to the SAS Metadata Server using the makeOMRConnection method from the MdOMRConnection interface.

An object factory is instantiated once per user. The following code instantiates an object factory, and creates a connection to the SAS Metadata Server using the makeOMRConnection method:

```
/**
 * The object factory instance.
 */
private MdFactory _factory = null;

/**
 * Default constructor
 */
public MdTesterExamples()
{
   // Calls the factory's constructor
   initializeFactory();
}

private void initializeFactory()
{
```

```
      try
      {
         // Initializes the factory.  The boolean parameter is used to
         // determine if the application is running in a remote or local environment.
         // If the data does not need to be accessible across remote JVMs,
         // then "false" can be used, as shown here.
         _factory = new MdFactoryImpl(false);

         // Defines debug logging, but does not turn it on.
         boolean debug = false;
         if (debug)
         {
            _factory.setDebug(false);
            _factory.setLoggingEnabled(false);

            // Sets the output streams for logging.  The logging output can be
            // directed to any OutputStream, including a file.
            _factory.getUtil().setOutputStream(System.out);
            _factory.getUtil().setLogStream(System.out);
         }

         // To be notified of changes that have been persisted to the SAS Metadata
         // Server within this factory (this includes adding objects, updating
         // objects, and deleting objects), we can add a listener to the factory
         // here. See MdFactory.addMdFactoryListener().
         // A listener is not needed for this example.
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }

   /**
    * The following statements define variables for SAS Metadata Server
    * connection properties, instantiate a connection factory, issue
    * the makeOMRConnection method, and check exceptions for error conditions.
    *
    */
   public boolean connectToServer()
   {
      String serverName = "MACHINE_NAME";
      String serverPort = "8561";
      String serverUser = "USERNAME";
      String serverPass = "PASSWORD";


      try
      {
         MdOMRConnection connection = _factory.getConnection();

         // This statement makes the connection to the server.
         connection.makeOMRConnection(serverName, serverPort, serverUser, serverPass);

         // The following statements define error handling and error
         // reporting messages.
```

```
      }
      catch (MdException e)
      {
         Throwable t = e.getCause();
         if (t != null)
         {
            String ErrorType = e.getSASMessageSeverity();
            String ErrorMsg = e.getSASMessage();
            if (ErrorType == null)
            {
               // If there is no SAS server message, write a Java/CORBA message.
            }
            else
            {
               // If there is a message from the server:
               System.out.println(ErrorType + ": " + ErrorMsg);
            }
            if (t instanceof org.omg.CORBA.COMM_FAILURE)
            {
               // If there is an invalid port number or host name:
               System.out.println(e.getLocalizedMessage());
            }
            else if (t instanceof org.omg.CORBA.NO_PERMISSION)
            {
               // If there is an invalid user ID or password:
               System.out.println(e.getLocalizedMessage());
            }
         }
         else
         {
            // If we cannot find a nested exception, get message and print.
            System.out.println(e.getLocalizedMessage());
         }
         // If there is an error, print the entire stack trace.
         e.printStackTrace();
         return false;
      }
      catch (RemoteException e)
      {
         // Unknown exception.
         e.printStackTrace();
         return false;
      }
      // If no errors occur, then a connection is made.
      return true;
   }
```

From this example, here are the results:

- An object factory in which to create SAS Metadata Model metadata objects.
- Log and output location definitions that can be turned on and off for debugging. SAS Java Metadata Interface logging methods should not be used for client-side logging.
- An available connection to the SAS Metadata Server.

You can now get information about repositories defined on the SAS Metadata Server, and you can create metadata object instances.

Getting Information about Repositories

Before you can read or write metadata, you must identify the repositories that are registered on a SAS Metadata Server. You should be familiar with the repository identifiers to indicate which repository to access. You can list the repositories that are defined on a SAS Metadata Server by using the getRepositories method on the MdOMIUtil interface.

```
/**
 * The following statements list the repositories that are registered
 * on the SAS Metadata Server.
 * @return the list of available repository (list of CMetadata objects)
 */
public List<CMetadata> getAllRepositories()
{
   try
   {
      System.out.println("\nThe repositories contained on this SAS Metadata " +
        "Server are:");

      // The getRepositories method lists all repositories.
      MdOMIUtil omiUtil = _factory.getOMIUtil();
      List<CMetadata> reposList = omiUtil.getRepositories();
      for (CMetadata repository : reposList)
      {
         // Print the name and id of each repository.
         System.out.println("Repository: " +
                            repository.getName()
                            + " (" + repository.getFQID() +")");
      }
      return reposList;
   }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
   {
      e.printStackTrace();
   }
   return Collections.emptyList();
}
```

Here is an example of the output that might be returned by the GetRepositories method:

```
The repositories contained on this SAS Metadata Server are:
Repository: Foundation (A0000001.A5PCE796)
Repository: MyProject (A0000001.A5RVLQQ9)
Repository: Custom1 (A0000001.A5T27ER8)
Repository: Custom2 (A0000001.A5IUI1BI)
```

The two-part number in each line is the repository identifier. The first part of the number (A0000001) is the SAS Repository Manager identifier and is the same for all repositories. The second part of the number is the unique repository ID. This is the identifier that we will use to create and read metadata.

The CMetadata interface is the base interface that is used to describe all metadata objects.

Creating Objects

You can create metadata objects by using the methods in the MdFactory interface. You must create a Java object instance for every new and existing metadata object that you want to read, update, or delete in a SAS Metadata Repository. You must create an object store in which to hold the metadata objects. The object store maintains a list of the metadata objects that need to be persisted to the SAS Metadata Server with a single request.

The following code creates a new PhysicalTable object, a new Column object, and a new Keyword object. The code creates associations between these objects. After the metadata objects are created, they are persisted to the SAS Metadata Server.

Notes:

- To create and persist a new metadata object, you must specify a metadata repository in which to store the object. You can specify a repository identifier directly in the createComplexMetadataObject method.
- Because these are new metadata objects, they are assigned permanent metadata object identifiers when they are persisted to the SAS Metadata Server. A request that creates Java object instances to represent existing metadata objects needs to determine their metadata object identifiers before persisting changes to the SAS Metadata Server. For more information, see “Getting and Updating Existing Objects” on page 50.
- When creating a new metadata object with createComplexMetadataObject, a valid metadata type must be passed in. For a list of all valid metadata types, see com.sas.metadata.remote.MetadataObjects.
- The example defines PublicType and UsageVersion attributes for the table and column objects, and an association to a folder. As a result, the objects are visible in the SAS Folders tree, and they can be accessed by the utilities that are available to objects in that tree. Later, these attribute values can be used to simplify metadata queries. For more information, see Chapter 3, “Using Interfaces That Read and Write Metadata in SAS 9.4,” on page 19.

  A metadata object must be unique within a folder. The example includes a private method that verifies that no other tables with the same Name and PublicType attribute values exist in the folder.

```
/**
    * The following statements create a table, column, and keyword on the column.
    * The objects are created in the user's My Folder folder.
    * @param repository CMetadata object with id of form:  A0000001.A5KHUI98
    */
   public void createTable(CMetadata repository)
   {
      if (repository != null)
```

```
{
   try
   {
      System.out.println("\nCreating objects on the server...");

      // We have a repository object.
      // We use the getFQID method to get its fully qualified ID.
      String reposFQID = repository.getFQID();

      // We need the short repository ID to create an object.
      String shortReposID = reposFQID.substring(reposFQID.indexOf('.') + 1,
                            reposFQID.length());

      // Now we create an object store to hold our objects.
      // This will be used to maintain a list of objects to persist
      // to the SAS Metadata Server.
      MdObjectStore store = _factory.createObjectStore();
      String tableName = "TableTest";
      String tableType = "Table";

      // The getUserHomeFolder method retrieves (or creates, if necessary)
      // the user's My Folder folder.  If the folder does not
      // exist, the method automatically creates it.
      // This is the folder in which we will create the table.
      Tree myFolder = _factory.getOMIUtil().getUserHomeFolder(store, "",
            MdOMIUtil.FOLDERTYPE_MYFOLDER, "", 0, true);

      // Before creating any objects, we must verify that the Table does
      // not already exist within the parent folder.  The table cannot be
      // created if it is not unique within the folder.
      if (!isUnique(myFolder, tableName, tableType))
      {
         // Create a PhysicalTable object named "TableTest".
         PhysicalTable table = (PhysicalTable) _factory.createComplexMetadataObject
            (store,
                  null,
                  "TableTest",
                  MetadataObjects.PHYSICALTABLE,
                  shortReposID);

         // Set the PublicType and UsageVersion attributes for the table.
         table.setPublicType("Table");
         table.setUsageVersion(1000000.0);

         // Add the table to the user's "My Folder" location.
         table.getTrees().add(myFolder);

         // Create a Column named "ColumnTest".
         Column column = (Column) _factory.createComplexMetadataObject
            (store,
                  null,
                  "ColumnTest",
                  MetadataObjects.COLUMN,
                  shortReposID);

         // Set the attributes of the column, including PublicType and
```

```
            // UsageVersion.
            column.setPublicType("Column");
            column.setUsageVersion(1000000.0);
            column.setColumnName("MyTestColumnName");
            column.setSASColumnName("MyTestSASColumnName");
            column.setDesc("This is a description of a column");

            // Use the get"AssociationName"() method to associate the column with
            // the table. This method creates an AssociationList object for the table
            // object. The inverse association will be created automatically.
            // The add(MetadataObject) method adds myColumn to the AssociationList.
            table.getColumns().add(column);

            // Create a keyword for the column named "KeywordTest".
            Keyword keyword = (Keyword) _factory.createComplexMetadataObject
               (store,
                     null,
                     "KeywordTest",
                     MetadataObjects.KEYWORD,
                     shortReposID);

            // Associate the keyword with the column.
            column.getKeywords().add(keyword);

            // Now, persist all of these changes to the server.
            table.updateMetadataAll();
         }

         // When finished, clean up the objects in the store if they
         // are no longer being used.
         store.dispose();
      }
      catch (MdException e)
      {
         e.printStackTrace();
      }
      catch (RemoteException e)
      {
         e.printStackTrace();
      }
   }
}

/**
 * isUnique() is a private method that determines whether an object is unique
 * within a given folder.
 * @param folder is the name of the parent folder
 * @param name is the Name value of the object
 * @param type is the PublicType value of the object
 * @return true if an object with the specified name and type exists within
 * the folder.
 */
private boolean isUnique(Tree folder, String name, String type)
   throws RemoteException, MdException
{
   // Now, retrieve the objects in the folder and make sure that the folder doesn't
```

```
      // already contain this table. The object's Name and PublicType attribute values
      //  are used to determine if it is unique.
      List members = folder.getMembers();
      for (Iterator iter = members.iterator(); iter.hasNext(); )
      {
         CMetadata meta = (CMetadata) iter.next();
         if (meta instanceof PrimaryType)
         {
            // Verify that the types and object names match
            // A case-insensitive match should be used when comparing the names.
            if (type.equals(((PrimaryType) meta).getPublicType()) &&
                name.equals(meta.getName()))
            {
               // We found a match.
               return true;
            }
         }
      }
      return false;
   }
```

For more information about object stores and AssociationList objects, see “SAS Java Metadata Interface Overview” on page 27.

Getting and Updating Existing Objects

To update an existing metadata object, you must know its metadata object identifier. The SAS Java Metadata Interface provides several ways to get information about existing metadata objects. This section provides an example of one way to get information about the metadata objects created in “Creating Objects” on page 47. The example uses the getMetadataObjectsSubset method from the MdOMIUtil interface.

The getMetadataObjectsSubset method gets a list of metadata objects in the repository of a specified metadata type. The method supports the use of SAS Open Metadata Interface flags and options to enable you to specify properties to return in the request, and to filter the metadata objects that are returned by the request. In the example, the <TEMPLATES> and <XMLSELECT> elements and their corresponding flags get all PhysicalTable objects named “TableTest,” their associated Column and Keyword metadata objects, and all attributes of the objects. The PublicType attribute is included in the <XMLSELECT> search string to specify to process only PhysicalTable objects that have their PublicType attribute set.

Note: The <TEMPLATES> and <XMLSELECT> elements submit input to the SAS Metadata Server as a string literal (a quoted string). To ensure that the string is parsed correctly, you must escape any additional double quotation marks specified in the input string (such as those denoting XML attribute values) to indicate that they should be treated as characters. In this example, additional double quotation marks are escaped by using a backslash character.

The objects are created in the specified object store, and they can be edited.

```
/**
```

```
 * This example reads the newly created objects back from the
 * SAS Metadata Server.
 * @param repository identifies the repository from which to read our objects.
 */
public void readTable(CMetadata repository)
{
   if(repository != null)
   {
      try
      {
         System.out.println("\nReading objects from the server...");

         // First we create an MdObjectStore as a container for the
         // objects that we will create/read/persist to the server as
         // one collection.
         MdObjectStore store = _factory.createObjectStore();

         // The following statements define variables used within the
         // getMetadataObjectsSubset method.These XML strings are used in conjunction
         // with SAS Open Metadata Interface flags. The <XMLSELECT> element
         // specifies filter criteria. The objects returned are filtered by the
         // PublicType and Name values. The <TEMPLATES> element specifies the
         // associations to be expanded for each object.

         String xmlSelect = "<XMLSELECT Search=\"*[@PublicType='Table' and " +
                            "@Name='TableTest']\"/>";
         String template =
            "<Templates>" +
               "<PhysicalTable>" +
                 "<Columns/>" +
               "</PhysicalTable>" +
               "<Column>" +
                  "<Keywords/>" +
               "</Column>" +
             "</Templates>";

         // Add the XMLSELECT and TEMPLATES strings together.
         String sOptions = xmlSelect + template;

         // The following statements go to the server with a fully-qualified
         // repository ID and specify the type of object we are searching for
         // (MetadataObjects.PHYSICALTABLE) using the OMI_XMLSELECT, OMI_TEMPLATE,
         // OMI_ALL_SIMPLE, and OMI_GET_METADATA flags. OMI_ALL_SIMPLE specifies
         // to get all simple attributes for all objects that are returned.
         // OMI_GET_METADATA activates the GetMetadata flags in the GetMetadataObjects
         // request.
         //
         // The table, column, and keyword will be read from the server and created
         // within the specified object store.
         int flags = MdOMIUtil.OMI_XMLSELECT | MdOMIUtil.OMI_TEMPLATE |
                     MdOMIUtil.OMI_ALL_SIMPLE | MdOMIUtil.OMI_GET_METADATA;
         List tableList = _factory.getOMIUtil().getMetadataObjectsSubset(store,
                                                repository.getFQID(),
                                                MetadataObjects.PHYSICALTABLE,
                                                flags,
                                                sOptions);
```

```
            Iterator iter = tableList.iterator();
            while (iter.hasNext())
            {
               // Print the Name, Id, PublicType, UsageVersion, and ObjPath values
               // of the table returned from the server. ObjPath is the folder location.
               PhysicalTable table = (PhysicalTable) iter.next();
               System.out.println("Found table: " + table.getName() + " (" +
                                  table.getId() + ")");

               System.out.println("\tType: " + table.getPublicType());
               System.out.println("\tUsage Version: " + table.getUsageVersion());
               System.out.println("\tPath: " + _factory.getOMIUtil().getObjectPath(store,
                                                 table, false));

               // Get the list of columns for this table.
               AssociationList columns = table.getColumns();
               for (int i = 0; i < columns.size(); i++)
               {
                  // Print the Name, Id, PublicType, UsageVersion, Desc, and ColumnName
                  // values for each column associated with the table.
                  Column column = (Column) columns.get(i);
                  System.out.println("Found column: " + column.getName() + " (" +
                                     column.getId() + ")");

                  System.out.println("\tType: " + column.getPublicType());
                  System.out.println("\tUsage Version: " + column.getUsageVersion());
                  System.out.println("\tDescription: " + column.getDesc());
                  System.out.println("\tColumnName: " + column.getColumnName());

                  // Get the list of keywords associated with the columns.
                  AssociationList keywords = column.getKeywords();
                  for (int j = 0; j < keywords.size(); j++)
                  {
                     // Print the Name and Id values of each keyword associated with
                     // the column.
                     Keyword keyword = (Keyword) keywords.get(j);
                     System.out.println("Found keyword: " + keyword.getName() + " (" +
                                        keyword.getId() + ")");
                  }
               }
            }

            // When finished, clean up the objects in the store if they
            // are no longer being used.
            store.dispose();
         }
         catch (MdException e)
         {
            e.printStackTrace();
         }
         catch (RemoteException e)
         {
            e.printStackTrace();
         }
      }
   }
```

Here is the output of the code:

```
Reading objects from the server...
Found table: TableTest (A5PCE796.B8002WKO)
    Type: Table
    Usage Version: 1000000.0
    Path: /User Folders/sasdemo/My Folder/TableTest
Found column: ColumnTest (A5PCE796.B5008RAF)
    Type: Column
    Usage Version: 1000000.0
    Description: This is a description of a column
    ColumnName: MyTestColumnName
Found keyword: KeywordTest (A5PCE796.AX001O1D)
```

The output prints the name, metadata object identifier, and PublicType and UsageVersion values of the table and column objects. In addition, it prints the name and metadata object identifier of the Keyword object. And, it prints the path of the table in the SAS Folders tree, and the Description and ColumnName values of the column object.

For more information about the search criteria supported in the <XMLSELECT> element, see Chapter 17, “Filtering a GetMetadataObjects Request,” on page 371. For more information about the templates supported in the <TEMPLATES> element, see “Understanding Templates” on page 337.

Deleting Objects

This section provides an example of how to delete metadata objects. Deleting objects is similar to updating objects. You must create a Java object instance that represents the server metadata object on the client before you can delete it.

In this example, the getMetadataObjectsSubset method gets the PhysicalTable object that was created and updated in “Creating Objects” on page 47, and in “Getting and Updating Existing Objects” on page 50. The deleteMetadataObjects method deletes the objects.

Because PhysicalTable is a PrimaryType subtype in the SAS Metadata Model, and the object instance stores a valid TypeName value in the PublicType attribute, there is no need to specify a template to get the associated Column and Keyword objects, and there is no need to specifically delete the objects. When you delete an object that specifies the name of a type definition from the SAS type dictionary in the PublicType attribute, the SAS Metadata Server uses the template that is internalized in the type definition to identify the associated objects in its logical metadata definition. The metadata server automatically deletes these associated objects as well. The getMetadataObjectsSubset method is in the MdOMIUtil interface. The deleteMetadataObjects method is in the MdFactory interface.

```
/**
    * This example deletes the objects that we created.
    * @param repository
    */
   public void deleteTable(CMetadata repository)
   {
      if(repository != null)
      {
```

```
            try
            {
               System.out.println("\nDeleting the objects from the server...");
               MdObjectStore store = _factory.createObjectStore();

               // Create a list of the objects that need to be deleted
               // from the server.
               List<CMetadata> deleteList = new ArrayList<CMetadata>();

               // Query for the table again.
               String xmlSelect = "<XMLSELECT Search=\"*[@PublicType='Table' and " +
                                  "@Name='TableTest']\"/>";

                  // Note: Since the object has a valid PublicType value, the SAS
                  // Metadata Server automatically deletes all objects associated
                  // with the table, such as its Column and Keyword objects, when the table
                  // is deleted. There is no need to specify a template to delete
                  // the associated objects.

               int flags = MdOMIUtil.OMI_XMLSELECT;
               List tableList = _factory.getOMIUtil().getMetadataObjectsSubset(store,
                                                      repository.getFQID(),
                                                      MetadataObjects.PHYSICALTABLE,
                                                      flags,
                                                      xmlSelect);


               // Add the found objects to the delete list.
               Iterator iter = tableList.iterator();
               while (iter.hasNext())
               {
                  PhysicalTable table = (PhysicalTable) iter.next();
                  deleteList.add(table);
               }

               // Delete everything that is in the delete list.
               if (deleteList.size() > 0)
               {
                  System.out.println("Deleting " + deleteList.size() + " objects");
                  _factory.deleteMetadataObjects(deleteList);
               }

               // When finished, clean up the objects in the store if it is no longer
               // being used
               store.dispose();
            }
            catch (MdException e)
            {
               e.printStackTrace();
            }
            catch (RemoteException e)
            {
               e.printStackTrace();
            }
         }
      }
```

For an executable version of this example and the examples in “Creating Objects,” “Getting and Updating Existing Objects,” and a few additional examples, see “Sample Program” on page 55.

Sample Program

The following is an executable version of the code examples from Chapter 6, “Using the SAS Java Metadata Interface,” on page 41.

```
/**
 * Copyright (c) 2011 by SAS Institute Inc., Cary, NC 27513
 */

package com.sas.metadata.remote.test;

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import com.sas.metadata.remote.AssociationList;
import com.sas.metadata.remote.CMetadata;
import com.sas.metadata.remote.Column;
import com.sas.metadata.remote.Keyword;
import com.sas.metadata.remote.MdException;
import com.sas.metadata.remote.MdFactory;
import com.sas.metadata.remote.MdFactoryImpl;
import com.sas.metadata.remote.MdOMIUtil;
import com.sas.metadata.remote.MdOMRConnection;
import com.sas.metadata.remote.MdObjectStore;
import com.sas.metadata.remote.MetadataObjects;
import com.sas.metadata.remote.PhysicalTable;
import com.sas.metadata.remote.PrimaryType;
import com.sas.metadata.remote.Tree;


/**
 * This is a test class that contains the examples for SAS Java Metadata Interface.
 */
public class MdTesterExamples
{

   /**
    * The object factory instance.
    */
   private MdFactory _factory = null;

   /**
    * Default constructor
    */
   public MdTesterExamples()
   {
```

```
      // Call the factory's constructor.
      initializeFactory();
   }

   private void initializeFactory()
   {
      try
      {
         // Initialize the factory.  The boolean parameter is used to determine if
         // the application is running in a remote or local environment.  If the
         // data does not need to be accessible across remote JVMs, then
         // "false" can be used, as shown here.
         _factory = new MdFactoryImpl(false);

         // Defines debug logging, but does not turn it on.
         boolean debug = false;
         if (debug)
         {
            _factory.setDebug(false);
            _factory.setLoggingEnabled(false);

            // Sets the output streams for logging.  The logging output can be
            // directed to any OutputStream, including a file.
            _factory.getUtil().setOutputStream(System.out);
            _factory.getUtil().setLogStream(System.out);
         }

         // To be notified when changes have been persisted to the SAS Metadata Server
         // within this factory (this includes adding objects, updating objects, and
         // deleting objects), we can add a listener to the factory here.
         // See MdFactory.addMdFactoryListener()
         // A listener is not needed for this example.
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }

   /**
    * The following statements make a connection to the SAS Metadata Server
    * and check exceptions if there is an error connecting.  The server name,
    * port, user, and password variables must be substituted with actual values.
    * @return true if the connection was successful.
    */
   public boolean connectToServer()
   {
      String serverName = "MACHINE_NAME";
      String serverPort = "8561";
      String serverUser = "USERNAME";
      String serverPass = "PASSWORD";

      try
      {
         MdOMRConnection connection = _factory.getConnection();
```

```
      // This statement makes the connection to the server.
      connection.makeOMRConnection(serverName, serverPort, serverUser, serverPass);

      // The following statements define error handling and error
      // reporting messages.
   }
   catch (MdException e)
   {
      Throwable t = e.getCause();
      if (t != null)
      {
         String ErrorType = e.getSASMessageSeverity();
         String ErrorMsg = e.getSASMessage();
         if (ErrorType == null)
         {
            // If there is no SAS server message, write a Java/CORBA message.
         }
         else
         {
            // If there is a message from the server:
            System.out.println(ErrorType + ": " + ErrorMsg);
         }
         if (t instanceof org.omg.CORBA.COMM_FAILURE)
         {
            // If there is an invalid port number or host name:
            System.out.println(e.getLocalizedMessage());
         }
         else if (t instanceof org.omg.CORBA.NO_PERMISSION)
         {
            // If there is an invalid user ID or password:
            System.out.println(e.getLocalizedMessage());
         }
      }
      else
      {
         // If we cannot find a nested exception, get message and print.
         System.out.println(e.getLocalizedMessage());
      }
      // If there is an error, print the entire stack trace.
      e.printStackTrace();
      return false;
   }
   catch (RemoteException e)
   {
      // Unknown exception.
      e.printStackTrace();
      return false;
   }
   // If no errors occur, then a connection is made.
   return true;
}

/**
 * The following statements get and display the status and version
 * of the SAS Metadata Server.
 */
```

```
public void displayServerInformation()
{
   try
   {
      MdOMRConnection connection = _factory.getConnection();

      // Check the status of the server.
      System.out.println("\nGetting server status...");
      int status = connection.getServerStatus();
      switch (status)
      {
         case MdOMRConnection.SERVER_STATUS_OK:
            System.out.println("Server is running");
            break;
         case MdOMRConnection.SERVER_STATUS_PAUSED:
            System.out.println("Server is paused");
            break;
         case MdOMRConnection.SERVER_STATUS_ERROR:
            System.out.println("Server is not running");
            break;
      }

      // Check the version of the server.
      int version = connection.getPlatformVersion();
      System.out.println("Server version: " + version);
   }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
   {
      e.printStackTrace();
   }
}

/**
 * The following statements get information about the foundation repository.
 * @return the foundation repository
 */
public CMetadata getFoundationRepository()
{
   try
   {
      System.out.println("\nGetting the Foundation repository...");

      // The getFoundationRepository method gets the foundation repository.
      return _factory.getOMIUtil().getFoundationRepository();
   }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
   {
      e.printStackTrace();
```

```
   }
   return null;
}

/**
 * The following statements list the repositories that are registered
 * on the SAS Metadata Server.
 * @return the list of available repository (list of CMetadata objects)
 */
public List<CMetadata> getAllRepositories()
{
   try
   {
      System.out.println("\nThe repositories contained on this SAS Metadata " +
        "Server are:");

      // The getRepositories method lists all repositories.
      MdOMIUtil omiUtil = _factory.getOMIUtil();
      List<CMetadata> reposList = omiUtil.getRepositories();
      for (CMetadata repository : reposList)
      {
         // Print the name and id of each repository.
         System.out.println("Repository: " +
                            repository.getName()
                            + " (" + repository.getFQID() +")");
      }
      return reposList;
   }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
   {
      e.printStackTrace();
   }
   return Collections.emptyList();
}

/**
 * The following statements list the metadata types available on the
 * SAS Metadata Server and their descriptions.
 */
public void displayMetadataTypes()
{
   try
   {
      System.out.println("\nThe object types contained on this SAS Metadata " +
              "Server are:");

      // Metadata types are listed with the getTypes method.
      List<String> nameList = new ArrayList<String>();
      List<String> descList = new ArrayList<String>();
      _factory.getOMIUtil().getTypes(nameList, descList);
      Iterator<String> nameIter = nameList.iterator();
      Iterator<String> descIter = descList.iterator();
```

```
            while (nameIter.hasNext() && descIter.hasNext())
            {
               // Print the name and description of each metadata object type.
               String name = nameIter.next();
               String desc = descIter.next();
               System.out.println("Type: " +
                                  name +
                                  " - Description: " +
                                  desc);
            }
         }
         catch (MdException e)
         {
            e.printStackTrace();
         }
         catch (RemoteException e)
         {
            e.printStackTrace();
         }
      }

      /**
       * The following statements create a table, column, and keyword on the column.
       * The objects are created in the user's My Folder folder.
       * @param repository CMetadata object with id of form:  A0000001.A5KHUI98
       */
      public void createTable(CMetadata repository)
      {
         if (repository != null)
         {
            try
            {
               System.out.println("\nCreating objects on the server...");

               // We have a repository object.
               // We use the getFQID method to get its fully qualified ID.
               String reposFQID = repository.getFQID();

               // We need the short repository ID to create an object.
               String shortReposID = reposFQID.substring(reposFQID.indexOf('.') + 1,
                                     reposFQID.length());

               // Now we create an object store to hold our objects.
               // This will be used to maintain a list of objects to persist
               // to the SAS Metadata Server.
               MdObjectStore store = _factory.createObjectStore();
               String tableName = "TableTest";
               String tableType = "Table";

               // The getUserHomeFolder method retrieves (or creates, if necessary)
               // the user's My Folder folder.  If the folder does not
               // exist, the method automatically creates it.
               // This is the folder in which we will create the table.
               Tree myFolder = _factory.getOMIUtil().getUserHomeFolder(store, "",
                     MdOMIUtil.FOLDERTYPE_MYFOLDER, "", 0, true);
```

```
// Before creating any objects, we must verify that the Table does
// not already exist within the parent folder.  The table cannot be
// created if it is not unique within the folder.
if (!isUnique(myFolder, tableName, tableType))
{
   // Create a PhysicalTable object named "TableTest".
   PhysicalTable table = (PhysicalTable) _factory.createComplexMetadataObject
      (store,
            null,
            "TableTest",
            MetadataObjects.PHYSICALTABLE,
            shortReposID);

   // Set the PublicType and UsageVersion attributes for the table.
   table.setPublicType("Table");
   table.setUsageVersion(1000000.0);

   // Add the table to the user's "My Folder" location.
   table.getTrees().add(myFolder);

   // Create a Column named "ColumnTest".
   Column column = (Column) _factory.createComplexMetadataObject
      (store,
            null,
            "ColumnTest",
            MetadataObjects.COLUMN,
            shortReposID);

   // Set the attributes of the column, including PublicType and
   // UsageVersion.
   column.setPublicType("Column");
   column.setUsageVersion(1000000.0);
   column.setColumnName("MyTestColumnName");
   column.setSASColumnName("MyTestSASColumnName");
   column.setDesc("This is a description of a column");

   // Use the get"AssociationName"() method to associate the column with
   // the table. This method creates an AssociationList object for the table
   // object. The inverse association will be created automatically.
   // The add(MetadataObject) method adds myColumn to the AssociationList.
   table.getColumns().add(column);

   // Create a keyword for the column named "KeywordTest".
   Keyword keyword = (Keyword) _factory.createComplexMetadataObject
      (store,
            null,
            "KeywordTest",
            MetadataObjects.KEYWORD,
            shortReposID);

   // Associate the keyword with the column.
   column.getKeywords().add(keyword);

   // Now, persist all of these changes to the server.
   table.updateMetadataAll();
}
```

```
            // When finished, clean up the objects in the store if they
            // are no longer being used.
            store.dispose();
         }
         catch (MdException e)
         {
            e.printStackTrace();
         }
         catch (RemoteException e)
         {
            e.printStackTrace();
         }
      }
   }

   /**
    * isUnique() is a private method that determines whether an object is unique
    * within a given folder.
    * @param folder is the name of the parent folder
    * @param name is the Name value of the object
    * @param type is the PublicType value of the object
    * @return true if an object with the specified name and type exists within
    * the folder.
    */
   private boolean isUnique(Tree folder, String name, String type)
      throws RemoteException, MdException
   {
      // Now, retrieve the objects in the folder and make sure that the folder doesn't
      // already contain this table. The object's Name and PublicType attribute values
      //  are used to determine if it is unique.
      List members = folder.getMembers();
      for (Iterator iter = members.iterator(); iter.hasNext(); )
      {
         CMetadata meta = (CMetadata) iter.next();
         if (meta instanceof PrimaryType)
         {
            // Verify that the types and object names match
            // A case-insensitive match should be used when comparing the names.
            if (type.equals(((PrimaryType) meta).getPublicType()) &&
                name.equals(meta.getName()))
            {
               // We found a match.
               return true;
            }
         }
      }
      return false;
   }

   /**
    * The following statements read the newly created objects back from the
    * SAS Metadata Server.
    * @param repository identifies the repository from which to read our objects.
    */
   public void readTable(CMetadata repository)
```

```
{
   if(repository != null)
   {
      try
      {
         System.out.println("\nReading objects from the server...");

         // First we create an MdObjectStore as a container for the
         // objects that we will create/read/persist to the server as
         // one collection.
         MdObjectStore store = _factory.createObjectStore();

         // The following statements define variables used within the
         // getMetadataObjectsSubset method. These XML strings are used
         // with SAS Open Metadata Interface flags. The <XMLSELECT> element
         // specifies filter criteria. The objects returned are filtered by the
         // PublicType and Name values. The <TEMPLATES> element specifies the
         // associations to be expanded for each object.

         String xmlSelect = "<XMLSELECT Search=\"*[@PublicType='Table' and " +
                            "@Name='TableTest']\"/>";
         String template =
            "<Templates>" +
               "<PhysicalTable>" +
                 "<Columns/>" +
               "</PhysicalTable>" +
               "<Column>" +
                  "<Keywords/>" +
               "</Column>" +
             "</Templates>";

         // Add the XMLSELECT and TEMPLATES strings together.
         String sOptions = xmlSelect + template;

         // The following statements go to the server with a fully-qualified
         // repository ID and specify the type of object we are searching for
         // (MetadataObjects.PHYSICALTABLE) using the OMI_XMLSELECT, OMI_TEMPLATE,
         // OMI_ALL_SIMPLE, and OMI_GET_METADATA flags. OMI_ALL_SIMPLE specifies
         // to get all simple attributes for all objects that are returned.
         // OMI_GET_METADATA activates the GetMetadata flags in the GetMetadataObjects
         // request.
         //
         // The table, column, and keyword will be read from the server and created
         // within the specified object store.
         int flags = MdOMIUtil.OMI_XMLSELECT | MdOMIUtil.OMI_TEMPLATE |
                        MdOMIUtil.OMI_ALL_SIMPLE | MdOMIUtil.OMI_GET_METADATA;
         List tableList = _factory.getOMIUtil().getMetadataObjectsSubset(store,
                                                  repository.getFQID(),
                                                  MetadataObjects.PHYSICALTABLE,
                                                  flags,
                                                  sOptions);
         Iterator iter = tableList.iterator();
         while (iter.hasNext())
         {
            // Print the Name, Id, PublicType, UsageVersion, and ObjPath values
            // of the table returned from the server. ObjPath is the folder location.
```

```
                PhysicalTable table = (PhysicalTable) iter.next();
                System.out.println("Found table: " + table.getName() + " (" +
                                   table.getId() + ")");

                System.out.println("\tType: " + table.getPublicType());
                System.out.println("\tUsage Version: " + table.getUsageVersion());
                System.out.println("\tPath: " + _factory.getOMIUtil().getObjectPath(store,
                                                 table, false));

                // Get the list of columns for this table.
                AssociationList columns = table.getColumns();
                for (int i = 0; i < columns.size(); i++)
                {
                   // Print the Name, Id, PublicType, UsageVersion, Desc, and ColumnName
                   // values for each column associated with the table.
                   Column column = (Column) columns.get(i);
                   System.out.println("Found column: " + column.getName() + " (" +
                                      column.getId() + ")");

                   System.out.println("\tType: " + column.getPublicType());
                   System.out.println("\tUsage Version: " + column.getUsageVersion());
                   System.out.println("\tDescription: " + column.getDesc());
                   System.out.println("\tColumnName: " + column.getColumnName());

                   // Get the list of keywords associated with the columns.
                   AssociationList keywords = column.getKeywords();
                   for (int j = 0; j < keywords.size(); j++)
                   {
                      // Print the Name and Id values of each keyword associated with
                      // the column.
                      Keyword keyword = (Keyword) keywords.get(j);
                      System.out.println("Found keyword: " + keyword.getName() + " (" +
                                         keyword.getId() + ")");
                   }
                }
             }

             // When finished, clean up the objects in the store if they
             // are no longer being used.
             store.dispose();
          }
          catch (MdException e)
          {
             e.printStackTrace();
          }
          catch (RemoteException e)
          {
             e.printStackTrace();
          }
       }
    }

    /**
     * The following statements delete the objects that we created.
     * @param repository
     */
```

```
public void deleteTable(CMetadata repository)
{
   if(repository != null)
   {
      try
      {
         System.out.println("\nDeleting the objects from the server...");
         MdObjectStore store = _factory.createObjectStore();

         // Create a list of the objects that need to be deleted
         // from the server.
         List<CMetadata> deleteList = new ArrayList<CMetadata>();

         // Query for the table again.
         String xmlSelect = "<XMLSELECT Search=\"*[@PublicType='Table' and " +
                            "@Name='TableTest']\"/>";

            // Note: Since the object has a valid PublicType value, the SAS
            // Metadata Server automatically deletes all objects associated
            // with the table, such as its Column and Keyword objects, when the table
            // is deleted. There is no need to specify a template to delete
            // the associated objects.

         int flags = MdOMIUtil.OMI_XMLSELECT;
         List tableList = _factory.getOMIUtil().getMetadataObjectsSubset(store,
                                                repository.getFQID(),
                                                MetadataObjects.PHYSICALTABLE,
                                                flags,
                                                xmlSelect);


         // Add the found objects to the delete list.
         Iterator iter = tableList.iterator();
         while (iter.hasNext())
         {
            PhysicalTable table = (PhysicalTable) iter.next();
            deleteList.add(table);
         }

         // Delete everything that is in the delete list.
         if (deleteList.size() > 0)
         {
            System.out.println("Deleting " + deleteList.size() + " objects");
            _factory.deleteMetadataObjects(deleteList);
         }

         // When finished, clean up the objects in the store if it is no longer
         // being used
         store.dispose();
      }
      catch (MdException e)
      {
         e.printStackTrace();
      }
      catch (RemoteException e)
      {
```

```
            e.printStackTrace();
         }
      }
   }

   /**
    * The following statements display the PhysicalTable objects in the repository.
    * @param repository CMetadata identifies the repository from which to read
    * the objects.
    */
   public void displayAllTables(CMetadata repository)
   {
      try
      {
         // Print a descriptive message about the request.
         System.out.println("\nRetrieving all PhysicalTable objects contained in " +
               " repository " + repository.getName());

         // Use the short repository ID to pass in the method.
         String reposID = repository.getFQID();

         // We get a list of PhysicalTable objects.
         MdObjectStore store = _factory.createObjectStore();

         // Use the OMI_ALL_SIMPLE flag to get all attributes for each table
         // that is returned.
         int flags = MdOMIUtil.OMI_GET_METADATA | MdOMIUtil.OMI_ALL_SIMPLE;
         List tables = _factory.getOMIUtil().getMetadataObjectsSubset
                (store,
                reposID,                          // Repository to search
                MetadataObjects.PHYSICALTABLE,    // Metadata type to search for
                flags,
                "" );

         // Print information about them.
         Iterator iter = tables.iterator();
         while( iter.hasNext())
         {
            PhysicalTable ptable = (PhysicalTable)iter.next();
            System.out.println("PhysicalTable: " +
                               ptable.getName() +
                               ", " +
                               ptable.getFQID() +
                               ", " +
                               ptable.getDesc());
         }

         // When finished, clean up the objects in the store if they
         // are no longer being used.
         store.dispose();
      }
      catch (MdException e)
      {
         e.printStackTrace();
      }
      catch (RemoteException e)
```

```
    {
        e.printStackTrace();
    }
}

/**
 * The following statements retrieve detailed information for a
 * specific PhysicalTable object.
 * @param table the table to retrieve
 */
public void getTableInformation(PhysicalTable table)
{
    try
    {
        // Print a descriptive message about the request.
        System.out.println("\nRetrieving information for a specific PhysicalTable");

        // Create a template to retrieve detailed information for this table.
        String template = "<Templates>" +
                            "<PhysicalTable>" +
                              "<Columns/>" +
                              "<Notes/>" +
                              "<Keywords/>" +
                            "</PhysicalTable>" +
                          "</Templates>";

        // Use the OMI_ALL_SIMPLE flag to get all attributes for the table.
        int flags = MdOMIUtil.OMI_GET_METADATA | MdOMIUtil.OMI_ALL_SIMPLE |
                        MdOMIUtil.OMI_TEMPLATE;
        table = (PhysicalTable) _factory.getOMIUtil().getMetadataAllDepths
               (table,
                null,
                null,
                template,
                flags);

        // Print information about the table.
        System.out.println("Table attributes: ");
        System.out.println("  Name = " + table.getName());
        System.out.println("  Id = " + table.getId());
        System.out.println("  Description = " + table.getDesc());
        System.out.println("  Created Date = " + table.getMetadataCreated());
        System.out.println("  Type = " + table.getPublicType());
        System.out.println("  Usage Version = " + table.getUsageVersion());
        System.out.println("  Path = " +
              _factory.getOMIUtil().getObjectPath((MdObjectStore) table.getObjectStore(),
                    table, false));

        System.out.println("Table associations: ");
        System.out.println("  Number of Columns = " + table.getColumns().size());
        System.out.println("  Number of Keywords = " + table.getKeywords().size());
        System.out.println("  Number of Notes = " + table.getNotes().size());
    }
    catch (MdException e)
    {
        e.printStackTrace();
```

```
            }
            catch (RemoteException e)
            {
               e.printStackTrace();
            }
         }

         /**
          * The main method for the class
          */
         public static void main(String[] args)
         {
            MdTesterExamples tester = new MdTesterExamples();

            // connect to the SAS Metadata Server
            boolean connected = tester.connectToServer();
            if(connected)
            {
               System.out.println("Connected...");
            }
            else
            {
               System.out.println("Error Connecting...");
               return;
            }

            // Now that we are connected, check the status of the server
            tester.displayServerInformation();

            // Get the list of repositories on the server
            tester.getAllRepositories();

            // Get the list of metadata types available on the server
            tester.displayMetadataTypes();

            // Get the foundation repository
            CMetadata repos = tester.getFoundationRepository();
            if (repos != null)
            {
               // Create a new PhysicalTable object and add it to the server
               tester.createTable(repos);

               // Query for the PhysicalTable just added to the metadata server
               tester.readTable(repos);

               // Delete the PhysicalTable
               tester.deleteTable(repos);
            }

            System.exit(1);
         }

      }
```

PART 3

Server Interfaces

7

Metadata Access (IOMI Interface)

Overview of the IOMI Server Interface

Audience

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

We recommend that Java clients use the SAS Java Metadata Interface to read and write metadata instead of using the IOMI server interface directly. Information about IOMI methods is provided for users of PROC METADATA, which enables users to submit IOMI method calls that are formatted for the DoRequest method from the IN= argument. This section also provides background information for users of the SAS Java Metadata Interface and SAS metadata DATA step functions.

Accessing the IOMI Methods

To access the IOMI methods, a client must connect to the SAS Metadata Server. A PROC METADATA user can specify SAS Metadata Server connection options in the procedure, in system options, or in a dialog box. For more information, see "Connection Options" in *SAS Language Interfaces to Metadata*.

Using the 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.

Methods that read and write metadata objects require you to pass a metadata property string that describes the object to the SAS Metadata Server. This metadata property string must be formatted in XML. For information about how to define a metadata property string, see "Constructing a Metadata Property String" on page 76.

Each IOMI method has two output parameters: a return code and a holder for information received from the SAS Metadata Server.

Return Code

The return code is a Boolean operator that indicates whether the method communicated with the SAS Metadata Server. A 0 indicates that communication was established. A 1 indicates that communication was not established. 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 SAS Metadata Server is in the form of formatted XML strings. The output typically mirrors the input, with the exception that requested values are filled in.

Constructing a Metadata Property String

How to Construct a Metadata Property String

To read or write a metadata object, you must pass a string of properties that describe the 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 the following:

- 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 SAS Metadata Model metadata type that you want to read or write, enclosed in angle brackets. See the SAS Metadata Model: Reference for information about supported metadata types. The following example shows the XML element representing the PhysicalTable metadata type:

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

A shorthand method of specifying this XML element is as follows:

```
<PhysicalTable/>
```

metadata type attributes
: specifies the attributes of the metadata type as XML attributes (enclosed in the angle brackets of the metadata type). The following example shows the PhysicalTable metadata type with "NE Sales" as the Name attribute.

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

association name and associated metadata type subelements
: describe the relationship between the metadata object in the main XML element and one or more other metadata types as nested XML elements. For example:

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

The SAS Metadata Model defines the association names that are supported for every metadata type, as well as the associated metadata types that are valid for each association name. In this example, the first nested element, Columns, is the association name subelement. The association name is a label that describes the relationship between the main XML element and the associated object subelement.

The second nested element, Column, is the associated object subelement. The associated object subelement specifies the associated metadata type that you are interested in. The Columns association name supports associated objects of the metadata types Column and ColumnRange. By specifying Column in the property string, you indicate to the SAS Metadata Server that you are interested only in associated objects of this metadata type.

The attributes that you specify in the input metadata property string depend on the method in which it will be used. For example, a metadata property string that is submitted to the AddMetadata method would not specify the Id attribute, as the server assigns a value for this attribute when the metadata object is created. The main element in a metadata property string for the UpdateMetadata and GetMetadata methods must specify the Id attribute.

CAUTION

To meet XML parsing rules, the metadata type, attribute, association, and associated metadata type names that you specify in the metadata property string must exactly match those published in the metadata type documentation.

Quotation Marks and Special Characters

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

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

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

< = <

> = >

& = &

Identifying Metadata

The documentation refers to "general, identifying information" about a metadata object. This phrase refers to the object's 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"/>
```

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

  The *reposid* is assigned to a metadata repository by the SAS Metadata Server when the repository is created. A *reposid* is a unique character string that identifies the metadata repository that stores the object.

  The *instanceid* is assigned to a metadata object by the SAS Metadata Server when the object is created. An *instanceid* is a unique character string that distinguishes one metadata object from other metadata objects of the same metadata type.

Name
: refers to the user-defined name of the metadata object. An object name is a non-null value up to 60 characters. The name cannot start or end with a whitespace character, or contain /, \, or control characters. Names can contain Unicode characters, subject to the previously noted restrictions. In the previous example, the Name value of the table column is New Column.

Note: Because different repository systems use different ID formats, do not make assumptions about the internal format of the Id attribute.

CAUTION
Do not attempt to assign Id values in a client application. Let the SAS Metadata Server assign identifiers to new objects.

Functional Index to IOMI Methods

In this book, IOMI methods are described in alphabetical order. This section categorizes IOMI methods by function.

Table 7.1 *Functional Index to IOMI Methods*

Category	Method	Description
Read Methods	"GetMetadata Method" on page 109	Gets specified properties for a specified metadata object
	"GetMetadataObjects Method" on page 115	Gets all metadata objects of the specified metadata type from the specified repository
Repository Methods	"GetRepositories Method" on page 122	Gets the metadata repositories on the SAS Metadata Server
Write Methods	"AddMetadata Method" on page 91	Adds metadata objects to a repository
	"DeleteMetadata Method" on page 102	Deletes metadata objects from a repository
	"UpdateMetadata Method" on page 139	Updates metadata objects in a repository
Messaging Method	"DoRequest Method" on page 107	Executes XML-formatted method calls
Management Methods	"GetNamespaces Method" on page 120	Gets the namespaces defined on the SAS Metadata Server
	"GetSubtypes Method" on page 128	Gets all possible subtypes for a specified metadata type
	"GetTypes Method" on page 132	Gets all of the metadata types in a namespace
	"GetTypeProperties Method" on page 130	Gets all possible properties for a specified metadata type
	"IsSubtypeOf Method" on page 137	Determines whether one metadata type is a subtype of another metadata type
User Interface Helper Methods	"AddResponsibleParty Method" on page 94	Creates a ResponsibleParty object for the specified identity in the repository that contains the identity's metadata definition
	"AddUserFolders Method" on page 97	Creates a user's home folder and subfolders

Category	Method	Description
	"ChangePassPhrase Method" on page 100	Re-encrypts all passwords in Login and Password objects that are stored in the SAS Metadata Server.
	"GetResponsiblePart y Method" on page 125	Gets the ResponsibleParty object associated with the specified Person or IdentityGroup and responsibility
	"GetUserFolders Method" on page 135	Gets a user's home folder or subfolders

Using IOMI Flags

Purpose of IOMI Flags

Various IOMI methods support flags. The write methods require that an OMI_TRUSTED_CLIENT flag be set to authenticate write operations. Other methods support flags to expand or filter metadata retrieval requests, or to request optional behaviors. See "Summary Table of IOMI Flags" on page 81 for a list of the available flags and the methods for which they are supported.

How to a Specify 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 gets all properties for the specified object, excluding properties for which a value has not been defined.

How to Specify a Corresponding XML Element

Most flags do not require additional input. When a flag does require additional input, 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 search="*criteria*"> element. The GetMetadata method OMI_TEMPLATE flag, which enables you to request additional properties

for metadata objects, requires that you submit a string identifying the additional properties in a <TEMPLATES> element. These additional XML elements are submitted in the OPTIONS parameter. See “Summary Table of IOMI Options” on page 87 for a list of these special XML elements.

Flag Behavior When Multiple Flags Are Used

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

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

Summary Table of IOMI Flags

The following table lists the flags that are supported for the metadata access methods:

Table 7.2 Flags Supported in IOMI Methods

Flag name	Numeric Indicator	Method	Description
OMI_ALL	1	"GetMetadata Method" on page 109 "GetRepositories Method" on page 122 "GetTypeProperties Method" on page 130	In GetMetadata, gets all of the properties of the requested object. This includes all of the attributes that are documented for the requested metadata type in its Attributes table, and all of the associations that are documented in the Associations table, whether they have values stored for them or not. The results include both unique and inherited properties. If the returned XML stream includes references to any associated objects, then GetMetadata returns only general, identifying information for the associated objects. In GetRepositories, gets information about repository location, format, type, and availability in addition to listing the repositories. In GetTypeProperties, gets a description of the supported value for each property.
OMI_ALL_DESCENDANTS	64	"GetSubtypes Method" on page 128	Gets the descendants of the returned subtypes and the subtypes.
OMI_ALL_SIMPLE	8	"GetMetadata Method" on page 109	Gets all of the attributes of the requested object.
OMI_DELETE	32	"DeleteMetadata Method" on page 102	Deletes the contents of a repository and the repository's registration.

Flag name	Numeric Indicator	Method	Description
OMI_DEPENDENCY_USED_BY	16384	"GetMetadata Method" on page 109 "GetMetadataObjects Method" on page 115	When issued in GetMetadata, specifies to include associations to objects that exist in project repositories in the method results. When issued in GetMetadataObjects, specifies to include objects from all project repositories in the method results.
OMI_DEPENDENCY_USES	8192	"GetMetadataObjects Method" on page 115	Specifies to include associations to objects from all production repositories (the foundation repository and all custom repositories) in the method results.
OMI_FULL_OBJECT	2	"GetMetadata Method" on page 109	Instructs the SAS Metadata Server to use a type definition from the SAS type dictionary to expand the object's definition, if a type definition exists.
OMI_GET_METADATA	256	"GetMetadataObjects Method" on page 115	Executes a GetMetadata call for each object that is returned by the GetMetadataObjects method.
OMI_IGNORE_NOTFOUND	134217728	"DeleteMetadata Method" on page 102 "UpdateMetadata Method" on page 139	Prevents a Delete or Update operation from being aborted when a request specifies to delete or update an object that does not exist.

Flag name	Numeric Indicator	Method	Description
OMI_INCLUDE_SUBTYPES	16	"GetMetadata Method" on page 109 "GetMetadataObjects Method" on page 115	Gets specified properties for metadata objects that are subtypes of the specified metadata type and the specified object. In GetMetadata, this flag must be used with at least one template and the OMI_TEMPLATE flag.
OMI_LOCK	32768	"GetMetadata Method" on page 109	Locks the specified object and any associated objects selected by GetMetadata flags and options from update by everyone except the caller.
OMI_MATCH_CASE	512	"GetMetadataObjects Method" on page 115	Performs a case-sensitive search that is based on criteria specified in the <XMLSELECT> element. The OMI_MATCH_CASE flag must be used with the OMI_XMLSELECT flag.
OMI_NOEXPAND_DUPS	524288	"GetMetadata Method" on page 109	Modifies OMI_TEMPLATE and OMI_FULL_OBJECT processing so that objects in a template (user-defined or from a type definition) are expanded only once per primary object specified in the INMETADATA parameter.

Flag name	Numeric Indicator	Method	Description
OMI_NOFORMAT	67108864	“GetMetadata Method” on page 109	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 OMI_NOFORMAT flag, the default US-English locale is used to format the values into recognizable character strings.
OMI_REINIT	2097152	“DeleteMetadata Method” on page 102	Deletes the contents of a repository, but does not remove the repository's registration from the SAS Repository Manager.
OMI_RETURN_LIST	1024	“DeleteMetadata Method” on page 102 “UpdateMetadata Method” on page 139	In DeleteMetadata, returns the identifiers of any dependent objects that were deleted or of any subordinate objects that were deleted in the method output, depending on the method's usage. In UpdateMetadata, returns the identifiers of any dependent objects that were deleted by the Update operation in the method output.

Flag name	Numeric Indicator	Method	Description
OMI_SUCCINCT	2048	"GetMetadata Method" on page 109 "GetTypes Method" on page 132	In GetMetadata, omits all properties that do not contain a value or that contain a null value. In GetTypes, checks the OPTIONS parameter for a <REPOSID> element, and lists the metadata types of objects that exist in the specified repository. For more information, see "Using GetTypes to Get Actual Metadata Types in a Repository" on page 310.
OMI_TEMPLATE	4	"DeleteMetadata Method" on page 102 "GetMetadata Method" on page 109	In DeleteMetadata, checks the OPTIONS parameter for user-defined templates that specify which associated objects to delete with the specified metadata object. In GetMetadata, checks the OPTIONS parameter for user-defined templates that define which metadata properties to return. In both methods, the templates are submitted in a <TEMPLATES> element. For more information, see Chapter 15, "Using Templates," on page 337.
OMI_TRUNCATE	4194304	"DeleteMetadata Method" on page 102	Deletes all metadata objects, but does not delete the metadata object containers from a repository or remove the repository's registration from the SAS Repository Manager.

Flag name	Numeric Indicator	Method	Description
OMI_TRUSTED_CLIENT	268435456	"AddMetadata Method" on page 91 "DeleteMetadata Method" on page 102 "UpdateMetadata Method" on page 139	Determines whether the client can call this method.
OMI_UNLOCK	131072	"GetMetadata Method" on page 109 "UpdateMetadata Method" on page 139	Unlocks an object lock that is held by the caller.
OMI_UNLOCK_FORCE	262144	"GetMetadata Method" on page 109 "UpdateMetadata Method" on page 139	Unlocks an object lock that is held by another user.
OMI_XMLSELECT	128	"GetMetadataObjects Method" on page 115	Checks the OPTIONS parameter for search criteria that filter the objects that are returned. The search criteria are passed as a search string in an <XMLSELECT> element. For more information, see Chapter 17, "Filtering a GetMetadataObjects Request," on page 371.

Summary Table of IOMI Options

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

Table 7.3 Options Supported for IOMI Methods

Option	Flag	Method	Description
<DOAS>	None	"AddMetadata Method" on page 91 "DeleteMetadata Method" on page 102 "GetMetadata Method" on page 109 "GetMetadataObjects Method" on page 115 "GetSubtypes Method" on page 128 "GetTypeProperties Method" on page 130 "IsSubtypeOf Method" on page 137 "UpdateMetadata Method" on page 139	Enables a client to make a request on behalf of another user. For more information, see "<DOAS> Option" on page 89.
<REPOSID>	OMI_SUCCINCT (2048)	"GetTypes Method" on page 132	Specifies the repository ID of the repository whose metadata you want to evaluate. For more information, see "Using GetTypes to Get Actual Metadata Types in a Repository" on page 310.

Option	Flag	Method	Description
<TEMPLATES>	OMI_TEMPLATE (4)	"DeleteMetadata Method" on page 102 "GetMetadata Method" on page 109	In DeleteMetadata, submits a <TEMPLATE> element that contains a property string that specifies associations to delete with the specified metadata object. In GetMetadata, submits property strings that specify properties to get for the specified metadata object in addition to those specified in the INMETADATA parameter and by GetMetadata flags. See "Understanding Templates" on page 337.
<XMLSELECT>	OMI_XMLSELECT (128) and OMI_MATCH_CASE (512)	"GetMetadataObjects Method" on page 115	Specifies a search string to filter the objects that are retrieved. For more information, see Chapter 17, "Filtering a GetMetadataObjects Request," on page 371.

<DOAS> Option

About the <DOAS> Option

IOMI methods support a <DOAS> element in the OPTIONS parameter that enables SAS Open Metadata Interface clients to make a metadata request for another user. Typically, when a metadata request is made, the authorization facility checks the

user ID and credentials of the requesting user to determine whether the request is allowed. The <DOAS> element permits the request to be made with 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 set also contains references to any IdentityGroup objects in which the principal identity is either directly or indirectly identified as a member.

The <DOAS> element enables middleware servers to use the identity of their own clients when making metadata requests. This way, the request is authorized based on the credentials of the client, rather than basing it on the credentials of the connecting user. That is, when the <DOAS> element is encountered, metadata is created, returned, and updated based on the credentials of the specified client, rather than the connecting user. It is the responsibility of the client to authenticate the user.

Specifying the <DOAS> Option

The <DOAS> element 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"/>
```

CredHandle is a handle that is returned by the ISecurity GetCredentials method that represents the other user's credentials. For more information, see "GetCredentials Method" on page 164.

A client must have trusted user status on the SAS Metadata Server to issue the ISecurity GetCredentials method. A trusted user is a special user whose user ID is defined in the trustedUsers.txt file.

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 returns only PhysicalTable objects that 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 the 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 is created only if the user who is identified in the credential handle has WriteMetadata permission to the specified repository.

AddMetadata Method

Short Description

Adds metadata objects to a repository.

Category

IOMI interface write methods

Syntax

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

Parameters

Table 7.4 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
inMetadata	C	in	Metadata property string that defines the object to be added.
reposid	C	in	Target repository ID.
outMetadata	C	out	Returned metadata property string that includes the object as a result of the add operation.
ns	C	in	Namespace to use as the context for the request.
flags	L	in	OMI_TRUSTED_CLIENT=268435456 Determines whether the client can call this method. This flag is required.
options	C	in	Passed indicator for options. <CREATEREPOSCONTAINER/> When AddMetadata is issued in the REPOS namespace to create a RepositoryBase object, this option creates the physical directory specified in the object's Path attribute, if the physical directory does not exist. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see “<DOAS> Option” on page 89.

Details

The AddMetadata method creates metadata objects. It is used to create both the metadata object that defines a repository, and the metadata objects that are within the repository. To update an existing metadata object, whether it defines a repository or a metadata object within the repository, use the UpdateMetadata method.

The INMETADATA parameter specifies a metadata property string that defines the properties to be added for the object. A request that creates a repository defines an object of the RepositoryBase metadata type and is issued in the REPOS namespace. A request that adds an object to a repository is issued in the SAS namespace and defines SAS namespace metadata types. Not all metadata types or their properties can be added. See the documentation for each metadata type. AddMetadata returns an error for any metadata type that cannot be added.

An AddMetadata request that creates the metadata object that defines a repository does not automatically create the physical directory specified in the Path attribute. You must create the physical directory specified in the Path attribute in advance. Or, you can pass the <CREATEREPOSCONTAINER/> element in the AddMetadata request to create the physical directory. The Path attribute accepts an absolute or relative pathname. Use backslashes or forward slashes (\ and /) to indicate the directory levels. <CREATEREPOSCONTAINER/> creates one directory, and it will be the last directory in the specified path. If the other directories in the path do not exist, the SAS Metadata Server returns an error.

The OUTMETADATA parameter mirrors the content of the INMETADATA parameter. In addition, it returns identifiers for the requested objects. Any invalid properties in the INMETADATA metadata property string remain in the OUTMETADATA metadata property string. For information about the structure of the metadata property string, see “Constructing a Metadata Property String” on page 76.

The AddMetadata method can be used to create an object only, to create an object and an association to an existing object, or to create an object, an association, and the associated object. Associations to objects can be made in the same repository or in a different repository. The attributes defining the objects indicate the type of operation to be performed. For more information, see Chapter 11, “Adding Metadata Objects,” on page 277.

Objects and associations are created subject to security constraints. For example, a requestor must have administrative status on the SAS Metadata Server to add a repository. A requestor must have WriteMetadata permission to a repository to add an object to the repository. When creating an association between a new object and an existing object, the requestor must have WriteMetadata permission either to the existing object, or to the repository in which the existing object resides.

The SAS Metadata Server assigns object identifiers after the successful completion of an AddMetadata request.

Check the return code of an AddMetadata method call. A nonzero return code indicates that a failure occurred while trying to write the metadata. A nonzero return code means none of the changes in the method call were made.

Example 1: Standard Interface

The following is an example of how to issue the AddMetadata method regardless of the programming environment. The request adds a new PhysicalTable object and provides the Name and Desc values.

```
<!-- Create a metadata list to be passed to AddMetadata method  -->
inMetadata = "<PhysicalTable Name="NECust"
               Desc="All customers in the northeast region"/>";

reposid= "A0000001.A2345678";
ns= "SAS";
```

```
<!-- OMI_TRUSTED_CLIENT flag -->
flags= 268435456;
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 is an example of an XML string that shows how to format the method call in example 1 for the INMETADATA parameter of the DoRequest method. A <METADATA> element, rather than an <INMETADATA> element, specifies the passed metadata 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

- “UpdateMetadata Method” on page 139
- “GetRepositories Method” on page 122

AddResponsibleParty Method

Short Description

Creates a ResponsibleParty object for the specified identity in the repository that contains the identity’s metadata definition.

Category

IOMI interface user interface helper methods

Syntax

```
rc=DoRequest("<AddResponsibleParty>...</AddResponsibleParty>",outMetadata);
```

Parameters

Table 7.5 *Method Parameters*

Parameter	Type	Direction	Description
<ResponsibleParty />	C	in	Metadata property string that creates a ResponsibleParty object. See the “Details” section for information about the format of the metadata property string.

Details

SAS Management Console and SAS Data Integration Studio allow users to define a set of responsibilities for an object. These product’s frameworks support two types of responsibilities — Owner and Administrator. The AddResponsibleParty method enables a user to easily create a ResponsibleParty object. (For example, a ResponsibleParty object can be created that defines a user named “John Smith” as the owner of a particular stored process.) A ResponsibleParty object can be defined for any object in metadata.

ResponsibleParty objects must be created in the same SAS Metadata Repository as the metadata definition of the identity that they describe. Metadata definitions for individual users (Person objects) are always created in the foundation repository. Metadata definitions for groups (IdentityGroup objects) can be created in the foundation repository or a custom repository. The AddMetadata method, which is provided to add objects to a repository, requires a requestor to have WriteMetadata permission to a repository to create an object. The AddResponsibleParty method is provided to allow users to create ResponsibleParty objects in the appropriate repository, even if they do not have WriteMetadata permission to that repository.

The AddResponsibleParty method is available in the DoRequest interface only. The method and its parameters are specified in an XML input string within the

INMETADATA parameter of the DoRequest method. The method's output is returned in the DoRequest method's OUTMETADATA parameter.

The XML input string consists of an <ADDRESPONSIBLEPARTY> element that passes a metadata property string that defines a ResponsibleParty object in the following form:

```
<ResponsibleParty IdentityName='name' Responsibility='role'/>
```

IDENTITYNAME='*name*'
: Specifies the name, up to 60 characters, of a Person or IdentityGroup object that is defined on the SAS Metadata Server. The SAS Metadata Server normalizes the value before storing it in the ResponsibleParty object's Name attribute. A null value implies the connected user. If the specified identity or connected user is Public, an error is returned. If the specified identity is not found in the SAS Metadata Server, an error stating that the object was not found is returned.

RESPONSIBILITY='*role*'
: Specifies a value, up to 100 characters, that is valid for the client. If the RESPONSIBILITY parameter is omitted or passes a null value, the SAS Metadata Server returns an error.

If you enter a value that is greater than the maximum character length for either the IDENTITYNAME or RESPONSIBILITY parameters, the value is truncated.

Before creating the requested ResponsibleParty object, the AddResponsibleParty method verifies that an object does not exist that meets the criteria. This causes additional locks on the repository, so the AddResponsibleParty method should be called by a client only after verifying the need to add an object with the GetResponsibleParty method.

The output of the AddResponsibleParty method mirrors the input, except the object identifier of the new object is returned.

Example

The following is an example of a DoRequest method call that issues an AddResponsibleParty request.

```
outMetadata=""
inMetadata =
"<AddResponsibleParty>
   <ResponsibleParty IdentityName=' ' Responsibility='Owner'/>
</AddResponsibleParty>";

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

In this example, which does not specify an IDENTITYNAME value, the ResponsibleParty object is created for the caller.

Related Methods

- "DoRequest Method" on page 107
- "GetResponsibleParty Method" on page 125

AddUserFolders Method

Short Description

Creates a user's home folder and subfolders.

Category

IOMI interface user interface helper methods

Syntax

```
rc=DoRequest("<AddUserFolders>...</AddUserFolders>",outMetadata);
```

Parameters

Table 7.6 *Method Parameters*

Parameter	Type	Direction	Description
<Tree/>	C	in	Metadata property string that creates a Tree object. See the “Details” section for information about the format of the metadata property string.

Details

The SAS Metadata Server supports the concept of a user folder to enable clients to provide a consistent user interface to metadata. The user folder is a work area similar to the My Documents area on a Windows system. Metadata that is created or accessed by a user is stored in a subfolder of the user folder. This subfolder is named “My Folder” by default. The work area also includes a subfolder named “Application Data” that stores system information about the user for the internal use of applications.

The AddUserFolders method can be used to create one or all of these folders for a specified user.

The AddUserFolders method is available in the DoRequest interface only. The method and its parameters are specified in an XML input string within the INMETADATA parameter of the DoRequest method. The method's output is returned in the DoRequest method's OUTMETADATA parameter.

The XML input string consists of an <ADDUSERFOLDERS> element that passes a metadata property string that defines a Tree object in the following form:

```
<Tree PersonName='name' FolderName='folder-type'/>
```

A folder is represented by the Tree metadata type in a SAS Metadata Repository.

PERSONNAME='*name*'
Specifies the name of the user for whom the folder is created.

The PERSONNAME value must be the unique name stored in a Person object's Name attribute or be blank. If a name value contains a forward slash (/) or backslash (\), the AddUserFolders method changes it to a dash (-) so that it does not interfere with the folder's pathname specification. When PERSONNAME is blank, the specified folder is created for the connected user. A user folder cannot be created for an IdentityGroup. If the name specified in PERSONNAME does not match the name of the requesting user, the connected user must be an administrative user of the SAS Metadata Server. Otherwise, the server returns an error. For more information about administrative user status on the SAS Metadata Server, see *SAS Intelligence Platform: Security Administration Guide*.

FOLDERNAME='*folder-type*'
Specifies the type of user folder to create. Valid values are “Home Folder,” “My Folder,” or “Application Data.”

When choosing a FOLDERNAME value for an AddUserFolders request, note that a request to create a folder named “Home Folder” does not also create subfolders. However, a request to create any of the subfolders (“My Folder” or “Application Data”) does create a home folder, if one does not exist.

If you specify a SAS Metadata Model metadata type other than Tree in the XML input string, the SAS Metadata Server returns an error. The name “Folder” is accepted in the XML input string, because Folder is the PublicType value that is assigned to a Tree object.

Do not specify more than one Tree or Folder definition within the <ADDUSERFOLDERS> element. If you want to define more than one user folder in a request, submit multiple <ADDUSERFOLDERS> elements within a <MULTIPLE_REQUESTS> element. For more information about the <MULTIPLE_REQUESTS> element, see “DoRequest Method” on page 107.

The AddUserFolders method uses the security policy defined for the foundation repository to determine where to create the folders. The foundation repository has a Metadata Location for Users' Folders policy that specifies the root folder in which to store the user folders. The default security policy is to create the user folders in the foundation repository in a /User Folders folder. However, the SAS Metadata Server supports storing folders in another folder of the foundation repository or in another repository, if the other repository and folder exist in the folder tree. The AddUserFolders method does not create a repository or /User Folders folder for you.

A successful AddUserFolders request creates a subfolder in the /User Folders folder that has the name specified in the PERSONNAME parameter. The request potentially creates one or both of the “My Folder” and “Application Data” subfolders.

The names “My Folder” and “Application Data” are localized. For example, if all possible folders were created for a user using the US-English locale, they would be displayed in the folder tree as follows:

- —Users
  - —*PersonName*
    - —My Folder
    - —Application Data

If the French locale were active, then “My Folder” and “Application Data” would be displayed in French. The locale used to create the Tree object's DisplayName attribute is provided to the SAS Metadata Server in the LOCALE server invocation option or in the sasv9.cfg file.

The AddUserFolders method automatically stores the following attribute values for each folder:

- PublicType="Folder"
- UsageVersion="1000000"
- TreeType="BIP Folder"
- DisplayName="*server-localized-version-of-folder-name*"

  This attribute is set only for subfolders of the user folder.

The Admin-Only Update ACT grants SAS Metadata Server administrators WriteMetadata and Write permissions to all user home folders, and denies the Public group WriteMetadata and Write permissions to user home folders. This enables only administrators to create and update home folders. The Private User Folder ACT grants SAS Metadata Server administrators full access to all "My Folder" and "Application Data" subfolders in the directory, and denies the Public group access to the subfolders. This ACT contains an access control entry (ACE) for each person indicated in PERSONNAME that grants them ReadMetadata, WriteMemberMetadata, and CheckinMetadata permissions on his or her “My Folder” and “Application Data” folders. These permissions enable the folder owners to view and create metadata in their user folders, but not to delete the folders.

If the administrator changes the Metadata Location for Users' Folders policy after some user folders have been created, then any home folders and subfolders for existing users in /User Folders will remain in this folder. Home folders for new users are created in the new location.

Example

The following is an example of a DoRequest method call that issues an AddUserFolders request. The request creates both a “My Folder” and home folder for user “SAS Web Administrator.”

```
outMetadata=""
inMetadata=
"<AddUserFolders>
  <Tree PersonName='SAS Web Administrator' FolderName='My Folder'/>
</AddUserFolders>";

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

If the request is successful, the XML returned in the OUTMETADATA parameter mirrors the input in the INMETADATA parameter. The output includes the 17-character metadata identifier of the newly created “My Folder” Tree object. A metadata property string and an Id value defining the home folder are not returned in the output. The folder is created if it did not exist.

The request is rejected with an authorization error if the requesting user is not “SAS Web Administrator,” is not an administrative user of the SAS Metadata Server, or if “SAS Web Administrator” is the name of an IdentityGroup.

Related Methods

- “DoRequest Method” on page 107
- “GetUserFolders Method” on page 135

ChangePassPhrase Method

Short Description

Re-encrypts all passwords in Login and SASPassword objects that are stored in the SAS Metadata Server.

Category

IOMI interface user interface helper methods

Syntax

```
rc=DoRequest("<ChangePassPhrase>...</ChangePassPhrase>",outMetadata);
```

Parameters

Table 7.7 Method Parameters

Parameter	Type	Direction	Description
<PassPhrase>*Value* </PassPhrase>	C	in	XML element that specifies a new pass phrase. See the "Details" section for information about the pass phrase.

Details

The ChangePassPhrase method enables an authorized user to re-encrypt all passwords in Login and SASPassword objects that are stored in the SAS Metadata Server. The default encryption algorithm for storing passwords on the metadata server is proprietary and uses a fixed key. When SAS/SECURE is used, additional algorithms are available. The strongest available algorithm is used by default when encrypting passwords.

When SAS/SECURE is used, a site can specify a pass phrase to use as a key for the encryption. If passwords are compromised, they can be made safe again by simply changing the pass phrase.

The ChangePassPhrase method updates the passwords in all repositories in the server. It cannot be applied to a single repository. The pass phrase will be the same for all repositories that are updated. Repositories that are offline, Read only, restored, unregistered and re-registered, or otherwise not available can have a different pass phrase. A repository that is using a different pass phrase from the other servers will log a message to the server log and continue to be accessible. At some point, the ChangePassPhrase method can be executed again to have all repositories use the same pass phrase.

An administrator must have unrestricted authorization to run ChangePassPhrase. The server must be paused in an ADMIN state before ChangePassPhrase is executed. When ChangePassPhrase has completed, the server can be resumed.

The method is available in the DoRequest method only. It takes one <PASSPHRASE>*value*</PASSPHRASE> XML element as a parameter. The specified pass phrase can be 0 to 1,016 characters. The pass phrase value can be clear text, encrypted, or blank. Encrypt the pass phrase with the PWENCODE procedure. Specifying a blank pass phrase will encrypt the password using a fixed-key pass phrase.

The ChangePassPhrase method writes messages to the server log. This log should be inspected after the method completes to verify the success of each repository conversion. A repository will not be processed if the server is not paused in an ADMIN state.

The following logger can be specified in the metadata server's log configuration file (logconfig.xml in most installations) to identify the objects whose Login objects have changed:

```
<logger name="Audit.Meta.Updates.PublicObjects">
      <level value="Trace"/>
 </logger>
 <logger name="Audit.Meta.Updates.Projects.PublicObjects">
      <level value="Debug"/>
 </logger>
```

Object log messages are written after the objects are committed to the repository.

Example

The following is an example of a DoRequest method call that issues a ChangePassPhrase request. The request assigns the pass phrase, "This is my pass phrase." to all SASPassword and Login objects on the server.

```
outMetadata=""
inMetadata=
"<ChangePassPhrase>
  <PassPhrase>This is my pass phrase.</PassPhrase>
</ChangePassPhrase>";

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

DeleteMetadata Method

Short Description

Deletes metadata objects from a repository.

Category

IOMI interface write methods

Syntax

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

Parameters

Table 7.8 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
inMetadata	C	in	Metadata property string that identifies the object to be deleted.
outMetadata	C	out	Returned metadata property string that includes the results of the Delete operation. The outMetadata parameter is used only if OMI_RETURN_LIST is specified.
ns	C	in	Namespace to use as the context for the request.

Parameter	Type	Direction	Description
flags	L	in	OMI_DELETE=32 Valid in the REPOS namespace only. Specifies to delete the contents of a repository and the repository's registration. OMI_IGNORE_NOTFOUND=134217728 Prevents a Delete operation from being canceled when a request specifies to delete an object that does not exist. OMI_REINIT=2097152 Valid in the REPOS namespace only. Specifies to delete the contents of a repository, but does not remove the repository's registration from the SAS Repository Manager. OMI_RETURN_LIST=1024 Specifies to return the identifiers of any dependent objects that were deleted, or of any subordinate objects that were deleted. OMI_TEMPLATE=4 Valid in the SAS namespace only. Checks the OPTIONS parameter for user-defined templates that specify associated objects to delete with the specified metadata object. The templates are passed in a <TEMPLATES> element in the OPTIONS parameter. OMI_TRUNCATE=4194304 Valid in the REPOS namespace only. Specifies to delete all metadata objects, but does not delete the metadata object containers from a repository, or remove the repository's registration. OMI_TRUSTED_CLIENT=268435456 Determines whether the client can call this method. This flag is required.

Parameter	Type	Direction	Description
options	C	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see "<DOAS> Option" on page 89. <TEMPLATES> Submits templates that identify associated objects to delete with the specified metadata objects. Each template is submitted in a <TEMPLATE> element within the <TEMPLATES> element. The <TEMPLATES> option must be specified with the OMI_TEMPLATE flag.

Details

The DeleteMetadata method deletes metadata objects from a repository. To replace or modify the properties of a metadata object, use the UpdateMetadata method.

The DeleteMetadata method is typically issued in the SAS namespace to delete metadata representing application elements. The method can also be issued in the REPOS namespace on a RepositoryBase object to unregister the repository, to destroy the repository, or to clear all objects from the repository without harming the repository's registration. Flags that are valid only in the REPOS namespace are provided to perform these tasks. For more information, see "Deleting a Repository" on page 400. You must have administrative status on the SAS Metadata Server to issue the DeleteMetadata method in the REPOS namespace. For more information about administrative user status, see the *SAS Intelligence Platform: Security Administration Guide*.

Regardless of the namespace in which it is issued (REPOS or SAS), a DeleteMetadata method call must set the OMI_TRUSTED_CLIENT flag (268435456). The OMI_TRUSTED_CLIENT flag is required in all method calls that write or remove metadata.

The object to delete is primarily identified in a metadata property string that is submitted to the method in the INMETADATA parameter. To delete multiple objects, stack their metadata property strings in the INMETADATA parameter.

In addition to deleting specified SAS Metadata Model objects, a DeleteMetadata method issued in the SAS namespace deletes associated objects using a type definition from the SAS type dictionary, or, when the OMI_TEMPLATE flag is set, associated objects that are specified in a template. For usage information, see Chapter 19, "Deleting Metadata Objects," on page 395.

Check the return code of a DeleteMetadata method call. A nonzero return code indicates that a failure occurred while trying to delete the metadata objects. A nonzero return code means none of the changes indicated by the method call were made.

Example 1: Standard Interface

The following is an example of how to issue the DeleteMetadata method regardless of the programming environment. The request deletes a SASLibrary object. When a SASLibrary object is deleted, any object in the library is deleted as well. The OMI_RETURN_LIST flag is specified (268435456 + 1024 =268436480) so the OUTMETADATA parameter returns the identifiers of all deleted objects.

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

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

Example 2: DoRequest Method

The following is an example of an XML string that shows how to format the method call in example 1 for the INMETADATA parameter of the DoRequest method.

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

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

Related Methods

- “AddMetadata Method” on page 91
- “UpdateMetadata Method” on page 139

DoRequest Method

Short Description

Executes XML-formatted method calls.

Category

IOMI interface messaging method

Syntax

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

Parameters

Table 7.9 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
inMetadata	C	in	XML string that contains a method to execute and the parameters for the method.
outMetadata	C	out	Returned metadata property string that includes the results of the method.

Details

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

```
<MethodName>
   <Parameter1>Value</Parameter1>
   <Parameter2>Value</Parameter2>
   <Parametern>Value</Parametern>
</MethodName>
```

where <METHODNAME> is an XML element that contains the name of an IOMI method. <PARAMETER 1–*n*> are XML elements that contain the names of the method's parameters.

Multiple methods can be submitted in one DoRequest request by placing them within a <MULTIPLE_REQUESTS> element and stacking the XML method strings. 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 method parameter names must be used for all method parameters except INMETADATA. A <METADATA> element must be used to represent the INMETADATA parameter within method calls that support this parameter. For other parameters, the method returns an error if parameter names other than the published names are used. For more information about the format of this method string, see the documentation for the method that you want to execute.

You submit the XML-formatted method calls to the SAS Metadata Server in the DoRequest method's INMETADATA parameter. The XML-formatted method calls are 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, such as those enclosing XML attribute values in the metadata property string, be marked to indicate that they should be treated as characters. Here are examples of using escape characters in different programming environments to mark the additional double quotation marks:

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

Any metadata-related (IOMI server interface) method can be submitted to the SAS Metadata Server using the DoRequest method. For information about the exact format of a method request, see the documentation for the method that you want to execute.

The DoRequest method supports requests to metadata objects in both the SAS namespace and the REPOS namespace. Requests that call the SAS and REPOS namespaces can be submitted within the same <MULTIPLE_REQUESTS> element.

The DoRequest method is ACID-compliant. ACID (Atomicity, Consistency, Isolation, Durability) is a term that refers to the guarantee that all of the tasks of a transaction are performed or none of them are. In other words, if multiple methods are submitted, and one method in a DoRequest fails, then all of the methods specified in the XML input string fail.

The DoRequest method's OUTMETADATA string mirrors the INMETADATA string, except requested values are provided.

Check the return code of a DoRequest method call. A nonzero return code indicates that a failure occurred while trying to write metadata. A nonzero return code means none of the changes in any of the methods in the DoRequest were made.

Example

The DoRequest method is issued in the standard interface. The following is an example of how to issue a DoRequest method call regardless of the programming environment.

```
outMetadata=" ";
inMetadata="XML-method-string";

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

GetMetadata Method

Short Description

Gets specified attributes and associations for the specified metadata object.

Category

IOMI interface read methods

Syntax

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

Parameters

Table 7.10 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
inMetadata	C	in	Metadata property string that identifies the object and attributes and associations to be read.
outMetadata	C	out	Returned metadata property string that includes the results of the Read operation.
ns	C	in	Namespace to use as the context for the request.

Parameter	Type	Direction	Description
flags	L	in	OMI_ALL=1 Specifies to get all of the attributes that are documented for the requested metadata type in its Attributes table, and all of the associations that are documented in its Associations table, whether they have values stored for them or not. The results include both unique and inherited attributes and associations. If the returned XML stream includes references to any associated objects, GetMetadata returns only general, identifying information for the associated objects. OMI_ALL_SIMPLE=8 Specifies to get all of the attributes of the requested object. If the request returns associated objects, it gets all attributes for those objects as well. The results include both unique and inherited attributes. OMI_DEPENDENCY_USED_BY=16384 Specifies to include associations to objects that exist in all project repositories in the method results. OMI_FULL_OBJECT= 2 Instructs the SAS Metadata Server to use a type definition from the SAS type dictionary to expand the specified object's metadata definition. The server reads the values in the object's PublicType and UsageVersion attributes to determine which type definition to use to process the request. OMI_INCLUDE_SUBTYPES=16 Specifies to get the specified properties for metadata objects that are subtypes of the specified metadata type, in addition to the specified metadata object. The OMI_INCLUDE_SUBTYPES flag must be set with the OMI_TEMPLATE flag and a template, or the flag is ignored.

Parameter	Type	Direction	Description
			OMI_LOCK=32768 Locks the specified object and any associated objects selected by GetMetadata flags and options from update by everyone except the caller. OMI_NOEXPAND_DUPS=524288 Modifies OMI_TEMPLATE and OMI_FULL_OBJECT processing so that objects are expanded by the associated template (user-defined or from the type dictionary) only once per primary object specified in the INMETADATA 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 OMI_NOFORMAT flag, the default US-English locale is used to format the values into recognizable character strings. OMI_SUCCINCT=2048 Specifies to omit attributes and associations that do not contain a value or that contain a null value from the returned XML string. OMI_TEMPLATE=4 Checks the OPTIONS parameter for user-defined templates that specify which metadata properties to return. The user-specified templates are submitted in a <TEMPLATES> element in the OPTIONS parameter. OMI_UNLOCK=131072 Unlocks an object lock that is held by the caller. OMI_UNLOCK_FORCE=262144 Unlocks an object lock that is held by another user.

Parameter	Type	Direction	Description
options	C	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see "<DOAS> Option" on page 89. <TEMPLATES> There are two ways to specify templates. You can specify a metadata type and properties to return directly in the <TEMPLATES> element. Or, you can specify multiple templates, each contained within a <TEMPLATE> subelement in the <TEMPLATES> element. Either way, the <TEMPLATES> element must be specified with the OMI_TEMPLATE flag. For more information, see Chapter 15, "Using Templates," on page 337.

Details

The GetMetadata method gets specified attributes and associations for the specified SAS Metadata Model metadata object.

The method's default behavior is to get attributes and associations that are specified in the INMETADATA parameter of the method call. As an alternative, you can set one or more flags that specify to get all attributes or all attributes and associations. Or you can specify a flag (OMI_FULL_OBJECT) that instructs the SAS Metadata Server to use a type definition from the SAS type dictionary to identify the association names to expand when getting the specified object.

The OMI_FULL_OBJECT flag is effective only if the specified object is a PrimaryType subtype in the SAS Metadata Model, and only if it stores the name of a type definition in the PublicType attribute. The type definition internalizes information about the associations that will be retrieved so that clients do not need to know the details. The primary metadata type and associations that describe a resource in the SAS Metadata Repository are referred to as the resource's logical metadata definition. For more information, see Chapter 14, "Getting the Properties of a Specified Metadata Object," on page 315, "GetMetadata and Logical Type Definitions" on page 316, and "Using the OMI_FULL_OBJECT Flag" on page 334.

The GetMetadata method has no concept of a logical metadata definition unless you set the OMI_FULL_OBJECT flag, or you specify a user-defined template that specifies association names to expand with the specified object. The GetMetadata method supports two template forms. For more information, see"Understanding Templates" on page 337 and "Creating Templates for the Get Methods" on page 340.

The OMI_UNLOCK and OMI_UNLOCK_FORCE flags release object locks set with the OMI_LOCK flag.

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

A GetMetadata method that requests associated objects and that is issued in a public repository (the foundation or a custom repository) returns associated objects from all public repositories. If you want to include associated objects that are in project repositories in a public GetMetadata request, set the OMI_DEPENDENCY_USED_BY flag.

The OMI_INCLUDE_SUBTYPES flag extends processing of user-defined templates to include associated metadata objects that are subtypes of the specified metadata object. This functionality is useful when you want to retrieve a common set of properties for multiple SAS Metadata Model metadata objects. For more information, see "Using GetMetadata to Get Common Properties for Sets of Objects" on page 328.

For more information about specifying GetMetadata flags, see "Using IOMI Flags" on page 80. For information about interdependencies between GetMetadata flags, see "Combining GetMetadata Flags" on page 334.

Example 1: Standard Interface

The following is an example of how to issue a GetMetadata method regardless of the programming environment. The request gets the Name, Description, and Column values of the PhysicalTable with an Id value of A5345678.A5000001.

```
<!-- Create a metadata list to be passed to GetMetadata method -->

inMetadata= "<PhysicalTable Id="A5345678.A5000001" Name="" Desc="">
              <Columns/>
            </PhysicalTable>";
ns="SAS";
flags=0;
options="";

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

<!-- outMetadata XML string returned -->
<PhysicalTable Id="A5345678.A5000001" Name="New Table" Desc="New Table added
through API">
  <Columns>
    <Column Id="A5345678.A3000001" Name="New Column" Desc="New Column added
through API"/>
    <Column Id="A5345678.A3000002" Name="New Column2" Desc="New Column2 added
through API"/>
  </Columns>
</PhysicalTable>
```

Example 2: DoRequest Method

The following is an example of an XML string that 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 -->

<GetMetadata>
  <Metadata>
    <PhysicalTable Id="A5345678.A500001" Name="" Desc="">
       <Columns/>
    </PhysicalTable>
  </Metadata>
  <NS>SAS</NS>
  <Flags>0</Flags>
  <Options/>
</GetMetadata>
```

Related Methods

- “GetMetadataObjects Method” on page 115

GetMetadataObjects Method

Short Description

Gets all metadata objects of the specified metadata type in the specified repository.

Category

IOMI interface read methods

Syntax

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

Parameters

Table 7.11 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
reposid	C	in	Target repository identifier.
type	C	in	Metadata type name.
objects	C	out	Returned list of metadata objects.
ns	C	in	Namespace to use as the context for the request.

Parameter	Type	Direction	Description
flags	L	in	OMI_DEPENDENCY_USED_BY=16384 Specifies to include objects from all project repositories in the method results. OMI_DEPENDENCY_USES=8192 Specifies to include objects from all public repositories (the foundation and custom repositories) in the method results. OMI_GET_METADATA=256 Specifies to execute a GetMetadata call for each object that is returned by the GetMetadataObjects request. OMI_INCLUDE_SUBTYPES=16 Specifies to get all of the metadata objects that are subtypes of the specified metadata type and metadata objects of the specified metadata type. If OMI_XMLSELECT is specified, it affects the subtypes that are retrieved. OMI_MATCH_CASE=512 Specifies to perform a case-sensitive search that is based on criteria specified in the <XMLSELECT> element. The OMI_MATCH_CASE flag must be used with the OMI_XMLSELECT flag, or the flag is ignored. OMI_XMLSELECT=128 Specifies to check the OPTIONS parameter for search criteria that filters the objects that are returned. The search criteria are passed as a search string in an <XMLSELECT> element.
options	C	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see “<DOAS> Option” on page 89. <XMLSELECT> Specifies a search string to filter the objects that are retrieved. For usage information, see Chapter 17, “Filtering a GetMetadataObjects Request,” on page 371.

Details

The GetMetadataObjects method gets a list of all metadata objects of the metadata type specified in the TYPE parameter from the repository specified in the REPOSID parameter. The default behavior is to get "Identifying Metadata" for each metadata object.

Flags enable you to get additional properties and to expand or filter the objects that are retrieved.

- OMI_INCLUDE_SUBTYPES expands the request to get subtypes of the specified metadata type.
- OMI_GET_METADATA enables you to execute a GetMetadata call for each object that is returned by the GetMetadataObjects request.
- The OMI_DEPENDENCY_USES and OMI_DEPENDENCY_USED_BY flags specify additional repositories from which to get objects.
- The OMI_XMLSELECT flag and <XMLSELECT> element enable you to filter the objects that are returned by specifying search criteria.

For usage information, see "Introduction to the GetMetadataObjects Method " on page 355.

The default behavior of the GetMetadataObjects method is to get objects of the specified metadata type from the specified repository. Set OMI_DEPENDENCY_USES to get metadata objects from all public repositories (the foundation and all custom repositories) in the method results, and to get metadata objects from the specified repository. Set OMI_DEPENDENCY_USED_BY only if you want to get metadata objects of the specified metadata type from all project repositories in the method results, in addition to metadata objects from the specified repository. Setting both flags will return metadata objects from all repositories that are registered in the SAS Metadata Server (foundation, custom, and project).

When the GetMetadataObjects method is issued in the SAS namespace, the REPOSID parameter is required, unless the OMI_DEPENDENCY_USED_BY flag, the OMI_DEPENDENCY_USES flag, or both is specified. When you specify a REPOSID value in addition to one or both of the flags, GetMetadataObjects gets metadata objects first from the repository specified in the REPOSID parameter, and then it gets metadata objects from the repositories specified by the flags. A request that specifies to get objects from all registered repositories returns the specified repository first, followed by the foundation repository, followed by custom repositories in the order in which they were registered, followed by project repositories in the order in which they were registered.

When the GetMetadataObjects method is issued in the REPOS namespace, it ignores the REPOSID parameter and searches the SAS Repository Manager.

When using GetMetadataObjects to list common and shared objects, use the metadata type and TypeName value indicated in the object's type definition in the SAS type dictionary to identify the objects to retrieve. For more information about the SAS type dictionary, see Chapter 3, "Using Interfaces That Read and Write Metadata in SAS 9.4," on page 19. Specify the metadata type in the TYPE parameter, set the OMI_XMLSELECT flag, and specify the TypeName value in the <XMLSELECT> element in the OPTIONS parameter as follows:

```
<XMLSELECT search="@PublicType='typename'"/>
```

Several common and shared objects are represented in the SAS Metadata Repository by the same metadata type. Use of the TypeName value filters the request to return only objects of the specified type.

Use the new GetMetadata OMI_FULL_OBJECT flag and GetMetadataObjects OMI_GET_METADATA flag with caution. The flags can return a lot of information.

New template features that are available in GetMetadata are also available in GetMetadataObjects when you set the OMI_GET_METADATA flag and the OMI_TEMPLATE flag in a GetMetadataObjects request. For more information, see Chapter 15, “Using Templates,” on page 337.

Example 1: Standard Interface

The following is an example of how to issue a GetMetadataObjects method regardless of the programming environment. The request gets all objects defined for metadata type PhysicalTable in repository A0000001.A5345678. It does not set any flags.

```
<!-- set repository Id and type -->
reposid="A0000001.A5345678";
type="PhysicalTable";
ns="SAS";
flags=0;
options="";

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

<!-- XML string returned in objects parameter -->

<Objects>
  <PhysicalTable Id="A5345678.A5000001" Name="New Table"/>
  <PhysicalTable Id="A5345678.A5000002" Name="New Table2"/>
</Objects>
```

Example 2: DoRequest Method

The following is an example of an XML string that 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 -->
<GetMetadataObjects>
  <Reposid>A0000001.A5345678</Reposid>
  <Type>PhysicalTable</Type>
  <Objects/>
  <NS>SAS</NS>
  <Flags>0</Flags>
  <Options/>
</GetMetadataObjects>
```

Related Methods

- “GetMetadata Method” on page 109

GetNamespaces Method

Short Description

Gets the namespaces defined on the SAS Metadata Server.

Category

IOMI interface management methods

Syntax

```
rc=GetNamespaces(namespaces,flags,options);
```

Parameters

Table 7.12 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
namespaces	C	out	Returned list of namespaces.
flags	L	in	Passed indicator for flags. No flags are currently defined. For no flags, a 0 should be passed.

Parameter	Type	Direction	Description
options	C	in	Passed indicator for options. No options are currently defined.

Details

A namespace specifies a group of related metadata types that can be accessed by the SAS Open Metadata Interface. The NAMESPACES parameter returns the name of the namespaces that are currently defined in the SAS Repository Manager.

The SAS Open Metadata Interface provides the following namespaces:

- The REPOS namespace contains the repository metadata types.
- The SAS namespace contains metadata types describing application elements.

Example 1: Standard Interface

The following is an example of how to issue the GetNamespaces method regardless of the programming environment. The request gets the namespaces in the current SAS Repository Manager.

```
namespaces="";
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 is an example of an XML string that 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 -->
<GetNamespaces>
   <Namespaces/>
   <Flags>0</Flags>
   <Options/>
</GetNamespaces>
```

Related Methods

- “GetTypes Method” on page 132
- “GetSubtypes Method” on page 128

GetRepositories Method

Short Description

Gets the metadata repositories on the SAS Metadata Server.

Category

IOMI interface repository methods

Syntax

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

Parameters

Table 7.13 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
repositories	C	out	Returns list of all repositories that are registered on the SAS Metadata Server.

Parameter	Type	Direction	Description
flags	L	in	OMI_ALL=1 Gets information about each repository, such as the repository's availability, location, format, type, engine, and created and last updated dates. Lists repositories.
options	C	in	Passed indicator for options. No options are currently defined.

Details

A repository is a collection of related metadata objects. Each repository is registered in the SAS Repository Manager, which is also a SAS Metadata Repository. The SAS Metadata Server can access only those repositories that are registered in the SAS Repository Manager. There is one SAS Repository Manager for a SAS Metadata Server.

By default, the GetRepositories method gets "Identifying Metadata" on page 78, the description, and the default namespace for each repository that is registered in the SAS Repository Manager.

When issued with the OMI_ALL (1) flag set, the GetRepositories method gets information about the SAS Repository Manager and the following additional attributes for each repository:

Access= "*value*"
a descriptor that indicates the access mode that the administrator set for the repository. Valid values are:

OMS_FULL
Specifies the repository is available to all users for Read and Write access.

OMS_ADMIN
Specifies the repository is available only to users who have administrative user status on the SAS Metadata Server.

OMS_OFFLINE
Specifies the repository is unavailable to all users.

OMS_READONLY
Specifies the repository is only to be read.

CurrentAccess= "OMS_FULL | OMS_READONLY | OMS_ADMIN | OMS_ADMINRO | OMS_OFFLINE"
The SAS Metadata Server manages two versions of repositories: a memory version and a disk version. The memory version enables updates to be made available to clients before the disk version is updated. This attribute is set by the SAS Metadata Server on the memory version of the repository when the repository cannot be updated by the server because the repository has an incompatible repository format or has encountered an I/O error. This attribute is not stored in the disk version of the repository.

The values that can be returned for CurrentAccess are the same as Access plus OMA_ADMINRO (specifies the repository is only to be read by administrators). When a problem is encountered, CurrentAccess will have fewer values compared to Access. When the SAS Metadata Server can access a repository as intended, GetRepositories returns a CurrentAccess value that matches the Access value.

Path= "*string*"
the pathname of the physical directory where the repository is located.

PauseState= "*empty-string* | ADMIN | ADMIN(READONLY) | READONLY | OFFLINE"
the repository state after a server pause. This attribute is set by the Pause method and cleared by the Resume method. If the server is not paused, the value is an empty string. The value is usually the server Pause value (ADMIN or OFFLINE) unless the repository is registered with a less restrictive Access value.

RepositoryFormat= "*number*"
a numeric double value indicating the format level of the repository. (For example, 14.0.)

RepositoryType= "FOUNDATION | CUSTOM | PROJECT"
the repository type.

Beginning in SAS 9.4, the GetRepositories method returns these attributes when OMI_ALL is set:

Engine="BASE"
the name of the engine that created the repository. The valid value is BASE.

MetadataCreated="*date*"
a server-generated value that indicates the date and time the repository object was created.

MetadataUpdated="*date*"
a server-generated value that indicates the date and time the repository object was last updated.

Options="*library-options*"
any library options defined on the repository.

The additional attributes that are retrieved by OMI_ALL are available in the standard interface and with the DoRequest method when the method is issued on a SAS Metadata Server that is ONLINE or paused to an ADMIN state. A GetRepositories method that is issued on a SAS Metadata Server that is paused to an OFFLINE state returns an error unless the method is issued in the standard interface.

Example 1: Standard Interface

The following is an example of how to issue the GetRepositories method regardless of the programming environment. The request has no flags set.

```
flags=0;
options="";
rc = GetRepositories(repositories,flags,options);
```

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- XML string returned in repositories parameter -->
<Repositories>
```

```
   <Repository Id="A0000001.A5FYFEK5" Name="Foundation"
      Desc="Foundation repository" DefaultNS="SAS"/>
     <Repository Id="A0000001.A5HZY944" Name="Repository 1"
      Desc="First repository created for FULL access"
      DefaultNS="SAS"/>
   <Repository Id="A0000001.A5G3R7J5" Name="Repository 2"
      Desc="Second repository created for READONLY access"
      DefaultNS="SAS"/>
   <Repository Id="A0000001.A5SAMPL3" Name="Repository 3"
      Desc="Third repository created for ADMIN access"
      DefaultNS="SAS"/>
   <Repository Id="A0000001.A56DZUC5" Name="Repository 4"
      Desc="Fourth repository created for OFFLINE access"
      DefaultNS="SAS"/>
   <Repository Id="A0000001.A5G67U31" Name="Repository 5"
      Desc="Project repository" DefaultNS="SAS"/>
</Repositories>
```

Example 2: DoRequest Method

The following is an example of an XML string that 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 -->
<GetRepositories>
   <Repositories/>
   <Flags>0</Flags>
   <Options/>
</GetRepositories>
```

For an example that sets the OMI_ALL (1) flag, see “Using GetRepositories to Get All Repository Attributes” on page 312.

GetResponsibleParty Method

Short Description

Gets the ResponsibleParty object associated with the specified Person or IdentityGroup and responsibility.

Category

IOMI interface user interface helper methods

Syntax

```
rc=DoRequest("<GetResponsibleParty>...</GetResponsibleParty>",outMetadata);
```

Parameters

Table 7.14 *Method Parameters*

Parameter	Type	Direction	Description
<GetResponsibleParty/>	C	in	Metadata property string that identifies a ResponsibleParty object. See the “Details” section for information about the format of the metadata property string.

Details

The GetResponsibleParty method enables clients to get a ResponsibleParty object for a specified identity that might exist in a SAS Metadata Repository to which the requesting user does not have ReadMetadata permission. The method gets ResponsibleParty objects associated with both Person and IdentityGroup objects. The GetMetadata method is typically used to get the ResponsibleParty objects associated with an identity. However, the GetMetadata method returns only objects which the requesting user is authorized to read.

The GetResponsibleParty method is available in the DoRequest interface only. The method and its parameters are specified in an XML input string within the INMETADATA parameter of the DoRequest method. The method’s output is returned in the DoRequest method’s OUTMETADATA parameter.

The XML input string consists of a <GETRESPONSIBLEPARTY> element that passes a metadata property string that identifies a ResponsibleParty object in the following form:

```
<ResponsibleParty IdentityName='name' Responsibility='role'/>
```

The GetResponsibleParty method gets the ResponsibleParty object whose Name and Role attribute values match the specified IDENTITYNAME and RESPONSIBILITY values.

A user who has administrative status on the SAS Metadata Server can get the ResponsibleParty object of any user and role. A typical user can get only a ResponsibleParty object for his or her name and role.

The name value comparison is not case sensitive as the security subsystem enforces name-uniqueness for Person objects in the SAS Metadata Server. A null value in IDENTITYNAME implies the connected user. If the specified or implied

identity is not found on the SAS Metadata Server, an error stating that the object was not found is returned.

The role value comparison is performed as follows:

1. A case-sensitive comparison is performed. If a match is found, then the Id value of the ResponsibleParty object is returned.
2. If a match is not found, a case-insensitive comparison is performed. If a match is found, then the Id value of the ResponsibleParty object is returned.
3. If a match is not found, then the Id value of the ResponsibleParty object is not returned. An error message is not issued.
4. A null value is not accepted in the RESPONSIBILITY parameter. The ResponsibleParty method returns an error if you omit the RESPONSIBILITY parameter or if it specifies a null value.

Although the AddResponsibleParty method does not create a ResponsibleParty object for the Public IdentityGroup, a user who is connected to the SAS Metadata Server as Public can use the GetResponsibleParty method to get the ResponsibleParty object of a valid IdentityGroup. An IdentityGroup is valid if Public is defined as a member.

The output of the GetResponsibleParty method mirrors the input, except the object identifier of the requested object is included, if a match is found.

Example

The following is an example of a DoRequest method call that issues a GetResponsibleParty request.

```
outMetadata=""
inMetadata =
"<GetResponsibleParty>
   <ResponsibleParty IdentityName=' ' Responsibility='Owner'/>
</GetResponsibleParty>";

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

In this example, the method gets the ResponsibleParty objects that store a Role value of “Owner” for the connected user. If the requesting user is connected as Public, the method returns an error.

Related Methods

- “DoRequest Method” on page 107
- “AddResponsibleParty Method” on page 94

GetSubtypes Method

Short Description

Gets all possible subtypes for a specified metadata type.

Category

IOMI interface management methods

Syntax

```
rc=GetSubtypes(supertype,subtypes,ns,flags,options);
```

Parameters

Table 7.15 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
supertype	C	in	Name of the metadata type for which you want to get a list of subtypes.
subtypes	C	out	Returned XML list of all subtypes for the specified metadata type.
ns	C	in	Namespace to use as the context for the request.
flags	L	in	OMI_ALL_DESCENDANTS=64 Specifies to get the descendants of the returned subtypes and the subtypes.

Parameter	Type	Direction	Description
options	C	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see "<DOAS> Option" on page 89.

Details

Subtypes are metadata types that adopt the characteristics of a specified metadata supertype. In addition, a subtype can have subtypes of its own.

The SUBTYPES parameter returns an XML string that has 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 any subtypes of its own. If it has a value of 1, then the subtype does have subtypes of its own.

The GetSubtypes method does not return metadata about descendants unless the OMI_ALL_DESCENDANTS flag is set.

Example 1: Standard Interface

The following is an example of how to issue the GetSubtypes method regardless of the programming environment. The request gets the subtypes for supertype DataTable.

```
supertype= "DataTable";
ns= "SAS";
flags= 0;
options= "";
rc = GetSubtypes(supertype,subtypes,ns,flags,options);
```

Here is an example of the output returned by the SAS 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 is an example of an XML string that 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 Method” on page 132
- “IsSubtypeOf Method” on page 137

GetTypeProperties Method

Short Description

Gets all possible properties for a specified metadata type.

Category

IOMI interface management methods

Syntax

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

Parameters

Table 7.16 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
type	C	in	Name of the metadata type for which you want to get a list of properties.
properties	C	out	Returned XML list of the attributes and associations defined for the specified metadata type in the SAS Metadata Model.
ns	C	in	Namespace to use as the context for the request.
flags	L	in	OMI_ALL=1 Specifies to get a description of the supported value for each property.
options	C	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see “<DOAS> Option” on page 89.

Details

The GetTypeProperties method gets an XML list of the attributes and associations defined for the specified metadata type in the SAS Metadata Model. The metadata type is specified in the TYPE parameter.

When the OMI_ALL (1) flag is set, the method also gets a description of each property.

Example 1: Standard Interface

The following is an example of how to issue the GetTypeProperties method regardless of the programming environment. The request gets the properties of the Column metadata type. No flags are set.

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

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

Example 2: DoRequest Method

The following is an example of an XML string that shows how to format the request in example 1 for the INMETADATA parameter of the DoRequest method. The OMI_ALL (1) flag is set.

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

<GetTypeProperties>
   <Type>Column</Type>
   <Properties/>
   <NS>SAS</NS>
   <Flags>1</Flags>
   <Options/>
</GetTypeProperties>
```

Related Methods

- “GetTypes Method” on page 132
- “GetSubtypes Method” on page 128

GetTypes Method

Short Description

Gets all of the metadata types in a namespace.

Category

IOMI interface management methods

Syntax

```
rc=GetTypes(types,ns,flags,options);
```

Parameters

Table 7.17 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
types	C	out	Returned XML list of metadata types.
ns	C	in	Namespace to use as the context for the request. Valid values are REPOS or SAS.
flags	L	in	OMI_SUCCINCT=2048 Specifies to check the OPTIONS parameter for a <REPOSID> element and to list the metadata types for objects that exist in the specified repository.
options	C	in	Passed indicator for options. <REPOSID> Specifies a repository identifier. See the “Details” section for information about how to format the information in this element.

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 in the SAS namespace, the method returns an XML string that lists only metadata types for which objects exist in the specified repository.

The XML string is returned in the TYPES parameter. Each metadata type listed has a HasSubtypes attribute that indicates whether the metadata type has any subtypes. If this attribute has a value of 0, then the metadata type does not have any subtypes. If it has a value of 1, then the metadata type does have subtypes.

The <REPOSID> element specifies a repository identifier in the following form:

```
<Reposid>A0000001.RepositoryId</Reposid>
```

A0000001 is the SAS Repository Manager identifier. *RepositoryId* is the unique 8–character identifier of a SAS Metadata Repository. The <REPOSID> element must be specified with the OMI_SUCCINCT flag.

Example 1: Standard Interface

The following is an example of how to issue the GetTypes method regardless of the programming environment. The request gets the metadata types defined in the SAS namespace. For an example of a GetTypes request that sets the OMI_SUCCINCT (2048) flag, see “Using GetTypes to Get Actual Metadata Types in a Repository” on page 310.

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

rc=GetTypes(types,ns,flags,options);
```

Example 2: DoRequest Method

The following is an example of an XML string that 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 -->
<GetTypes>
   <Types/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
 </GetTypes>
```

Related Methods

- “GetNamespaces Method” on page 120
- “GetSubtypes Method” on page 128

GetUserFolders Method

Short Description

Gets a user’s home folder or subfolders.

Category

IOMI interface user interface helper methods

Syntax

```
rc=DoRequest("<GetUserFolders>...</GetUserFolders>",outMetadata);
```

Parameters

Table 7.18 *Method Parameters*

Parameter	Type	Direction	Description
<Tree/>	C	in	Metadata property string that gets a home folder or subfolder. See the “Details” section for information about the format of the metadata property string.

Details

The GetUserFolders method is available in the DoRequest interface only. The method and its parameters are specified in an XML input string within the INMETADATA parameter of the DoRequest method. The method's output is returned in the DoRequest method's OUTMETADATA parameter.

User folders are represented in the SAS Metadata Server as Tree metadata objects. The XML input string consists of a <GETUSERFOLDERS> element that passes a metadata property string that identifies a Tree object in the following form:

```
<Tree PersonName='name' FolderName='folder-type'/>
```

The PERSONNAME value must specify the Name attribute value of a Person object or be blank. If PERSONNAME is blank and the requesting user has a metadata identity, the user folder belonging to the requesting user is returned. If an IdentityGroup name is specified, the method will return an error. User folders are not supported for IdentityGroups at this time.

The FOLDERNAME value must be one of "Home Folder", "My Folder", or "Application Data", or the method will return an error.

The method uses the AssociatedHomeFolder association defined for the Person object identified by PERSONNAME to locate the folder requested by FOLDERNAME. The method returns the Tree object's 17-character metadata identifier and DisplayName attribute value. The locale used to create the DisplayName value is provided to the SAS Metadata Server in the LOCALE server invocation option or in the sasv9.cfg file.

Example

The following is an example of a DoRequest method that issues a GetUserFolders request. The method requests to get the "My Folder" folder of a person named "SAS Web Administrator."

```
outMetadata=""
inMetadata =
"<GetUserFolders>
<Tree PersonName='SAS Web Administrator' FolderName='My Folder'/>
</GetUserFolders>";

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

If the request is successful, the XML returned in the OUTMETADATA parameter mirrors the input in the INMETADATA parameter. The output includes the Id and DisplayName values of the specified folder.

The request is rejected with an authorization error if the requesting user is not "SAS Web Administrator," is not an administrative user of the SAS Metadata Server, or if "SAS Web Administrator" is the name of an IdentityGroup.

Related Methods

- “DoRequest Method” on page 107
- “AddUserFolders Method” on page 97

IsSubtypeOf Method

Short Description

Determines whether one metadata type is a subtype of another metadata type.

Category

IOMI interface management methods

Syntax

```
rc=IsSubTypeOf(type,supertype,result,ns,flags,options);
```

Parameters

Table 7.19 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
type	C	in	Name of the metadata type that might be a subtype of SUPERTYPE.
supertype	C	in	Name of the metadata type that might be a supertype of TYPE.

Parameter	Type	Direction	Description
result	N	out	Returned indicator. 0 indicates that TYPE is not a subtype of SUPERTYPE. 1 indicates that it is a subtype.
ns	C	in	Namespace to use as the context for the request.
flags	L	in	Passed indicator for flags. No flags are currently defined.
options	B	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see "<DOAS> Option" on page 89.

Example 1: Standard Interface

The following is an example of how to issue the IsSubtypeOf method regardless of the programming environment. The request determines whether WorkTable is a subtype of DataTable.

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

rc = IsSubtypeOf(WorkTable, DataTable, result, ns, flags, options);
```

Example 2: DoRequest Method

The following is an example of an XML string that 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 -->

<IsSubtypeOf>
   <Type>WorkTable</Type>
   <Supertype>DataTable</Supertype>
   <Result/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</IsSubtypeOf>
```

Related Methods

- “GetSubtypes Method” on page 128

UpdateMetadata Method

Short Description

Updates specified metadata objects in a repository.

Category

IOMI interface write methods

Syntax

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

Parameters

Table 7.20 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 75.
inMetadata	C	in	Metadata property string that specifies the object and properties to be updated.
outMetadata	C	out	Returned metadata property string that includes the results of the Update operation.

Parameter	Type	Direction	Description
ns	C	in	Namespace to use as the context for the request.
flags	L	in	OMI_IGNORE_NOTFOUND = 134217728 Prevents an Update operation from being canceled when the Function="REMOVE" attribute is specified on an association and the association does not exist. OMI_RETURN_LIST = 1024 Specifies to return the identifiers of any dependent objects that were deleted as a result of the Update operation. OMI_TRUSTED_CLIENT = 268435456 Determines whether the client can call this method. This flag is required. OMI_UNLOCK=131072 Unlocks an object lock that is held by the caller. OMI_UNLOCK_FORCE=262144 Unlocks an object lock that is held by another user.
options	C	in	Passed indicator for options. <DOAS Credential="*credHandle*"/> Enables a client to make a metadata request for another user. For more information, see “<DOAS> Option” on page 89.

Details

The UpdateMetadata method enables you to update the properties of existing metadata objects. It returns an error if the metadata object to be updated does not exist.

You can modify an object’s attributes and associations, unless the association is designated as "required for add" in the metadata type documentation.

When modifying an association, you must specify a directive in the association name element in the input metadata property string. This directive indicates whether the association is being appended, modified, removed, or replaced in the object’s association list. Different directives are supported for single and multiple associations. For more information about these directives and general UpdateMetadata usage, see Chapter 12, “Updating Metadata Objects,” on page 293.

You must have a metadata identity defined on the SAS Metadata Server 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. The OMI_LOCK flag is set in the GetMetadata method to provide basic concurrency controls in preparation for an update. For an overview of multi-user concurrency controls supported by the SAS Open Metadata Interface, see Chapter 18, “Metadata Locking Options,” on page 393. When OMI_UNLOCK or OMI_UNLOCK_FORCE is set, only specified objects are unlocked. Associated objects are not unlocked.

Check the return code of an UpdateMetadata method call. A nonzero return code indicates that a failure occurred while trying to write the metadata. A nonzero return code means none of the changes in the method call were made.

Example 1: Standard Interface

The following is an example of how to issue the UpdateMetadata method regardless of the programming environment. The specified attribute values replace values 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";
<!- OMI_TRUSTED_CLIENT flag ->
flags= 268435456;
options= "";

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

Example 2: DoRequest Method

The following is an example of an XML string that 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 -->

<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 Method” on page 102
- “GetMetadata Method” on page 109

8

Authorization (ISecurity Interface)

Overview of the ISecurity Server Interface

The methods described in this section are provided in the ISecurity server interface. The methods can be used in a SAS Open Metadata Interface client that you create to request authorizations on SAS Metadata Server resources. The methods can be used to get authorizations on both metadata and on the data that is represented by the metadata.

ISecurity methods are available only through the standard interface. For more information, see "Communicating with the SAS Metadata Server" on page 14.

Two versions of the ISecurity server interface are supported.

- ISecurity 1.0 enables SAS 9.1 clients to work the same way they worked in SAS 9.1. Only methods that were supported in SAS 9.1 are available in ISecurity 1.0.
- ISecurity 1.1 provides versions of the SAS 9.1 methods that work in SAS 9.2 and later environments. SAS 9.2 introduced support for server authentication via internal user accounts as well as the traditional external user accounts. It also added security administration methods that were not available in SAS 9.1.

The following information applies to all of the ISecurity methods.

- Errors are surfaced through the exception-handling in IOM. Each method returns a set of documented exceptions. Use TRY and CATCH logic in your Java programs to determine when an exception is returned.
- The methods make authorization decisions based on user and access control metadata that is stored in metadata repositories. Appropriate metadata must be defined for authorization decisions to be meaningful.

  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 with macros. For information about the plug-ins, see SAS Management Console documentation.

  Access control metadata is defined by using the SAS Management Console Authorization Manager plug-in or by using ISecurityAdmin methods. For information about ISecurityAdmin methods, see Chapter 9, "Security Administration (ISecurityAdmin Interface)," on page 195.

  For information about access controls supported by the SAS Open Metadata Architecture authorization facility and enterprise user import macros, see the *SAS Intelligence Platform: Security Administration Guide*.

- The methods assume the calling user and any user IDs specified by the calling program have been authenticated before calling the SAS Metadata Server. A caller that is invoking ISecurity methods for itself does not have to be a trusted user. A caller that is invoking the GetCredential method for another user, or is using the credential handle obtained from GetCredentials for another user, must be a trusted user.
- In the examples, iSecurity is an instantiation of the ISecurity interface.

Using the ISecurity Server Interface

Calling the Server Interface

The ISecurity interface is called by connecting to the SAS Metadata Server and obtaining a handle to the ISecurity server interface.

A SAS Java Metadata Interface client accesses the ISecurity server interface by importing the appropriate packages, instantiating an object factory, and connecting to the SAS Metadata Server with a handle to the interface that is appropriate for the task that it wants to perform.

The ISecurity server interface is provided in the sas.oma.omi.jar file in the SAS Platform VJR. A Java client accesses the ISecurity server interface by importing the appropriate com.sas.meta.SASOMI packages.

The ISecurity interface versions are designed so that existing SAS clients can continue to work unchanged.

- To use SAS 9.1 methods only, import com.sas.meta.SASOMI.ISecurity and com.sas.meta.SASOMI.ISecurityPackage.
- To use SAS 9.1 methods and methods introduced after that release, import com.sas.meta.SASOMI.ISecurity_1_1 and com.sas.meta.SASOMI.ISecurity_1_1Package.

The SAS Java Metadata Interface provides the MdFactory interface to instantiate an object factory for the SAS Metadata Server and the MdOMRConnection interface for connecting to the SAS Metadata Server. Use the MdOMRConnection interface's makeISecurityConnection method to connect to the server with the ISecurity server interface.

Identifying Resources to ISecurity Methods

Many ISecurity methods have a resource parameter. A resource is a metadata object that represents the entity on which authorization or another action is requested.

A resource is identified by a URN in one of two forms:

`OMSOBJ:`*`MetadataType/ObjectId`*

`REPOSID:`*`_reposID`*

OMSOBJ indicates that the request is to the SAS namespace of the SAS Metadata Model. The SAS namespace contains metadata types that describe application elements. *MetadataType* is one of the SAS namespace metadata types. For a list of supported metadata types, see the SAS Metadata Model documentation. *ObjectId* is the requested object's 17–character metadata object identifier. The first eight characters of the object identifier are a repository identifier; the remaining eight characters are the unique object instance identifier.

REPOSID indicates the request is to the REPOS namespace of the SAS Metadata Model. The REPOS namespaces contains metadata types that describe a repository. The first eight characters of a repository ID are the SAS Repository Manager identifier A0000001, which is the same for all repositories. Therefore, you need specify only a repository's unique 8–character object instance identifier in *_reposID*.

Identifying Users

The SAS Metadata Server supports user identities of metadata type Person, IdentityGroup, and Role.

Most ISecurity methods accept a credential handle or use the user ID of the calling user to identify the identity for which to return an authorization or information. A credential handle is a token representing an identity's authorizations on the SAS Metadata Server . A handle is obtained with the GetCredentials method. For more information, see “GetCredentials Method” on page 164.

The following methods support additional ways to specify the identity for which to process a request:

- The GetApplicationActionsAuthorizations method supports submission of the string ROLE_*rolename* to specify a Role. For more information, see “GetApplicationActionsAuthorizations Method” on page 153.
- The GetIdentity method supports submission of the string LOGINID: *userid* to identity a Person or IdentityGroup. For more information, see “GetIdentity Method” on page 166.
- The GetInfo method supports the submission of an identity resource identifier in the form *IdentityType: Name*, where *IdentityType* can be Person, IdentityGroup, or Role. For more information, see “GetInfo Method” on page 168.

Methods that create and manage internal user accounts use a different convention to identify a user. Internal user accounts are supported only for identities of metadata type Person. These accounts rely on the Person object's Name value to identify the account. Therefore, methods that create and operate on internal user accounts require you to identify the internal user by name. For more information, see the following:

- “SetInternalPassword Method” on page 193
- “SetInternalLoginUserOptions Method” on page 189
- “GetInternalLoginUserInfo Method” on page 176
- “DeleteInternalLogin Method” on page 150

Understanding the ISecurity 1.0 Interface

The ISecurity 1.0 interface includes the following authorization methods:

GetCredentials
: Returns a handle to a credential. For more information, see “GetCredentials Method” on page 164.

FreeCredentials
: Frees the handle returned by GetCredentials. For more information, see “FreeCredentials Method” on page 152.

GetAuthorizations
: Gets authorization information for a resource, depending on the type of authorization requested. For more information, see “GetAuthorizations Method” on page 156.

GetIdentity
: Gets identity metadata for the specified user. For more information, see “GetIdentity Method” on page 166.

IsAuthorized
: Determines whether an authenticated user is authorized to access a resource with a specific permission. For more information, see “IsAuthorized Method” on page 183.

Understanding the ISecurity 1.1 Interface

The ISecurity 1.1 interface contains three categories of methods:

- ISecurity 1.0 authorization methods that were updated to support internal user accounts as well as the traditional external user accounts
- Internal authentication methods
- Generalized authorization methods

In order of use, the internal authentication methods are the following:

GetInternalLoginSitePolicies
: Returns the active server-level internal authentication policies. For more information, see “GetInternalLoginSitePolicies Method” on page 174.

SetInternalPassword
: Creates an InternalLogin object for the specified user. For more information, see “SetInternalPassword Method” on page 193.

SetInternalLoginUserOptions
: Customizes internal authentication policies for the specified user. For more information, see “SetInternalLoginUserOptions Method” on page 189.

GetInternalLoginUserInfo
: Gets availability information and internal authentication settings for the specified user. For more information, see “GetInternalLoginUserInfo Method” on page 176.

DeleteInternalLogin
: Deletes the InternalLogin object that is associated with the specified user. For more information, see “DeleteInternalLogin Method” on page 150.

In alphabetical order, the generalized authorization methods are the following:

GetApplicationActionsAuthorizations
: Returns authorizations for ApplicationActions in a SoftwareComponent object. For more information, see “GetApplicationActionsAuthorizations Method” on page 153.

GetAuthorizationsForObjects
: Gets authorizations for a specified set of objects and permissions. For more information, see “GetAuthorizationsforObjects Method” on page 159.

GetInfo
: Retrieves identity information, depending on the value in the INFOTYPE parameter, including the origin of a specified identity's privileges, the value of active enterprise policies, and so on. For more information, see “GetInfo Method” on page 168.

GetLoginsforAuthDomain
: Retrieves the logins for the connected user for the specified authentication domain in order of identity precedence. For more information, see “GetLoginsforAuthDomain Method” on page 180.

IsInRole
: Returns the TRUE value when the user specified in CREDHANDLE is in a role. For more information, see “IsInRole Method” on page 186.

DeleteInternalLogin Method

Short Description

Deletes the InternalLogin object that is associated with the specified user.

Category

ISecurity interface internal authentication methods

Interface Version

ISecurity 1.1

Syntax

```
DeleteInternalLogin(personName);
```

Parameters

Table 8.1 *Method Parameters*

Parameter	Type	Direction	Description
personName	string	in	Specifies the Name attribute value of the Person object whose InternalLogin you want to delete. Unlike in other security methods, the Name value is specified as simply*Name*.

Details

You must have user administration capabilities on the SAS Metadata Server to delete an InternalLogin object. For information about user administration capabilities, see “Users, Groups, and Roles: Main Administrative Roles” in the *SAS Intelligence Platform: Security Administration Guide*.

The DeleteInternalLogin method deletes the InternalLogin object that is associated with the specified user. Use the DeleteMetadata method to delete the Person object that is associated with the InternalLogin object.

Exceptions Thrown

The DeleteInternalLogin method does not return any exceptions.

Examples

The following is a Java example of a DeleteInternalLogin method call:

```
// Assumes a Person object with Name='testId' exists
// and has an InternalLogin object associated with it
String personName = "testId";

iSecurity.DeleteInternalLogin(personName);
```

Related Methods

- “SetInternalLoginUserOptions Method” on page 189

FreeCredentials Method

Short Description

Frees the handle returned by GetCredentials.

Category

ISecurity interface authorization methods

Interface Version

ISecurity 1.0

Syntax

```
FreeCredentials(credHandle);
```

Parameters

Table 8.2 Method Parameters

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle to free.

Details

The FreeCredentials method frees the SAS Metadata Server credentials associated with the handle returned by the GetCredentials method. Each handle returned by the GetCredentials method should be freed.

Exceptions Thrown

The FreeCredentials method does not return any exceptions.

Example

The following is a Java example of a FreeCredentials method call:

```
//  Assumes parameter is a valid credential handle that
// was previously obtained with the GetCredentials method
iSecurity.FreeCredentials(credHandle.value);
```

Related Methods

- “GetCredentials Method” on page 164

GetApplicationActionsAuthorizations Method

Short Description

Returns authorizations for ApplicationActions in a SoftwareComponent object.

Category

ISecurity interface generalized authorization methods

Interface Version

ISecurity 1.1

Syntax

```
GetApplicationActionsAuthorizations(credHandle,applicationContext,options,output);
```

Parameters

Table 8.3 *Method Parameters*

Parameter	Type	Direction	Description
credHandle	string	in	Credentials handle identifying a user identity, an empty string, or the Name of a Role in the form ROLE_*rolename*.
applicationContext	string	in	The 17-character metadata object identifier of the SoftwareComponent object representing the application on which the actions are registered.
options	string array	in	Two-dimensional string array with two input columns. Specifies additional properties to return about granted ApplicationActions. Supported options are an empty string and the keyword-only options: PERMCOND Requests to return any permission conditions that are defined. ALLATTRS Requests to return all attributes. An empty string indicates that no additional properties are requested.

Parameter	Type	Direction	Description
output	string array	out	Two-dimensional string array with a varying number of output columns, depending on which OPTIONS are set. Possible columns include the following: Column 0: ActionIdentifier Column 1: PermissionCondition Column 2: 'Y' - user is granted; otherwise, an empty string Column 3: Name Column 4: ActionType Column 5: ObjectIdentifier

Details

The GetApplicationActionsAuthorizations method returns authorizations based on ApplicationAction objects that are associated with a SoftwareComponent object. These authorizations indicate the actions that a user can perform in the application that is represented by the SoftwareComponent object.

The expected use is that applications define ApplicationAction objects that are valid for their application, as well as for a user context. The GetApplicationActionAuthorizations method lists the ApplicationActions for which the specified user has been granted Execute permission.

When a credential handle is used, the method returns authorizations for the identity that corresponds to the specified handle. If the CREDHANDLE parameter is an empty string, the method returns authorizations for the calling user.

If authorization is requested based on role membership, you should specify the Role name in the form ROLE_*rolename*. In the string ROLE_*rolename*:

- ROLE_ is a character constant prefix.
- *rolename* is the Name value of a Role object on the SAS Metadata Server.

The PERMCOND option returns any PermissionCondition objects that have been defined to qualify an authorization.

The ALLATTRS option returns the following attributes about each granted ApplicationAction:

ActionIdentifier
: fixed system name of the ApplicationAction

Name
: localizable name of the ApplicationAction

ActionType
: optional application-specific descriptor for the ApplicationAction

ObjectIdentifier
metadata identifier of the ApplicationAction object

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetApplicationActionsAuthorizations method:

- NotTrustedUser
- InvalidCredHandle—This exception is also returned when the ROLE_*rolename* value is invalid or does not exist.
- InvalidResourceSpec

Related Methods

"IsInRole Method" on page 186

GetAuthorizations Method

Short Description

Gets authorization information for a resource, depending on the type of authorization requested.

Category

ISecurity interface generalized authorization methods

Interface Version

ISecurity 1.0

Syntax

```
GetAuthorizations(authType,credHandle,resource,permission,authorizations);
```

Parameters

Table 8.4 *Method Parameters*

Parameter	Type	Direction	Description
authType	string	in	The type of authorization to perform. Valid values are Cube or SharedDimension.
credHandle	string	in	Credential handle identifying a user identity, or an empty string.
resource	string	in	Passed resource identifier.
permission	string	in	Mnemonic representation of the permission for which authorization is being requested. This parameter can be an empty string for some AUTHTYPE values.
authorizations	string array	out	Returned two-dimensional string array. The content and structure of the array varies depending on the authorization type specified in the AUTHTYPE parameter.

Details

The GetAuthorizations method performs authorization queries. The input for processing the query, and the format and content of the information returned, are determined by the AUTHTYPE parameter.

An AUTHTYPE value of Cube returns a two-dimensional string array. The number of rows depends on the structure of the cube. Each row has the following four columns:

Type
: Indicates the metadata type in the row. This is either Hierarchy, Dimension, SharedDimension, Measure, or Level.

Name
: Returns the Name attribute of the metadata type instance.

Authorized
: Returns a Y or N, indicating whether the permission being requested has been granted to the user in this cube component.

PermissionCondition
When the Authorized column has a value of Y, this column returns a condition that must be enforced on the cube component to allow access. For more information, see the description of the PERMISSIONCONDITION parameter in "IsAuthorized Method" on page 183.

An AUTHTYPE value of SharedDimension returns a two-dimensional string array. In this case, the first row of output contains the aforementioned column values for the SharedDimension, itself. Subsequent rows contain column values for each Level and Hierarchy object associated to the SharedDimension.

If the CREDHANDLE parameter is an empty string, the method returns authorizations for the calling user.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetAuthorizations method:

- NotTrustedUser
- InvalidCredHandle
- InvalidResourceSpec
- InvalidAuthType

Examples

The following is a Java example of a GetAuthorizations method. The method call gets authorizations for a cube. Code is included that formats and prints the results of the request.

```
public void getAuthorizationsforCube() throws Exception {
   try
   {
// Issue GetAuthorizations on a predefined cube. Assume that a credential
// handle was obtained earlier. Supported authType values are "Cube"
// or "SharedDimension". This call gets authorizations for a cube.
// "Read" is the permission being sought.

 iSecurity.GetAuthorizations("Cube", credHandle.value, cube_URN, "Read", auths);

               System.out.println();
               // Specifies to print a title and parameter values.
               System.out.println("<<<<<< getAuthorizations() call parameters
(Read Permission) with results >>>>>>");
               System.out.println("credHandle=" + credHandle.value);
               System.out.println("resourceURN=" + cube_URN);
               System.out.println("permission=Read");

              // Defines  a string array to store method output
              String[][] returnArray = auths.value;
               for (int i=0; i < returnArray.length; i++ )
```

```
                {
                        String[] returnRow = returnArray[i];
                        // Return values are in fixed column positions:
                        // Type | Name | Authorized (Y/N) | PermissonCondition
                        System.out.print("Type="+returnRow[0] + ", ");
                        System.out.print("Name="+returnRow[1] + ", ");
                        System.out.print("Authorized="+returnRow[2] + ", ");
                        System.out.print("PermissonCondition="+returnRow[3]);
                        System.out.println(); // force NewLine
                }

                         System.out.println("<<<< End getAuthorizationsForCube() >>>>" );
        }
        // Catch the method's exceptions.
        catch (Exception e) {
                System.out.println("GetAuthorizations: GetInfo: other Exception");
                e.printStackTrace();
                throw e;
        }

}
```

Here is the output from the request:

```
<<<<<< getAuthorizations() call parameters (Read Permission) with results >>>>>>
credHandle=33f824f400000003
resourceURN=OMSOBJ:Cube/A5CY5BIY.AS000001
permission=Read
Type=Hierarchy, Name=testHier1, Authorized=Y, PermissonCondition=
Type=Dimension, Name=testDim1, Authorized=Y, PermissonCondition=
Condition for an OLAP Dimension
Type=Dimension, Name=testDim2, Authorized=N, PermissonCondition=

<<<< End getAuthorizationsForCube() >>>>
```

Related Methods

- “IsAuthorized Method” on page 183

GetAuthorizationsforObjects Method

Short Description

Gets authorizations for a specified set of objects and permissions.

Category

ISecurity interface authorization methods

Interface Version

ISecurity 1.1

Syntax

```
GetAuthorizationsforObjects(credHandle,permissions,resources,permMask,GRANT,
conditionNDXs,conditionPermMasks,conditions);
```

Parameters

Table 8.5 Method Parameters

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle identifying a user identity, or an empty string.
permissions	string array	in	Permissions for which authorizations are requested for the resources in the RESOURCES parameter. See the “Details” section for an example.
resources	string array	in	A one-dimensional string array containing passed resource identifiers. See the “Details” section for an example.

Parameter	Type	Direction	Description
permMask	integer array	in	A one-dimensional integer array, where each element corresponds positionally to each resource in the RESOURCES array, and each bit in an element corresponds positionally to each permission in the PERMISSIONS array. Each PERMMASK element is a bit pattern where 1 in a bit position means that the permission in the PERMISSIONS array is enforced for the corresponding object. A 0 in a bit position means that the GetAuthorizationsforObjects method should ignore the corresponding permission. See the “Details” section for an example.
GRANT	integer array	out	A one-dimensional integer array, where each element corresponds positionally to each resource in the RESOURCES array, and each bit in an element corresponds positionally to each permission in the PERMISSIONS array. Each GRANT element is a bit pattern, where 1 in a bit position means that the permission in the PERMISSIONS array is granted for the corresponding object. A 0 in a bit position means that the permission is denied or not selected for enforcement in the PERMMASK for the corresponding object.
conditionNDXs	integer array	out	A one-dimensional integer array, where each element corresponds positionally to each PermissionCondition in the CONDITIONS array. Each CONDITIONNDXS element value is the index into the RESOURCES array for which the PermissionCondition in the CONDITIONS array corresponds. If no PermissionConditions are returned for any of the resources, then the CONDITIONNDXS array is empty.

Parameter	Type	Direction	Description
conditionPermMasks	integer array	out	A one-dimensional integer array, where each element corresponds positionally to each index in the CONDITIONNDXS and CONDITIONS arrays. Each CONDITIONPERMMASKS element is a bit pattern, where 1 in a bit position means that the corresponding permission in the PERMISSIONS array has a PermissionCondition. If no PermissionCondition objects are returned for any of the resources, then the CONDITIONPERMMASKS array is empty. The CONDITIONPERMMASKS array lists the permissions for which PermissionCondition objects were returned for the resource referenced in the corresponding element in the CONDITIONNDXS array.
conditions	string array	out	A one-dimensional string array, where each element corresponds positionally to each permission in the CONDITIONNDXs and CONDITIONPERMMASKS arrays and contains a returned PermissionCondition value. If no PermissionCondition objects are returned for any of the resources, then the CONDITIONS array is empty.

Details

The GetAuthorizationsforObject method reduces the number of calls to the SAS Metadata Server for authorization decisions that require permissions on multiple metadata objects to be evaluated. For the specified set of metadata objects and a corresponding set of permissions (which can be different for each object), the method returns GRANT or a null value, and any PermissionCondition objects that are associated with a GRANT. A null value indicates that the permission was denied or not specified for the object.

When an empty string is passed in CREDHANDLE, the method evaluates authorizations for the calling user.

This is an example of a PERMISSIONS array:

```
{ "Read", "Write", "Create Table", "Select" }
```

For information about the format of a resource identifier, see “Identifying Resources to ISecurity Methods” on page 147.

This is an example of a RESOURCES array:

```
{
"OMSOBJ:Library/A5DRX6L4.AQ000001",
"OMSOBJ:Table/A5DRX6L4.AT000001",

"OMSOBJ:Column/A5DRX6L4.AU000006",
"OMSOBJ:Column/A5DRX6L4.AU000007"
}
```

This is an example of a PERMMASK array:

```
{  7, 15, 1,  2 }
```

Using information from the previous examples, the PERMMASK array indicates the following:

- the Read, Write, and Create Table permissions are enforced for OMSOBJ:Library/A5DRX6L4.AQ000001
- the Read, Write, Create Table, and Select permissions are enforced for "OMSOBJ:Table/A5DRX6L4.AT000001"
- the Read permission is enforced for OMSOBJ:Column/A5DRX6L4.AU000006
- the Write permission is enforced for OMSOBJ:Column/A5DRX6L4.AU000007

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetAuthorizationsforObjects method:

- InvalidCredHandle
- PermissionDoesNotExist
- InvalidObjectSpecification
- ObjectDoesNotExist
- InvalidPermMask

Related Methods

- “GetAuthorizations Method” on page 156
- “GetApplicationActionsAuthorizations Method” on page 153

GetCredentials Method

Short Description

Returns a handle to a credential.

Category

ISecurity interface authorization methods

Interface Version

ISecurity 1.0

Syntax

```
GetCredentials(userid,credHandle);
```

Parameters

Table 8.6 Method Parameters

Parameter	Type	Direction	Description
userid	string	in	Passed user ID of the authenticated user for whom a credential is requested, or an empty string.
credHandle	string	out	Returned credential handle identifying a user.

Details

The GetCredentials method returns a credential handle for the user identified in the USERID parameter. If the USERID parameter contains an empty string, a credential handle is returned for the user making the request.

A credential handle is a token representing an identity's authorizations on the SAS Metadata Server. Clients get and use the handle to reduce the number of authorization requests made to the SAS Metadata Server on behalf of a user.

Every credential handle that is returned by the GetCredentials method should be freed using the FreeCredentials method when it is no longer needed.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetCredentials method:

- NoCredential
- NotTrustedUser

Example

The following is a Java example of a GetCredentials method call:

```
public void getCredentials() throws Exception {
   try
   {
      String testUserId = new String("myDomain\myUserID");
      StringHolder credHandle = new StringHolder();

      iSecurity.GetCredentials(testUserId,credHandle);
   }
catch (Exception e) {
   System.out.println("GetCredentials: Exceptions");
   e.printStackTrace();
   throw e;
   }

}
```

Related Methods

- "FreeCredentials Method" on page 152
- "GetIdentity Method" on page 166

GetIdentity Method

Short Description

Gets identity metadata for the specified user.

Category

ISecurity interface authorization methods

Interface Version

ISecurity 1.0

Syntax

```
GetIdentity(credHandle,identity);
```

Parameters

Table 8.7 *Method Parameters*

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle identifying a user identity, an empty string, or a user ID in the form LOGINID:*userid*.
identity	string	out	Resource identifier describing the identity represented by the credential handle.

Details

By specifying a credential handle to the GetIdentity method, it returns a URN-like string describing the identity that corresponds to the specified handle. An identity refers to a Person or IdentityGroup object describing a user in a SAS Metadata Repository. The URN-like string is in the following form:

`OMSOBJ: MetadataType/ObjectId`

where

- *MetadataType* is Person or IdentityGroup.
- *ObjectId* is a unique metadata object instance identifier in the form *Reposid.ObjectId*.

If the CREDHANDLE parameter is an empty string, the output is identity metadata for the requesting user.

If the call is being made on behalf of a user whose user ID is known, specify it in the form LOGINID:*userid* to eliminate the need to issue GetCredentials and FreeCredentials calls before GetIdentity. In the LOGINID:*userid* string:

- LOGINID is a keyword that specifies to search Login objects.
- *userid* is the value of a Login object's UserID attribute.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exception for the GetIdentity method:

- InvalidCredHandle

Examples

The following are Java examples that show the three ways a GetIdentity call can be issued:

```
// 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.

 iSecurity.GetIdentity("",identityValue);

//  2) The first parameter is a valid credential handle that was obtained
```

```
//    with GetCredentials (not shown here). In this case, the returned
//   Identity corresponds to the credential handle.

 iSecurity.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");
iSecurity. GetIdentity(loginId.value,identityValue);
```

Related Methods

- “GetCredentials Method” on page 164

GetInfo Method

Short Description

Retrieves identity information, depending on the value in the INFOTYPE parameter, including the origin of a specified user’s privileges, the value of active enterprise policies, and so on.

Category

ISecurity interface generalized authorization methods

Interface Version

ISecurity 1.1

Syntax

```
GetInfo("infoType",credHandle,options,output);
```

Parameters

Table 8.8 Method Parameters

Parameters	Type	Direction	Description
infoType	string	in	Specifies the identity information to get. Valid values are: ■ GetIdentityInfo ■ EnterprisePolicies ■ SASPW_Alias
credHandle	string	in	A string that identifies the user identity for which information is requested. Valid values are: ■ A credential handle obtained by calling the GetCredentials method. ■ An empty string. ■ When INFOTYPE is "GetIdentityInfo", a valid URN for an identity or simply *IdentityType*:*Name*. In *IdentityType*:*Name*, *IdentityType* is Person, IdentityGroup, or Role. *Name* is the Name attribute value of the identity.
options	string array	in	Options submitted in a two-dimensional string array. Options are specific to the INFOTYPE value. The first column in the array must contain an option keyword. The second column contains the keyword value, if there is one. See the “Details” section for information about valid option values.
output	string array	out	A two-dimensional string array containing the output for the requested INFOTYPE. The first column has the name of the attribute whose value is being returned in the second column. See the “Details” section for information about the output for each INFOTYPE.

Details

CREDHANDLE Options

If CREDHANDLE is an empty string, then “INFOTYPE” requests information for the connected user. If CREDHANDLE is a credential handle and the connected user is a trusted user, or it is a URN-like or *IdentityType: Name* identifier, then information is returned for the specified identity. For information about the format of a URN, see “Identifying Resources to ISecurity Methods” on page 147. When an identifier other than a credential handle is used, the connected user does not have to be a trusted user. However, they must be granted ReadMetadata permission on the identity’s metadata object.

The *IdentityType*:*Name* form enables clients to obtain identity information when a credential cannot be obtained. This can happen because the associated login is not known or is not available in a particular scenario. An example of this type of scenario is when a client needs to determine whether an identity has extended privileges as a result of membership in the Unrestricted, User Administrator, or Operator roles, but has no way to authenticate the identity using any of the identity’s logins.

The following are examples of how the *IdentityType*:*Name* form is used:

```
'Person:Jane'
'IdentityGroup:AccountingDept'
'Role:AccountsPayableClerks'
```

INFOTYPE=“GetIdentityInfo” Options and Outputs

The “GetIdentityInfo” value supports the following option keywords:

ReturnUnrestrictedSource
: Returns an additional row in the output array if the specified user is an unrestricted user. Otherwise, an additional row is not returned. When a row is returned, the valid values are the following:

  Role
  : Indicates the user identity is a member of the SAS Metadata Server: Unrestricted role.

  ConfigFile
  : Indicates the user has a login user ID that matches a *user ID entry in the adminUsers.txt file.

  Role, ConfigFile
  : Indicates the user is unrestricted from both the Role and ConfigFile sources.

UserClass
: Returns one or more of the following values that describe the source of the identity’s privileges. When Unrestricted is returned, all of the privileges of Administrator and Operator are assumed. The privileges of Trusted are not assumed.

Unrestricted
Indicates the privilege comes from a *user ID entry in the adminUsers.txt file, or from a metadata identity that has membership in the SAS Metadata Server: Unrestricted role.

Administrator
Indicates the privilege comes from a user ID entry in the adminUsers.txt file that does not have an asterisk.

IdentityAdmin
Indicates the privilege comes from a metadata identity that has membership in the SAS Metadata Server: User and Group Administrators role.

Operator
Indicates the privilege comes from a metadata identity that has membership in the SAS Metadata Server: Operator role.

Normal
Indicates the user does not have any special privileges.

Trusted
Indicates the privilege comes from a user ID entry in the trustedUsers.txt file.

AuthenticatedUserid
Returns the domain-qualified user ID used to make the connection to the SAS Metadata Server, or the domain-qualified user ID corresponding to the specified CREDHANDLE.

IdentityName
Returns the Name attribute value of the Person or IdentityGroup object that corresponds to the authenticated user ID.

IdentityType
Returns Person or IdentityGroup.

IdentityObjectID
Returns the 17–character metadata object identifier of the specified identity.

UnrestrictedSource
Valid values are Role, ConfigFile, or 'Role, ConfigFile'.

INFOTYPE=“EnterprisePolicies” Options and Outputs

The “EnterprisePolicies” value requests enterprise policies. It supports the following option keywords:

ALL
Specifies to return all enterprise policies and their values.

SASSEC_LOCAL_PW_SAVE
Specifies to return the value of the SASSEC_LOCAL_PW_SAVE server configuration option. This server configuration option specifies whether users can create a local copy of the user ID and password that they submit when they log on to a SAS desktop application. A value of 0 indicates Yes. A value of 1 indicates No.

INFOTYPE=“SASPW_Alias” Output

The “SASPW_Alias” value has no option keywords. It returns the AuthenticationDomain alias of the SASPassword authentication provider. The

default value is saspw. However, if the AUTHPROVIDERDOMAIN start-up option is used to specify a different alias, then this INFOTYPE value returns the alias.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetInfo method:

- InvalidCredHandle
- InvalidInfoType
- InvalidOptionName
- InvalidOptionValue

Examples

The following is a Java example of a GetInfo method call. The method is issued twice. The first time it is issued, it gets identity information for the connected user. The second time, it gets identity information for a credentialed user. The example includes code that formats and prints the information returned by the two requests:

```
public void getInfo() throws Exception  {
    try
    {
        // Defines the GetIdentityInfo "ReturnUnrestrictedSource" option.
        final String[][] options ={{"ReturnUnrestrictedSource",""}};

        System.out.println(""); // Skip a line
        System.out.println("<<<< Begin getInfo() >>>>" );

        // Defines a stringholder for the info output parameter.
        VariableArray2dOfStringHolder info = new VariableArray2dOfStringHolder();

        // Issues the GetInfo method for the current iSecurity connection user.
        iSecurity.GetInfo("GetIdentityInfo","", options, info);
        String[][] returnArray = info.value;

        System.out.println();
        // Specifies a title for the output.
        System.out.println("<<<<<< getInfo() for ISecurity Connection User >>>>>>");
        System.out.println("credHandle=''");
        for (int i=0; i< returnArray.length; i++ )
        {
            System.out.println(returnArray[i][0] + "=" + returnArray[i][1]);
        }
        // Defines a stringholder for the credential handle.
        StringHolder credHandle = new StringHolder();

        // Issues the GetCredentials method.
        iSecurity.GetCredentials(testUserId, credHandle);
        // Issues the GetInfo method for the credentialed user
```

```
            iSecurity.GetInfo("GetIdentityInfo",credHandle.value, options, info);
            returnArray = info.value;

            System.out.println();
            // Skip one line
            // Specifies a title to print in the output.
            System.out.println("<<<<<< getInfo() for Credentialed User >>>>>>");
            System.out.println("credHandle=" + credHandle.value);
            for (int i=0; i< returnArray.length; i++ )
            {
                System.out.println(returnArray[i][0] + "=" + returnArray[i][1]);
            }

            // Issues the FreeCredentials method.
            iSecurity.FreeCredentials(credHandle.value);

            System.out.println("");
                   // Skip a line
            System.out.println("<<<< End getInfo() >>>>" );
        }
        // The following code catches the method's exceptions.
    catch (Exception e) {
            System.out.println("GetInfo: Exceptions");
            e.printStackTrace();
            throw e;
        }

    }
```

Here is the output from the requests:

```
<<<< Begin getInfo() >>>>

<<<<<< getInfo() for ISecurity Connection User >>>>>>
credHandle=''
UserClass=Unrestricted, Trusted
AuthenticatedUserid=TESTUSR7@CARYNT
IdentityName=PUBLIC
IdentityType=IdentityGroup
IdentityObjectID=A5CY5BIY.A3000002
UnrestrictedSource=ConfigFile

<<<<<< getInfo() for Credentialed User >>>>>>
credHandle=2d91581c00000000
UserClass=IdentityAdmin
AuthenticatedUserid=TESTUSER@SASPW
IdentityName=testUser
IdentityType=Person
IdentityObjectID=A5CY5BIY.AN000003

<<<< End getInfo() >>>>
```

Related Methods

- “GetInternalLoginUserInfo Method” on page 176

GetInternalLoginSitePolicies Method

Short Description

Returns the active server-level internal authentication policies.

Category

ISecurity interface internal authentication methods

Interface Version

ISecurity 1.1

Syntax

```
GetInternalLoginSitePolicies(siteMinPasswordLength,siteIsDigitRequired,
siteIsMixedCaseRequired,siteSizeHistoryList,sitePasswordChangeDelayInMinutes,
siteExpirationDays,siteNumFailuresForLockout,siteLockoutInMinutes,
siteDaysToSuspension);
```

Parameters

Table 8.9 *Method Parameters*

Parameter	Type	Direction	Description
siteMinPasswordLength	int	out	Specifies the minimum length for passwords in characters.

Parameter	Type	Direction	Description
siteIsDigitRequired	boolean	out	Specifies whether passwords must include at least one digit.
siteIsMixedCaseRequired	boolean	out	Specifies whether passwords must include at least one uppercase letter and at least one lowercase letter.
siteSizeHistoryList	int	out	Specifies the number of previous passwords that are required to be saved before a password value can be reused.
sitePasswordChangeDelayInMinutes	int	out	Specifies the number of minutes that must elapse between password changes.
siteExpirationDays	int	out	Specifies the number of days after a password is set that the password expires.
siteNumFailuresForLockout	int	out	Specifies the number of consecutive unsuccessful logon attempts after which an account to be locked.
siteLockoutInMinutes	int	out	Specifies the number of minutes for which an account is locked following excessive login failures.
siteDaysToSuspension	int	out	Specifies the number of days after which an unused account is suspended.

Details

Parameters are holders for receiving output values, and all parameters are required. That is, the caller must specify all parameters to get values back. A caller cannot leave any variables empty to indicate that he or she doesn't want a value for that parameter.

Exceptions Thrown

The GetInternalLoginSitePolicies method does not return any exceptions.

Examples

The following is a Java example of a GetInternalLoginSitePolicies method call:

```
IntHolder siteMinPasswordLength = new IntHolder();
BooleanHolder siteIsDigitRequired = new BooleanHolder();
BooleanHolder siteIsMixedCaseRequired = new BooleanHolder();
IntHolder siteSizeHistoryList = new IntHolder();
IntHolder sitePasswordChangeDelayInMinutes = new IntHolder();
IntHolder siteExpirationDays = new IntHolder();
IntHolder siteNumFailuresForLockout = new IntHolder();
IntHolder siteLockoutInMinutes = new IntHolder();
IntHolder siteDaysToSuspension = new IntHolder();

iSecurity.GetInternalLoginSitePolicies( siteMinPasswordLength,
                                        siteIsDigitRequired,
                                        siteIsMixedCaseRequired,
                                        siteSizeHistoryList,
                                        sitePasswordChangeDelayInMinutes,
                                        siteExpirationDays,
                                        siteNumFailuresForLockout,
                                        siteLockoutInMinutes,
                                        siteDaysToSuspension );
```

Related Methods

- “GetInternalLoginUserInfo Method” on page 176
- “SetInternalLoginUserOptions Method” on page 189

GetInternalLoginUserInfo Method

Short Description

Gets availability information and internal authentication settings for the specified user.

Category

ISecurity interface internal authentication methods

Interface Version

ISecurity 1.1

Syntax

```
GetInternalLoginUserInfo(personName,hasInternalLogin,isDisabled,bypassStrength,
bypassHistory,useStdExpirationDays,expirationDays,bypassLockout,
bypassInactivitySuspension,doesAccountExpire,accountExpirationDate,
lastPasswordChange,lastLogin,numFailuresSinceLogin,
lastLockout,isLockedOut,isExpired,isSuspend,isAccountExpired);
```

Parameters

Table 8.10 *Method Parameters*

Parameter	Type	Direction	Description
personName	string	in	Specifies the Name attribute value of the Person object for which the InternalLogin is defined. Unlike in other security methods, the Name value is specified as simply*Name*.
hasInternalLogin	boolean	out	Returns T or F, indicating whether the specified user has an InternalLogin object defined.
bypassStrength	boolean	out	Returns T or F, indicating whether a custom password complexity policy is defined.
bypassHistory	boolean	out	Returns T or F, indicating whether a custom history requirement is defined.
useStdExpirationDays	boolean	out	Returns T or F, indicating whether the InternalLogin has a password expiration date.
expirationDays	int	out	Specifies the expiration period.
bypassLockout	boolean	out	Returns T or F, indicating whether a custom lockout policy is defined.

Parameter	Type	Direction	Description
bypassInactivitySuspension	boolean	out	Returns T or F, indicating whether a custom inactivity suspension policy is defined.
doesAccountExpire	boolean	out	Returns T or F, indicating whether the internal account has an expiration date.
accountExpirationDate	datetime	out	Returns the account expiration date if one is defined.
lastPasswordChange	datetime	out	Returns a datetime value, indicating when the password was last changed.
lastLogin	datetime	out	Returns a datetime value, indicating the last time the login was used.
numFailuresSinceLogin	int	out	Returns a number, indicating the number of unsuccessful login attempts since the last successful login.
lastLockout	datetime	out	Returns a datetime value, indicating the last time the account was locked because of consecutive unsuccessful login attempts.
isLockedOut	boolean	out	Returns T or F, indicating whether the account is currently locked because of login failures.
isExpired	boolean	out	Returns T or F, indicating whether the password is currently expired.
isSuspended	boolean	out	Returns T or F, indicating whether the account is currently suspended because of inactivity.
isAccountExpired	boolean	out	Returns T or F, indicating whether the account is currently expired.

Details

Except for PERSONNAME, parameters are holders for receiving output values.

If an internal user account suddenly becomes unavailable, use the GetInternalLoginUserInfo method to determine why the account is unavailable. In addition to returning the specified Person object's internal authentication policy settings, output parameters indicate whether the account is active, disabled, expired, locked out because of unsuccessful authentication, or suspended because of inactivity.

Exceptions Thrown

The GetInternalLoginUserInfo method does not return any exceptions.

Examples

The following is a Java example of a GetInternalLoginUserInfo method call:

```
// Assumes a Person object with Name='Test1' exists
// and has an InternalLogin object associated with it
String personName ="Test1";
BooleanHolder hasInternalLogin = new BooleanHolder();
BooleanHolder isDisabled  = new BooleanHolder();
BooleanHolder bypassStrength  = new BooleanHolder();
BooleanHolder bypassHistory  = new BooleanHolder();
BooleanHolder useStdExpirationDays = new BooleanHolder();
IntHolder expirationDays = new IntHolder();
BooleanHolder bypassLockout = new BooleanHolder();
BooleanHolder bypassInactivitySuspension = new BooleanHolder();
BooleanHolder doesAccountExpire = new BooleanHolder();
DateTimeHolder accountExpirationDate = new DateTimeHolder();
DateTimeHolder lastPasswordChange = new DateTimeHolder();
DateTimeHolder lastLogin = new DateTimeHolder();
IntHolder numFailuresSinceLogin = new IntHolder();
DateTimeHolder lastLockout = new DateTimeHolder();
BooleanHolder isLockedOut = new BooleanHolder();
BooleanHolder isExpired = new BooleanHolder();
BooleanHolder isSuspended = new BooleanHolder();
BooleanHolder isAccountExpired = new BooleanHolder();

GetInternalLoginUserInfo( personName,
                          hasInternalLogin,
                          isDisabled,
                          bypassStrength,
                          bypassHistory,
                          useStdExpirationDays,
                          expirationDays,
                          bypassLockout,
                          bypassInactivitySuspension,
                          doesAccountExpire,
                          accountExpirationDate,
                          lastPasswordChange,
                          lastLogin,
                          numFailuresSinceLogin,
                          lastLockout,
```

```
isLockedOut,
isExpired,
isSuspended,
isAccountExpired );
```

Related Methods

- “SetInternalLoginUserOptions Method” on page 189
- “GetInternalLoginSitePolicies Method” on page 174

GetLoginsforAuthDomain Method

Short Description

Retrieves the logins for the connected user for the specified authentication domain in order of identity precedence.

Category

ISecurity interface generalized authorization methods

Interface Version

ISecurity 1.1

Syntax

```
GetLoginsforAuthDomain(credHandle,authDomain,options,output);
```

Parameters

Table 8.11 *Method Parameters*

Parameter	Type	Direction	Description
credHandle	string	in	A credential handle identifying a user identity, or an empty string.
authDomain	string	in	The name of an AuthenticationDomain, such as DefaultAuth or saspw.
options	string array	in	A two-dimensional string array. Each row contains an option keyword in column zero, and a corresponding value in column one, as described: MaxListLen An integer that indicates the maximum number of logins to return. The default value is 1. IncludeBlankPasswords A value of Yes specifies to include logins that do not have passwords. A value of No (the default value) specifies to exclude logins that do not have passwords. PrimaryOnly A value of Yes specifies to return only logins that are directly associated to the primary identity. A value of No (the default value if this option is omitted) specifies to return logins from group memberships as well. IdentityInfo A value of Yes specifies to also return output columns containing the OwnerName, OwnerType, and OwnerId for the owning identity. A value of No (the default value if this option is omitted) specifies not to return this information.

Parameter	Type	Direction	Description
output	string array	out	A two-dimensional string array in which each row represents the information for a login. The default column values returned are the following: Column 0: UserId Column 1: Password Column 2: ObjectId When IdentityInfo="Yes", also: Column 3: OwnerName Column 4: OwnerType Column 5: OwnerId

Details

CREDHANDLE identifies the user identity for whom logins are being requested. When this value is an empty string, the user identity of the caller is used.

Logins are returned in priority order following identity precedence. For information about identity precedence, see the *SAS Intelligence Platform: Security Administration Guide*.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetLoginsforAuthDomain method:

- InvalidCredHandle
- InvalidOptionName
- InvalidOptionValue
- AuthDomainDoesNotExist

Related Methods

- “GetCredentials Method” on page 164
- “GetInfo Method” on page 168

IsAuthorized Method

Short Description

Determines whether an authenticated user is authorized to access a resource with a specific permission.

Category

ISecurity interface authorization methods

Interface Version

ISecurity 1.0

Syntax

```
IsAuthorized(credHandle,resource,permission,permissionCondition,authorized);
```

Parameters

Table 8.12 *Method Parameters*

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle identifying a user identity, or an empty string.
resource	string	in	Passed resource identifier.
permission	string	in	Passed user access permission.
permissionCondition	string	out	Returned permission conditions associated with access to the resource.

Parameter	Type	Direction	Description
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, authorization is returned for the requesting user.

The RESOURCE parameter identifies the object to which access is requested. The parameter accepts two types of input:

- A URN that specifies an application element in the following form:

  `OMSOBJ: MetadataType/ObjectId`

- A URN that specifies a repository in the following form:

  `REPOSID:_reposID`

  _reposID is the unique, 8-character identifier of a repository. (This is the 8 characters following the period in a RepositoryBase object's 17–character metadata identifier.)

Use of a repository URN causes the IsAuthorized method to check the specified repository's default ACT for information to make the authorization decision. The repository ACT controls whether a user can create objects in the repository. A client can use the URN to determine whether the user represented by the CREDHANDLE parameter is granted or denied WriteMetadata, which determines whether the user can create objects in the repository. Group memberships are evaluated when making the decision. For example, if the requesting user is not specifically denied WriteMetadata permission in the repository ACT, and a group to which he belongs is granted WriteMetadata permission in the repository ACT, then he is allowed to create objects in the repository. For more information about identity precedence, see *SAS Intelligence Platform: Security Administration Guide*.

The PERMISSION parameter specifies the permission to check for. A single permission value can be passed to the IsAuthorized method.

The PERMISSIONCONDITION parameter is used with data permissions, such as Read and Write. A value returned in this parameter indicates that a permission is granted, but only if the condition specified in an associated PermissionCondition object is met. The syntax of a permission condition is not defined. It is specific to the resource being protected and to the technology responsible for enforcing the security of the resource. For example, a PermissionCondition object for a table would contain an SQL WHERE clause, but for an OLAP dimension, it would contain an MDX expression identifying the level members that can be accessed in the OLAP dimension.

It is possible for a user to have multiple permission conditions associated with his or her access to a resource. In this case, the PERMISSIONCONDITION parameter is returned with multiple strings embedded. Each embedded condition is separated from the preceding condition by the string <!--CONDITION-->. If you receive a PERMISSIONCONDITION output string, you must check to see whether it contains

multiple permission conditions by searching for <!--CONDITION--> in the returned string. If multiple permission conditions are found, then they should be used to filter data so the resulting data is a union of the data returned for each permission condition individually. In other words, the permission conditions would have the OR operation performed on them.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the isAuthorized method:

- NotTrustedUser
- InvalidCredHandle
- InvalidResourceSpec

Example

The following is a Java example of the IsAuthorized method. The method is issued to determine whether the credentialed user has Read permission to the requested table. The example includes code that formats and prints the results of the request.

```
  public void isAuthorized() throws Exception {

          try
          {
                  System.out.println("");
                 // Skip a line
                  System.out.println("<<<< Begin isAuthorized() >>>>" );

                  // These statements define holders for the credHandle,
                  // permissionCondition, and authorized parameters. Assume the
                  // requested resource, a table, was defined earlier. Also
                  // that a credential handle was obtained earlier.
                  StringHolder credHandle = new StringHolder();
                  StringHolder permCond = new StringHolder();
                  BooleanHolder isAuth = new BooleanHolder();

                   // Issues the isAuthorized method specifying the Read permission.
                  iSecurity.IsAuthorized(credHandle.value, table_URN, "Read",
permCond, isAuth);

                  System.out.println();
                  // Specify a title for the output and to print parameter
                  // values along with the isAuthorized result.
                  System.out.println("<<<<<< isAuthorized() call parameters with
(Read Permission) results >>>>>>");
                  System.out.print("credHandle=" + credHandle.value + ", ");
                  System.out.print("resourceURN=" + table_URN + ", ");
                  System.out.print("permission=Read, ");
                  System.out.print("permissonCondition=" + permCond.value + ", ");
                  System.out.print("isAuth=" + isAuth.value);
```

```
                System.out.println();
               // force NewLine

                System.out.println("<<<< End isAuthorized() >>>>" );
        }
        // The following statement catches the method's exceptions.
        catch (Exception e) {
                System.out.println("IsAuthorized: Exceptions");
                e.printStackTrace();
                throw e;
        }

  }
```

Here is the output from the request:

```
<<<< Begin isAuthorized() >>>>

<<<<<< isAuthorized() call parameters with (Read Permission) results >>>>>>
credHandle=1e11e9ff00000002, resourceURN=OMSOBJ:PhysicalTable/A5CY5BIY.AO000003,
permission=Read, permissonCondition=Based on this condition, isAuth=true

<<<< End isAuthorized() >>>>
```

The user represented by the credential handle has Read permission to PhysicalTable A5CY5BIY.AO000003.

Related Methods

- "GetAuthorizations Method" on page 156

IsInRole Method

Short Description

Returns the TRUE value when the user specified in CREDHANDLE is in a role.

Category

ISecurity interface generalized authorization methods

Interface Version

ISecurity 1.1

Syntax

```
IsInRole(credHandle,roleSpec,options,inRole);
```

Parameters

Table 8.13 *Method Parameters*

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle identifying a user identity, or an empty string.
roleSpec	string	in	A role specification in one of the following forms: ROLE_OBJNAME : *Role-Object-Name* ROLE_OBJID: *Role-Object-Identifier*
options	string array	in	Two-dimensional string array for options. No options are currently defined.
inRole	C	out	A Boolean value indicating whether the user is in the specified role. TRUE - User is in the specified role. FALSE - User is not in the specified role.

Details

The IsInRole method determines whether a user is in the specified role. The role is identified by the value in the Role object's Name attribute or by its metadata object identifier.

This method is most appropriate for static role implementations.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the IsInRole method:

- NotTrustedUser
- InvalidCredHandle
- InvalidResourceSpec

Examples

The following is a Java example of the IsInRole method. The method requests to know whether the Person object named testUser has membership in the User and Group Administrators role. The example includes code that formats and prints the results of the request.

```
public void isInRole() throws Exception {

   try
   {
    // Define a two-dimensional string array for options
       final String[][] options ={{"",""}};

         System.out.println("");
            // Skip a line
         System.out.println("<<<< Begin isInRole() >>>>" );

         // Define a holder for the credential handle
       StringHolder credHandle = new StringHolder();
       // Define a holder for the method output
         BooleanHolder inRole = new BooleanHolder();

         // Get a credential handle
       iSecurity.GetCredentials(testUserId, credHandle);
       // Execute the method
        iSecurity.IsInRole(
          credHandle.value,
          "ROLE_OBJNAME:" + UGAdminRole,
          options,
          inRole
          );

         // Print information about the method call and results
       System.out.println();
         System.out.println("<<<<<< isInRole() call parameters with results >>>>>>");
         System.out.print("credHandle=" + credHandle.value + ", ");
         System.out.print("roleSpecification=" + "ROLE_OBJNAME:" + UGAdminRole + ", ");
         System.out.print("isInRole=" + inRole.value);
         System.out.println();
            // force NewLine
```

```
          // Free the credentials
            iSecurity.FreeCredentials(credHandle.value);

            System.out.println("");
               // Skip a line
            System.out.println("<<<< End isInRole() >>>>" );
    }
      // Catch the method's exceptions.
          catch (Exception e) {
         System.out.println("IsInRole: GetInfo: Exceptions");
         e.printStackTrace();
         throw e;
        }

}
```

Here is the output from the request:

```
<<<< Begin isInRole() >>>>

<<<<<< isInRole() call parameters with results >>>>>>
credHandle=71cedcb300000001, roleSpecification=ROLE_OBJNAME:META: User
and Group Administrators Role, isInRole=true

<<<< End isInRole() >>>>
```

Related Methods

- "GetApplicationActionsAuthorizations Method" on page 153

SetInternalLoginUserOptions Method

Short Description

Customizes internal authentication policies for the specified user.

Category

ISecurity interface internal authentication methods

Interface Version

ISecurity 1.1

Syntax

```
SetInternalLoginUserOptions(personName,isDisabled,bypassStrength,bypassHistory,
useStdPasswordExpirationDate,passwordExpirationDays,bypassLockout,
bypassInactivitySuspension,expireAccount,accountExpirationDate);
```

Parameters

Table 8.14 *Method Parameters*

Parameters	Type	Direction	Description
personName	string	in	Specifies the Name attribute value of the Person object whose InternalLogin object will be modified. The Name value is specified as simply *Name*.
isDisabled	boolean	in	Specifies whether the account is disabled. To disable the account, specify T. The default value is F.
bypassStrength	boolean	in	Specifies whether to exempt the login from the site's policies about minimum password length and complexity. To exempt the login, specify T. The default value is F.
bypassHistory	boolean	in	Specifies whether to exempt the login from the site's password history policy. To exempt the login, specify T. The default value is F.
useStdPasswordExpirationDays	boolean	in	Specifies whether to enforce a password expiration period. The default value is T. Specify F if you do not want the password to expire.

Parameters	Type	Direction	Description
passwordExpirationDays	integer	in	Specifies the password expiration period in days from the day the password was initially set. A number from 0 to 32767 is supported. The default password expiration period is 30 days.
bypassLockout	boolean	in	Specifies whether to exempt the login from the site's account lockout policy. The default value is F.
bypassInactivitySuspension	boolean	in	Specifies whether to exempt the login from the site's inactivity suspension policy. The default value is F.
expireAccount	boolean	in	Specifies whether to enforce an expiration date on the account. To enforce an expiration date, specify T. The default value is F.
accountExpirationDate	int	in	Specifies the number of days from the day the account was created that the account will expire. A number from 0-32767 is supported. The default value is 0.

Details

You must have user administration capabilities on the SAS Metadata Server to modify the properties of an internal user account. For information about user administration capabilities, see "Users, Groups, and Roles: Main Administrative Roles" in the *SAS Intelligence Platform: Security Administration Guide*.

An internal account has a Person object with a simple name value. For example, Name="Joe". It also has an associated InternalLogin object, whose Name attribute is *person*@saspw. For example, Name="Joe@saspw." All SAS internal accounts must use the suffix @saspw.

The Person object is created with the AddMetadata method. Its attributes are modified with the UpdateMetadata method. An InternalLogin object is created with the SetInternalPassword method. Its attributes are modified with the SetInternalLoginUserOptions method.

By default, new InternalLogin objects are created with the active server-level internal account policies. The active server-level account policies are the system defaults as modified by omaconfig.xml options. The SetInternalLoginUserOptions method enables you to customize the server-level policies for a particular internal account.

For information about system defaults, see “How to Change Internal Account Policies” in the *SAS Intelligence Platform: Security Administration Guide*. To determine what the active policy settings are after the omaconfig.xml options are applied, use the GetInternalLoginSitePolicies method.

New InternalLogin objects are created with a 30–day password expiration period. If you change the USESTDPASSWORDEXPIRATIONDAYS parameter to F, then the password does not expire and the integer value in passwordExpirationDays is ignored.

To view the policy settings on an existing internal account, use the GetInternalLoginUserInfo method. The GetInternalLoginUserInfo method also reports the status of the internal account. For example, returned values indicate whether the account is active, disabled, locked out because of unsuccessful authentication, or suspended because of inactivity.

Exceptions Thrown

The SetInternalLoginUserOptions method does not return any exceptions.

Examples

The following is a Java example of a SetInternalLoginUserOptions method call:

```
// Assumes a Person object with Name='testId' already exists
// and has an InternalLogin object associated with it
iSecurity.SetInternalLoginUserOptions( testId, // username
                                       false,      // isDisabled
                                       false,    // bypassStrength
                                       true,       // bypassHistory
                                       false,    // useStdPasswordExpirationDays
                                       30,    // passwordExpirationDays
                                       false,      // bypassLockout
                                       true,       // bypassInactivitySuspension
                                       false,    // expireAccount
                                       0           // accountExpirationDate );
```

Related Methods

- “GetInternalLoginSitePolicies Method” on page 174
- “GetInternalLoginUserInfo Method” on page 176
- “DeleteInternalLogin Method” on page 150

SetInternalPassword Method

Short Description

Creates an InternalLogin object for the specified user.

Category

ISecurity interface internal authentication methods

Interface Version

ISecurity 1.1

Syntax

```
SetInternalPassword(personName,passwordValue);
```

Parameters

Table 8.15 Method Parameters

Parameter	Type	Direction	Description
personName	string	in	Specifies the Name attribute value of the Person object for which the InternalLogin object will be created. Person objects that are used for internal accounts have a one-word name, and are identified by this name.
passwordValue	string	in	A password that meets the site's password authentication policies.

Details

You must have user administration capabilities on the SAS Metadata Server to create an InternalLogin object. For information about user administration capabilities, see “Users, Groups, and Roles: Main Administrative Roles” in the *SAS Intelligence Platform: Security Administration Guide*.

Internal logins are not intended for regular users. They are intended for metadata administrators and some service identities. For more information, see “SAS Internal Authentication” in the *SAS Intelligence Platform: Security Administration Guide*.

The SetInternalPassword method creates an InternalLogin object and associates it with the specified Person object. Together, the two objects define an internal account. The new InternalLogin object is created with the site’s internal authentication policies. To determine what the active policy settings are, use the GetInternalLoginSitePolicies method. Or, use the GetInternalLoginUserInfo method to list the new object’s properties.

New InternalLogin objects are created with a 30-day password expiration period. To deactivate the password expiration period or customize its length, or to customize other internal authentication settings, use the SetInternalLoginUserOptions method. If the ExpirePasswordonReset option is set in the site’s omaconfig.xml file, the user will have to reset the initial password before the internal account can be used.

Exceptions Thrown

The SetInternalPassword method does not return any exceptions.

Examples

The following is a Java example of a SetInternalPassword method call:

```
// Defines parameters personName and passwordValue assuming
// a Person object with Name='testId' already exists
String personName = "testId";
String passwordValue = "pw1234";

iSecurity.SetInternalPassword(personName,passwordValue);
```

Related Methods

- “GetInternalLoginSitePolicies Method” on page 174
- “GetInternalLoginUserInfo Method” on page 176
- “SetInternalLoginUserOptions Method” on page 189

9

Security Administration (ISecurityAdmin Interface)

Overview of the ISecurityAdmin Server Interface

The methods described in this section are provided in the ISecurityAdmin server interface, and can be used in a SAS Open Metadata Interface client that you create to administer authorizations on metadata resources and to manage ACTs.

ISecurityAdmin methods are available only in the standard interface. For more information, see “Communicating with the SAS Metadata Server” on page 14.

ISecurityAdmin contains three categories of methods:

- Transaction context methods enable programmers of interactive clients to record user interactions and return correct effective permissions for authorization changes, factoring in group memberships, before applying the changes to authorization metadata on the SAS Metadata Server. The BeginTransactionContext method creates a transaction context by returning a handle for a specified object. General authorization administration methods

reference this handle in their requests. The transaction context is closed by using the EndTransactionContext method, which can commit or discard the changes.

- General authorization administration methods enable programmers to easily set and get authorizations on resources, list authorized identities on resources, and apply and remove ACTs from resources.
- ACT administration methods create, modify, list, and destroy ACTs.

The following information applies to all of the ISecurityAdmin methods.

- Errors are surfaced through exception-handling in IOM. Each method returns a set of documented exceptions. Use TRY and CATCH logic in your Java program to determine when an exception is returned. If your client does not need to handle specific exceptions for an ISecurityAdmin method, then the generic Java exception might be caught.
- The methods define and get authorizations on user and resource metadata that is defined in SAS Metadata Repositories. 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 with import macros. Resource metadata can be created with the SAS Java Metadata Interface or other SAS Open Metadata Architecture clients.
- The requesting user must have ReadMetadata permission on the target resource to use ISecurityAdmin methods that read access control information. The requesting user must have ReadMetadata and WriteMetadata permissions on the target resource to use ISecurityAdmin methods that modify access control information. These methods include SetAuthorizationsOnObj(), ApplyAccessControlTemplateToObj(), RemoveAccessControlTemplateFromObj(), DestroyAccessControlTemplate(), and SetAccessControlTemplateAttribs(). The requesting user must have WriteMetadata permission on the default ACT of the specified repository to use CreateAccessControlTemplate().
- In the examples, iSecurityAdmin is an instantiation of the ISecurityAdmin interface.

Using the ISecurityAdmin Server Interface

Calling the Server Interface

The ISecurityAdmin interface is called by connecting to the SAS Metadata Server and obtaining a handle to the ISecurityAdmin server interface.

A SAS Java Metadata Interface client accesses the ISecurityAdmin interface by importing the appropriate packages, instantiating an object factory, and connecting to the SAS Metadata Server with a handle to the interface that is appropriate for the task that it wants to perform.

The ISecurityAdmin interface is provided in the sas.oma.omi.jar file in the SAS Platform VJR. A Java client accesses the ISecurityAdmin interface by importing the appropriate com.sas.meta.SASOMI packages. Import

com.sas.meta.SASOMI.ISecurityAdmin, com.sas.meta.SASOMI.ISecurityAdminPackage, and com.sas.metadata.remote into your client.

The SAS Java Metadata Interface provides the MdFactory interface to instantiate an object factory for the SAS Metadata Server and the MdOMRConnection interface for connecting to the SAS Metadata Server. Use the MdOMRConnection interface's makeISecurityAdminConnection method to connect to the server with the ISecurityAdmin interface.

Identifying Resources to ISecurityAdmin Methods

ISecurityAdmin general authorization administration and ACT administration methods can be issued on a transaction context, or directly on a resource.

When you specify a transaction context, do not include a target resource identifier in the method call, unless you are issuing the EndTransactionContext method. The changes that are requested by ISecurityAdmin methods that are invoked with a transaction context are persisted to the metadata server (or discarded) with the EndTransactionContext method. An EndTransactionContext method call that persists changes to the SAS Metadata Server specifies a transaction context handle, the resource on which to apply the changes indicated in the handle, and sets the SECAD_COMMIT_TC flag.

Changes that are requested by an ISecurityAdmin method that specifies a resource identifier are persisted on the SAS Metadata Server immediately. A resource is identified with a URN. For more information about URNs, see "Identifying Resources to ISecurity Methods" on page 147.

Understanding the Transaction Context Methods

With interactive clients, a well-defined set of interactions between the client and server are required to support evaluating and changing Permission values for an object. To facilitate these tasks, a server-side transaction context is now supported to maintain state during client requests. ISecurityAdmin methods that get or set Permission values can request a handle for a transaction context. This transaction context is a server-side structure that tracks incremental Permission changes, so that IdentityGroup memberships can be factored in for the client. For example, in SAS Management Console, in the Authorization tab of a resource's Properties window, a user can select grant or deny on different permissions for the identities displayed. The state of all currently persisted and effective Permission values, along with incremental Permission changes pending from selections in the GUI, are maintained by the transaction context on the server. If the user clicks OK, the client commits the changes on the SAS Metadata Server. If the user clicks Cancel, the changes in the transaction context are discarded.

A transaction context is created by using the BeginTransactionContext method. It is committed or discarded by using the EndTransactionContext method. For more information, see "BeginTransactionContext Method" on page 204. Also see "EndTransactionContext Method" on page 211.

Understanding the General Authorization Administration Methods

The general authorization administration methods set and get authorizations on metadata resources. An authorization associates an identity, a permission, and a grant or denial of that permission with a resource. The authorizations can be set directly on a resource, or applied to the resource in an ACT. Authorizations can also be set on ACTs to control who is authorized to modify the ACT.

The general authorization administration methods include the following:

ApplyACTToObj
: Applies the authorizations defined in an ACT to the specified resource. For more information, see “ApplyACTToObj Method” on page 201.

GetAccessControlTemplatesOnObj
: Lists the ACTs that are associated with a resource. For more information, see “GetAccessControlTemplatesOnObj Method” on page 213.

GetAuthorizationsOnObj
: Returns the authorizations that apply to a resource for specified identities and permissions. For more information, see “GetAuthorizationsOnObj Method” on page 220.

GetIdentitiesOnObj
: Returns Person, IdentityGroup, and Role objects associated with a specified resource. For more information, see “GetIdentitiesOnObj Method” on page 225.

RemoveACTFromObj
: Removes the authorizations defined by an ACT from the specified resource. For more information, see “RemoveACTFromObj Method” on page 228.

SetAuthorizationsOnObj
: Sets permissions for identities on a resource. For more information, see “SetAuthorizationsOnObj Method” on page 232.

Understanding the ACT Administration Methods

The ACT administration methods create and manage ACTs. They cannot be used to add or modify authorizations in an ACT. To modify the authorizations in an ACT, use the “SetAuthorizationsOnObj Method” on page 232 method.

The ACT administration methods include the following:

CreateAccessControlTemplate
: Creates an ACT. For more information, see “CreateAccessControlTemplate Method” on page 206.

DestroyAccessControlTemplate
: Destroys an ACT and removes references to it from all associated objects. For more information, see “DestroyAccessControlTemplate Method” on page 208.

GetAccessControlTemplateAttribs
: Retrieves the attributes of an ACT. For more information, see “GetAccessControlTemplateAttribs Method” on page 215.

GetAccessControlTemplateList
: Lists all ACTs in the specified repository or in all public repositories. For more information, see “GetAccessControlTemplateList Method” on page 216.

SetAccessControlTemplateAttribs
: Changes the attributes of an ACT. For more information, see “SetAccessControlTemplateAttribs Method” on page 231.

ApplyACTToObj Method

Short Description

Applies the authorizations defined in an ACT to the specified resource.

Category

ISecurityAdmin general authorization administration methods

Syntax

```
ApplyACTToObj(tCtxt,resource,flags,ACTresource);
```

Parameters

Table 9.1 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.

Parameter	Type	Direction	Description
resource	string	in	Optional resource identifier of the object to which the ACT should be applied. If TCTXT is used, do not specify a value in RESOURCE.
flags	int	in	Currently unused. Callers should set this parameter to 0.
ACTresource	string	in	Passed resource identifier of an ACT.

Details

The ACT must exist before you can apply it with the ApplyACTToObj method. You can create an ACT with the CreateAccessControlTemplate method.

When TCTXT is set to a valid value, the permanent application of the ACT is deferred until the EndTransactionContext method is invoked on a resource with the SECAD_COMMIT_TC flag. However, a subsequent call to the GetAccessControlTemplatesOnObj method with the TCTXT value returns the applied ACT for the object represented by TCTXT.

When TCTXT is null and RESOURCE is set to a valid value, the ACT is applied to the specified resource immediately by the SAS Metadata Server.

The method fails if the caller does not have WriteMetadata permission on the target resource, and ReadMetadata permission for the ACT being applied.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the ApplyACTToObj method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_INVALID_ACTION
- SECAD_OBJECT_NOT_ACT
- SECAD_ACT_DOES_NOT_EXIST
- SECAD_ACT_IN_DEPENDENT_REPOSITORY
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the ApplyACTToObj method is issued in a Java environment:

```
public void applyAccessControlTemplateToObj(String transCtxt, String resource,
int options, String ACTspec ) throws Exception {

    try
    {
        SecurityAdmin.ApplyACTToObj(transCtxt, resource, options, ACTspec);
    }
           catch (Exception e)
       {
             System.out.println("ApplyACTToObj: Exceptions");
             e.printStackTrace();
             throw e;
       }
}
```

The following example issues the ApplyACTToObj method to apply a predefined ACT to an existing Tree object that represents a folder. The ACT is identified by ACTSPEC:

```
public void ApplyACTToObj() throws Exception {
       // Define an object variable for the Tree
      Tree_URN = "OMSOBJ:Tree/metadata-identifier";

     // Apply the ACT to a Tree. Because resource is used, the tCtxt parameter is null.
     iSecurityAdmin.ApplyAccessControlTemplateToObj("",Tree_URN, 0, ACTspec);

     //If we had submitted a tCtxt value, resource would be null.

         }
        catch (Exception e)
        {
                throw e;
        }
}
```

Related Methods

- “CreateAccessControlTemplate Method” on page 206
- “RemoveACTFromObj Method” on page 228
- “DestroyAccessControlTemplate Method” on page 208

BeginTransactionContext Method

Short Description

Creates a transaction context for an authorization request.

Category

ISecurityAdmin interface transaction context methods

Syntax

```
BeginTransactionContext(resource,flags,tCtxt);
```

Parameters

Table 9.2 Method Parameters

Parameter	Type	Direction	Description
resource	string	in	Passed resource identifier for the object for which a transaction context is to be started, or an empty string.
flags	int	in	Currently unused. Callers should set this parameter to 0.
tCtxt	string	out	Returned handle representing a server-side transaction context.

Details

The BeginTransactionContext method gets a transaction context for the metadata object specified in RESOURCE. A handle to the new transaction context is returned

in the output TCTXT parameter. Use this handle to identify the pertinent transaction context in general authorization administration methods and ACT administration methods.

If the target resource is not immediately known, submit the method with an empty string in the RESOURCE parameter. You can identify the target resource for the authorization changes in the EndTransactionContext method.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the BeginTransactionContext method:

- SECAD_INVALID_RESOURCE_SPEC
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the BeginTransactionContext method is issued in a Java environment:

```
public void beginTransactionContext (String obj, StringHolder tCtxt) throws Exception
    {
        try
        {
            iSecurityAdmin.BeginTransactionContext (obj, 0, tCtxt);
        }
        catch (Exception e)
        {
            System.out.println("BeginTransactionContext: Exceptions");
            e.printStackTrace();
            throw e;
        }
}
```

The following example issues the BeginTransactionContext method to begin a transaction context on a preexisting Tree object defined by the object variable Tree_URN:

```
// Define a holder for the transaction context
 StringHolder tCtxt = new StringHolder();

       // Begin a transaction context on the Tree object.
       iSecurityAdmin.BeginTransactionContext(Tree_URN, tCtxt);
```

Related Methods

- “EndTransactionContext Method” on page 211

CreateAccessControlTemplate Method

Short Description

Creates an ACT.

Category

ISecurityAdmin interface ACT administration methods

Syntax

```
CreateAccessControlTemplate(tCtxt,REPOSresource,ACT_attributes);
```

Parameters

Table 9.3 Method Parameters

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
REPOSresource	string	in	Optional resource identifier for the repository in which the ACT is created. If TCTXT is used, do not specify a value in RESOURCE.
ACT_attributes	string array	in	Passed two-dimensional string array that defines ACT attributes in two columns. Column 1 specifies the attribute name. Column 2 specifies the attribute value.

Details

The CreateAccessControlTemplate method creates an ACT object. You must use the SetAuthorizationOnObjs method with the SETACTCONTENTS parameter set to TRUE to add or remove authorizations on the ACT object.

Only the Name attribute is required to be defined in ACT_ATTRIBUTES to create an ACT object. The Name value must be unique in the target repository.

Two other attributes are supported in ACT_ATTRIBUTES:

Desc="*text*"
: Specifies a description of the ACT. A string up to 200 characters is supported.

Use="null | REPOS"
: Specifies an empty string or the value REPOS. An empty string indicates the ACT is applied to one or more objects in the repository. The value REPOS sets the ACT as the repository default ACT.

To change the attributes of an existing ACT, use the SetAccessControlTemplateAttribs method.

TCTXT identifies an optional transaction context in which to execute the request. When TCTXT is null, the ACT is immediately persisted to the SAS Metadata Server instead of being cached in a transaction context.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the CreateAccessControlTemplate method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_REPOS_SPEC
- SECAD_ACT_ALREADY_EXISTS
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the CreateAccessControlTemplate method is issued in a Java environment:

```
public void createAccessControlTemplate(String transCtxt, String repository,
String[][] ACTattributes ) throws Exception {

        try
        {
    iSecurityAdmin.CreateAccessControlTemplate(transCtxt, repository, ACTattributes);
        }
        catch (Exception e) {
            System.out.println("CreateAccessControlTemplate: Exceptions");
```

```
            e.printStackTrace();
            throw e;
        }
}
```

The following example issues the CreateAccessControlTemplate method to create an ACT in the repository defined in REPOSRESOURCE:

```
public void createAccessControlTemplate() throws Exception {

// Name and Desc values for ACT
        final String[][] ActAttribs =
        {
                {"NAME", testUserACTname},
                {"DESC", "ACT to project testUser's resources"}
        };
 // Repository in which the ACT will be created
                StringHolder REPOSresource = new StringHolder(REPOSID:_reposid);
        try {
              iSecurityAdmin. createAccessControlTemplate("", REPOSresource.value,
ActAttribs);
                 }
        catch (Exception e ){
                 throw e;
        }
}
```

Related Methods

- “SetAuthorizationsOnObj Method” on page 232
- “SetAccessControlTemplateAttribs Method” on page 231
- “DestroyAccessControlTemplate Method” on page 208

DestroyAccessControlTemplate Method

Short Description

Destroys an ACT and removes references to it from all associated objects.

Category

ISecurityAdmin interface ACT administration methods

Syntax

```
DestroyAccessControlTemplate(tCtxt,ACTresource);
```

Parameters

Table 9.4 Method Parameters

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
ACTresource	string	in	Optional resource identifier of an ACT object. If TCTXT is used, do not specify a value in ACTRESOURCE.

Details

The DestroyAccessControlTemplate method destroys an ACT object and removes all references to it from all associated objects.

When TCTXT is set to a valid value, the destruction of the ACT and removal of its references is deferred until the EndTransactionControl method is invoked on a resource with the SECAD_COMMIT_TC flag.

When TCTXT is null and ACTRESOURCE is set to a valid value, the ACT is destroyed immediately, along with all references. For instructions on how to format a URN for the ACTRESOURCE parameter, see “Using the ISecurityAdmin Server Interface” on page 198.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the DestroyAccessControlTemplate method:

- OK
- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_OBJECT_NOT_ACT
- SECAD_ACT_DOES_NOT_EXIST

- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the DestroyAccessControlTemplate method is issued in a Java environment:

```
public void destroyAccessControlTemplate(String transCtxt, String
ACTSpec)
 throws Exception {

        try
        {
            iSecurityAdmin.DestroyAccessControlTemplate(transCtxt, ACTSpec);
        }
        catch (Exception e) {
            System.out.println("DestroyAccessControlTemplate: Exceptions");
            e.printStackTrace();
            throw e;
        }
}
```

The following example destroys the ACT identified by ACTSPEC:

```
public void termACT() throws Exception {

      try {
          iSecurityAdmin.destroyAccessControlTemplate("", ACTspec);
      }
      catch (Exception e)
      {
          throw e;
      }
 }
```

Related Methods

- “CreateAccessControlTemplate Method” on page 206
- “RemoveACTFromObj Method” on page 228

EndTransactionContext Method

Short Description

Terminates the transaction context for the specified TCTXT handle.

Category

ISecurityAdmin interface transaction context methods

Syntax

```
EndTransactionContext(tCtxt,resource,flags);
```

Parameters

Table 9.5 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Passed handle representing a server-side transaction context.
resource	string	in	Passed resource identifier for the object for which the transaction context is to be terminated, if TCTXT was obtained using a null value.
flags	int	in	SECAD_COMMIT_TC Specifies to persist the security changes in the transaction context to the SAS Metadata Server. SECAD_DISCARD_TC Specifies to discard the security changes in the transaction context and cancel the security update.

Details

The EndTransactionContext method terminates the transaction context on the specified resource.

If a valid existing resource was specified when the corresponding BeginTransactionContext() method was called, then RESOURCE should be an empty string.

SECAD_COMMIT_TC and SECAD_DISCARD_TC are symbols representing the valid values for the FLAGS parameter.

- Set SECAD_COMMIT_TC to commit changes in the transaction context before terminating the transaction context. The changes in the transaction context are written to the SAS Metadata Server as authorization metadata objects that are associated with the specified resource.
- Set SECAD_DISCARD_TC to discard the changes.

The caller must have WriteMetadata permission on the target resource to commit the changes to the SAS Metadata Server.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the EndTransactionContext method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_ACTION
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the EndTransactionContext method is issued in a Java environment:

```
public void endTransactionContext (String transCtxt, int options) throws
Exception
    {
        try
        {
            iSecurityAdmin.EndTransactionContext (transCtxt, "", options);
        }
        catch (Exception e)
        {
            System.out.println("EndTransactionContext: Exceptions");
            e.printStackTrace();
            throw e;
        }
}
```

The following example issues an EndTransactionContext method to specify to commit the changes indicated in tCtxt.value:

```
iSecurityAdmin.EndTransactionContext(tCtxt.value, ISecurityAdmin.SECAD_COMMIT_TC);
```

Related Methods

- “BeginTransactionContext Method” on page 204

GetAccessControlTemplatesOnObj Method

Short Description

Lists the ACTs that are associated with a resource.

Category

ISecurityAdmin interface general authorization administration methods

Syntax

```
GetAccessControlTemplatesOnObj(tCtxt,resource,flags,ACT_list);
```

Parameters

Table 9.6 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
resource	string	in	Passed resource identifier for the object for which ACTs are requested. If TCTXT is used, do not specify a value in RESOURCE.

Parameter	Type	Direction	Description
flags	int	in	Currently unused. Callers should set this parameter to 0.
ACT_list	string array	out	Returned two-dimensional string array with three columns. Each row in the array represents an ACT. See the “Details” section for more information. Column 0: Contains the ACT metadata object identifier. Column 1: Contains the ACT Name attribute value. Column 2: Contains the ACT Description attribute value.

Details

The GetAccessControlTemplatesOnObj method returns ACT_LIST when TCTXT or RESOURCE is specified, even if there are no ACTs (an empty list is returned).

When TCTXT is specified and previous calls to the ApplyACTToObj or RemoveACTFromObj method on this TCTXT modified the list of ACTs protecting the resource, then the modified list is returned. Until the EndTransactionContext method is executed on the TCTXT with SECAD_COMMIT_TC, the content of ACT_LIST might not reflect the actual ACTs currently protecting the resource.

When RESOURCE is specified in the GetAccessControlTemplatesOnObj method, then ACT_LIST returns the actual ACTs protecting the resource.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetAccessControlTemplatesOnObj method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_INVALID_ACTION
- SECAD_NOT_AUTHORIZED

Related Methods

- “ApplyACTToObj Method” on page 201
- “RemoveACTFromObj Method” on page 228
- “CreateAccessControlTemplate Method” on page 206

GetAccessControlTemplateAttribs Method

Short Description

Retrieves the attributes of an ACT.

Category

ISecurityAdmin interface ACT administration methods

Syntax

```
GetAccessControlTemplateAttribs(tCtxt,ACTresource,ACT_attributes);
```

Parameters

Table 9.7 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
ACTresource	string	in	Passed resource identifier of an ACT object. If TCTXT is used, do not specify a value for ACTRESOURCE.

Parameter	Type	Direction	Description
ACT_attributes	string array	out	Returned two-dimensional string array containing ACT attributes.

Details

The GetAccessControlTemplateAttribs method retrieves the attributes of the specified ACT object. The requested attributes are returned in the ACT_ATTRIBUTES parameter. For a description of the ACT attributes, see “CreateAccessControlTemplate Method” on page 206.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetAccessControlTemplateAttribs method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_OBJECT_NOT_ACT
- SECAD_ACT_DOES_NOT_EXIST
- SECAD_NOT_AUTHORIZED

Related Methods

- “SetAccessControlTemplateAttribs Method” on page 231

GetAccessControlTemplateList Method

Short Description

Lists all ACTs in the specified repository or in all public repositories.

Category

ISecurityAdmin interface ACT administration methods

Syntax

```
GetAccessControlTemplateList(tCtxt,REPOSresource,flags,ACT_list);
```

Parameters

Table 9.8 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
REPOSresource	string	in	Passed resource identifier for the repository from which you want to get ACTs. If TCTXT is used, do not specify a value for REPOSRESOURCE.
flags	int	in	Passed indicator for options. Valid values are SECAD_REPOSITORY_DEPENDENCY_USES or 0. SECAD_REPOSITORY_DEPENDENCY_USES Expands the request to return ACTs from all public repositories (the foundation repository and all custom repositories).

Parameter	Type	Direction	Description
ACT_list	string array	out	Returned two-dimensional string array with five columns. Each row describes an ACT. See the “Details” section for more information. Column 0: Contains the Name attribute value of the repository where the ACT resides. Column 1: Contains the ACT’s 17-character metadata object identifier. Column 2: Contains the ACT’s Name attribute value. Column 3: Contains the ACT’s Description attribute value. Column 4: Contains the ACT’s Use attribute value. Valid values are REPOS, which indicates the ACT is enforced on the repository, or an empty string, which indicates the ACT is enforced on objects within the repository.

Details

The GetAccessControlTemplateList method can return a large number of objects, especially if the SECAD_REPOSITORY_DEPENDENCY_USES flag is set. Use of this method within a transaction context is not recommended because of the overhead associated with the method.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetAccessControlTemplateList method:

- OK
- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the GetAccessControlTemplateList method is issued in a Java environment:

```
public void getAccessControlTemplateList(String transCtxt, String repositorySpec,
int options, VariableArray2dOfStringHolder ACTlist) throws Exception {

    try
    {
        iSecurityAdmin.GetAccessControlTemplateList(transCtxt, repositorySpec,
                    options, ACTlist);
    }
    catch (Exception e) {
        System.out.println("GetAccessControlTemplateList: Exceptions");
        e.printStackTrace();
        throw e;
    }
}
```

The following example lists the ACTs in REPOSID:

```
public void GetAccessControlTemplateList() throws Exception {
// Define a holder for ACTlist
VariableArray2dOfStringHolder ACTlist = new VariableArray2dOfStringHolder;

{
try
{
iSecurityAdmin.GetAccessControlTemplateList("", "REPOSID:" + reposId, 0, ACTlist);
}
catch (Exception e) {
    e.printStackTrace();
    throw e;
 }
}
```

Related Methods

- “GetAccessControlTemplateAttribs Method” on page 215

GetAuthorizationsOnObj Method

Short Description

Returns the authorizations that apply to a resource for specified identities and permissions.

Category

ISecurityAdmin interface general authorization administration methods

Syntax

```
GetAuthorizationsOnObj(tCtxt,resource,flags,identities,permissions,authorizations);
```

Parameters

Table 9.9 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
resource	string	in	Passed resource identifier for the object from which the active permissions are requested. If TCTXT is used, do not specify a value for RESOURCE.

Parameter	Type	Direction	Description
flags	int	in	Passed indicator for optional functionality. SECAD_ACT_CONTENTS When TCTXT or RESOURCE references an ACT, this flag returns the authorizations that are defined in the ACT, rather than authorizations that protect the ACT. SECAD_DO_NOT_RETURN_PERMCOND Omits PermissionCondition values from column 5 of the AUTHORIZATIONS output array, if associated PermissionCondition objects are found. SECAD_RETURN_DISPLAY_NAME Returns the value of each identity's DisplayName attribute in column 6 of the AUTHORIZATIONS output array. SECAD_RETURN_ROLE_TYPE When an IdentityGroup has a GroupType value of Role, this flag returns the word "Role" in column 1 of the AUTHORIZATIONS output array.
identities	string array	in	Passed two-dimensional string array with two columns. Each row in the array specifies an identity for which permissions are to be queried and returned in the array referenced by the AUTHORIZATIONS parameter. If IDENTITIES is empty, then permissions for all associated identities are returned in the AUTHORIZATIONS output array. Column 1: Specify Person, IdentityGroup or Role to indicate the identity type. Column 2: Specify the identity's Name attribute value.
permissions	string	in	Passed string containing zero or more comma-delimited permission names for which authorizations are being queried. If PERMISSIONS is empty, then authorizations on all relevant permissions are returned in the AUTHORIZATIONS output array.

Parameter	Type	Direction	Description
authorizations	any array	out	Returned two-dimensional array with five or six columns. A row is returned for each identity. The order of the rows corresponds to the order of the permissions in the PERMISSIONS parameter. See the “Details” section for more information. Column 0: Contains the value Person, IdentityGroup, or Role, indicating the identity type. Column 1: Contains the Name attribute value of the identity. Column 2: Contains an integer that represents a symbol that indicates Deny or Grant and the origin of the authorization. See the table in the “Details” section for an explanation of the returned values. Column 3: Contains a Permission name. For example, ReadMetadata, WriteMetadata, and so on. Column 4: Contains a PermissionCondition value for the identity and permission, unless the SECAD_DO_NOT_RETURN_PERMCOND flag is set. If this flag is set, the column is empty or contains the results of the SECAD_RETURN_DISPLAY_NAME, if the SECAD_RETURN_DISPLAY_NAME flag is set. Column 5: Contains the DisplayName attribute value of the identity, if the SECAD_RETURN_DISPLAY_NAME flag is set, and the SECAD_DO_NOT_RETURN_PERMCOND flag is not set. If SECAD_DO_NOT_RETURN_PERMCOND is set, the column is empty.

Details

The GetAuthorizationsOnObj method returns authorizations for the resource specified by the TCTXT or RESOURCE parameter.

Grant or denial of a permission for an identity is indicated by an integer in column 2 of the array that is returned in the AUTHORIZATIONS parameter. Nine integer values are supported, which correspond with a symbol that indicates the origin of the authorization and whether the permission is granted. The integer values are described in the following table.

Table 9.10 *Authorization Integer Translation Table*

Integer	Symbol	Permission Type	Description
1	SECAD_PERM_EXPD	Explicit Deny	Deny was specified directly on the object.
2	SECAD_PERM_EXPG	Explicit Grant	Grant was specified directly on the object.
0x03	SECAD_PERM_EXPM	Explicit Mask	Mask to extract explicit value.
4	SECAD_PERM_ACTD	ACT Deny	Deny from an ACT other than the default ACT.
8	SECAD_PERM_ACTG	ACT Grant	Grant from an ACT other than the default ACT.
0x0C	SECAD_PERM_ACTM	ACT Mask	Mask to extract ACT value.
16	SECAD_PERM_NDRD	Indirect Deny	Deny from IdentityGroup inheritance or from the default ACT.
32	SECAD_PERM_NDRG	Indirect Grant	Grant from IdentityGroup inheritance or from the default ACT.
0X30	SECAD_PERM_NDRM	Indirect Mask	Mask to extract indirect value.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetAuthorizationsOnObj method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_INVALID_ACTION
- SECAD_INVALID_IDENTITY_SPEC
- SECAD_IDENTITY_DOES_NOT_EXIST
- SECAD_INVALID_PERMISSION_SPEC
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the GetAuthorizationsOnObj method is issued in a Java environment:

```
public void getAccessControlTemplateList(String transCtxt, String repositorySpec,
int options, VariableArray2dOfStringHolder ACTlist ) throws Exception {

        try
        {
            iSecurityAdmin.GetAccessControlTemplateList(transCtxt,
                                                        repositorySpec,
                                                        options,
                                                        ACTlist
                                                        );
        }
        catch (Exception e) {
            System.out.println("GetAccessControlTemplateList: Exceptions");
            e.printStackTrace();
            throw e;
        }
}
```

The example issues the GetAuthorizationsOnObj method to get the inherited authorizations on a table that is identified by Table_URN.

```
public void testAuthsOnTable() throws Exception {
        try {
        // Get existing authorizations on the table.
        iSecurityAdmin.GetAuthorizationsOnObj("",
                                              Table_URN,
                                              0,
                                              Identities,
                                              Permissions,
                                              authRslt
                                              );
```

```
        }
            catch (Exception e)
              {
        throw e;
             }
}
```

Related Methods

- “SetAuthorizationsOnObj Method” on page 232

GetIdentitiesOnObj Method

Short Description

Returns Person, IdentityGroup, and Role objects associated with a specified resource.

Category

ISecurityAdmin interface general authorization administration methods

Syntax

```
GetIdentitiesOnObj(tCtxt,resource,flags,id_List);
```

Parameters

Table 9.11 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.

Parameter	Type	Direction	Description
resource	string	in	Passed resource identifier for the object for which identities are being queried. If TCTXT is used, do not specify a value for RESOURCE.
flags	int	in	SECAD_ACT_CONTENTS When TCTXT or RESOURCE references an ACT, this flag specifies to return the identities that have permissions defined in the ACT, rather than permissions defined to protect the ACT. SECAD_RETURN_DISPLAY_NAME Returns the value of the DisplayName attribute of each identity. SECAD_RETURN_ROLE_TYPE When a returned IdentityGroup has a GroupType value of Role, this flag returns the word “Role” in column 1 of the ID_LIST output array. SECAD_RETURN_IDENTITY_ORIGIN Returns one or two characters that indicate the origin of each identity. D indicates the origin was a direct ACE or ACT defined on the object. I indicates an inherited identity, or an identity set in the default ACT. DI indicates the identity comes from both direct and inherited origins.

Parameter	Type	Direction	Description
id_List	string array	out	Returned two-dimensional string array of identity values with two to four columns. Each row in the array represents an identity. The content of the columns depends on which flags were set. See the “Details” section for more information. Column 0: Contains the value Person, IdentityGroup or Role, indicating the identity type. Column 1: Contains the Name attribute value of the identity. Column 2: If both the SECAD_RETURN_IDENTITY_ORIGIN and SECAD_RETURN_DISPLAY_NAME flags are set, contains the DisplayName attribute value of the identity. If SECAD_RETURN_DISPLAY_NAME is not set and SECAD_RETURN_IDENTITY_ORIGIN is set, contains a value indicating the origin of the permission. Column 3: Contains a value indicating the origin of an identity’s permission, or is empty, depending on which flags are set in the GetIdentitiesOnObj request.

Details

The GetIdentitiesOnObj method returns Person, IdentityGroup, and Role objects that have permissions defined on a specified resource. Flags can be set to return the identity’s DisplayName value and a value describing the origin of the permission.

When the specified resource is an ACT object, the method lists the identities that are assigned permissions to protect the ACT, unless the SECAD_ACT_CONTENTS flag is set. When this flag is set, the method lists identities that have permissions defined in the ACT.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the GetIdentitiesOnObj method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_INVALID_ACTION
- SECAD_NOT_AUTHORIZED

Related Methods

- “GetAuthorizationsOnObj Method” on page 220

RemoveACTFromObj Method

Short Description

Removes the authorizations defined by an ACT from the specified resource.

Category

ISecurityAdmin interface general authorization administration methods

Syntax

```
RemoveACTFromObj(tCtxt,resource,flags,ACTresource);
```

Parameters

Table 9.12 Method Parameters

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
resource	string	in	Passed resource identifier for the object from which the ACT should be removed. If TCTXT is used, do not specify a value in RESOURCE.
flags	int	in	Currently unused. Callers should set this parameter to 0.
ACTresource	string	in	Resource identifier of an ACT object.

Details

The RemoveACTFromObj method disassociates an ACT from a resource, while leaving the ACT intact. Use the DestroyAccessControlTemplate method to destroy the ACT.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the RemoveACTFromObj method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_INVALID_ACTION
- SECAD_OBJECT_NOT_ACT
- SECAD_ACT_DOES_NOT_EXIST
- SECAD_ACT_NOT_REMOVED
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the RemoveACTFromObj method is issued from a Java environment:

```
public void removeAccessControlTemplateFromObj(String transCtxt, String resource,
 int options, String ACTspec ) throws Exception {

    try
    {
        iSecurityAdmin.RemoveACTFromObj(transCtxt, resource, options, ACTspec);
    }
    catch (Exception e) {
        System.out.println("RemoveACTFromObj: Exceptions");
        e.printStackTrace();
        throw e;
    }
}
```

The following example removes the ACT identified by ACTspec from a Tree object identified by Tree_URN:

```
public void RemoveAccessControlTemplateFromObj() throws Exception {
   try
   {
     // Remove the ACT from the Tree to see the impact on authorizations
    iSecurityAdmin.RemoveAccessControlTemplateFromObj "",Tree_URN, 0, ACTspec);
   }
  catch(Exception e)
     {
    e.printStackTrace();
    throw e;
   }
}
```

Related Methods

- “DestroyAccessControlTemplate Method” on page 208
- “ApplyACTToObj Method” on page 201

SetAccessControlTemplateAttribs Method

Short Description

Changes the attributes of an ACT.

Category

ISecurityAdmin interface ACT administration methods

Syntax

```
SetAccessControlTemplateAttribs(tCtxt,ACTresource,ACT_attributes);
```

Parameters

Table 9.13 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
ACTresource	string	in	Passed resource identifier for an ACT object. If TCTXT is used, do not specify a value in ACTRESOURCE.
ACT_attributes	string array	in	Two-dimensional string array that defines ACT attributes in two columns. Column 1 specifies the attribute name. Column 2 specifies the attribute value.

Details

The SetAccessControlTemplateAttribs method can be used to rename an ACT, modify its description, and add or remove the ACT as the current default repository ACT. These changes are made by specifying new values for the Name, Desc, and Use attributes.

The specified values replace the current values for the ACT identified by TCTXT or in ACTRESOURCE. For information about the attributes, see "CreateAccessControlTemplate Method" on page 206.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the SetAccessControlTemplateAttribs method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_OBJECT_NOT_ACT
- SECAD_ACT_ALREADY_EXISTS
- SECAD_ACT_DOES_NOT_EXIST
- SECAD_NOT_AUTHORIZED

Related Methods

- "CreateAccessControlTemplate Method" on page 206
- "GetAccessControlTemplateAttribs Method" on page 215

SetAuthorizationsOnObj Method

Short Description

Sets permissions for identities on a resource.

Category

ISecurityAdmin interface general authorization administration methods

Syntax

```
SetAuthorizationsOnObj(tCtxt,resource,flags,authorizations);
```

Parameters

Table 9.14 *Method Parameters*

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
resource	string	in	Passed resource identifier for the object for which authorizations are defined. If TCTXT is used, do not specify a value for RESOURCE.
flags	int	in	SECAD_ACT_CONTENTS When TCTXT or RESOURCE references an ACT, this flag specifies to apply the authorizations to the ACT's content, rather than to the authorizations that protect the ACT.

Parameter	Type	Direction	Description
authorizations	string array	in	Passed two-dimensional string array with five columns. Each row in the array represents a permission being set for an identity. See the “Details” section for more information. Column 0: Specify Person, IdentityGroup, or Role, indicating the identity’s type. Column 1: Specify the identity’s Name attribute value. Column 2: Specify a permission directive: D for Deny, G for Grant, or R for Remove. Column 3: Specify a Permission name. For example, Read, Write, and so on. Caution: If you specify R in column 2 and leave column 3 empty, then all permissions will be removed for the identity that is identified in columns 0 and 1. Column 4: Specify a permission condition for the identity and permission, or leave empty.

Details

The SetAuthorizationsOnObj method adds or removes permissions for an identity on a resource. The TCTXT or RESOURCE parameter and the AUTHORIZATIONS parameter are required. Other parameters can have a null value.

TCTXT or RESOURCE can specify an application metadata object or an ACT. When RESOURCE is an ACT, be aware that the SECAD_ACT_CONTENTS flag changes the behavior of the method. When this flag is set, the permission changes that you specified in AUTHORIZATIONS are applied to the contents that define the ACT. As a result, the changes affect all objects with which the ACT is associated. When this flag is not set, the permission changes are applied to the authorizations that protect the ACT object.

Use the AUTHORIZATIONS string to specify which identities are affected and the permissions that should be added or removed. The method uses this input to define or modify ACT and ACE objects on the SAS Metadata Server. Any permission conditions that you specify define or modify PermissionCondition objects.

The SetAuthorizationsOnObj method fails if the requesting user does not have ReadMetadata and WriteMetadata permissions on the target resource.

Exceptions Thrown

The SAS Open Metadata Interface explicitly returns the following exceptions for the SetAuthorizationsOnObj method:

- SECAD_INVALID_TC_HANDLE
- SECAD_INVALID_RESOURCE_SPEC
- SECAD_INVALID_ACTION
- SECAD_INVALID_IDENTITY_SPEC
- SECAD_IDENTITY_DOES_NOT_EXIST
- SECAD_INVALID_PERMISSION_SPEC
- SECAD_NOT_AUTHORIZED

Examples

The following code fragment shows how the SetAuthorizationsOnObj method is issued in a Java environment:

```
public void setAuthorizationsOnObj(String transCtxt, String resource, int options,
String[][] auths ) throws Exception {

        try
        {
            iSecurityAdmin.SetAuthorizationsOnObj(transCtxt,
                                                  resource,
                                                  options,
                                                  auths
                                                  );
        }
        catch (Exceptions e) {
            System.out.println("SetAuthorizationsOnObj:  Exceptions");
            e.printStackTrace();
            throw e;
        }
}
```

The following example issues the SetAuthorizationsOnObj to define authorizations in a predefined ACT identified as ACTspec.

```
public void defineACT() throws Exception {
        // Authorizations to place in the ACT
        final String[][] ACTauths =
                             {{"IdentityGroup", Public, "D", "ReadMetadata", ""},
                              {"IdentityGroup", Public, "D", "WriteMetadata", ""},
                              {"Person", testUserName, "G", "ReadMetadata", ""},
                              {"Person", testUserName, "G", "WriteMemberMetadata",""},
                              {"Person", testUserName, "G", "CheckinMetadata", ""}};

        try {
```

```
            // Set the authorizations defined in ACTauths on the ACT identified by ACTspec.
            // Note that tCtxt is null, because resource has a value.
 iSecurityAdmin. setAuthorizationsOnObj("", ACTspec, ISecurityAdmin.SECAD_ACT_CONTENTS,
ACTauths);
            }
           catch (Exception e ){
                  throw e;
           }
}
```

Related Methods

- “GetAuthorizationsOnObj Method” on page 220
- “CreateAccessControlTemplate Method” on page 206

10

Server Control (IServer Interface)

Overview of the IServer Server Interface

The methods described in this section are provided in the IServer server interface and can be used in a SAS Open Metadata Interface client that you create to perform server administrative tasks.

Except for the Status method, a user must have administrative user status on the SAS Metadata Server to issue all methods. For more information about administrative user status, see the *SAS Intelligence Platform: Security Administration Guide*.

Except for the Status method, IServer methods are available only in the standard interface. For more information, see "Communicating with the SAS Metadata Server" on page 14.

The following information applies to all of the IServer methods.

- The variable RC captures the return code of the method.
- In the examples, serverObject is an instantiation of the IServer interface.

Using the IServer Server Interface

Calling the Server Interface

The IServer server interface is called by connecting to the SAS Metadata Server and obtaining a handle to the IServer interface.

A SAS Java Metadata Interface client accesses the IServer interface by importing the appropriate packages, instantiating an object factory, and connecting to the SAS Metadata Server with a handle to the interface that is appropriate for the task that it wants to perform.

The IServer interface is provided in the sas.oma.omi.jar file in the SAS Platform VJR. A Java client accesses the IServer interface by importing the appropriate com.sas.meta.SASOMI packages. Import com.sas.meta.SASOMI.IServer and com.sas.metadata.remote into your client.

The SAS Java Metadata Interface provides the MdFactory interface to instantiate an object factory for the SAS Metadata Server and provides the MdOMRConnection interface for connecting to the SAS Metadata Server. Use the MdOMRConnection interface's makeIServerConnection method to connect to the server with the IServer interface.

Understanding the IServer Server Interface

The IServer interface includes the following server control methods:

Pause
: Temporarily limits the availability of the SAS Metadata Server. For more information, see “Pause()” on page 241.

Refresh
: Changes certain SAS Metadata Server invocation and configuration options on a running server. For more information, see “Refresh Method” on page 244.

Resume
: Returns a paused SAS Metadata Server to an ONLINE state. For more information, see “Resume Method” on page 253.

Status
: Polls the SAS Metadata Server for status, platform version, SAS Metadata Model version, server locale, server configuration information, and journaling statistics. For more information, see “Status Method” on page 255.

Stop
: Shuts down the SAS Metadata Server. For more information, see “Stop Method” on page 271.

Server Backup and Recovery Facility

The SAS Metadata Server includes a server-based facility that can be used to perform unassisted, scheduled SAS Metadata Server backups. These backups do not interrupt the regular operation of the SAS Metadata Server. The facility can perform scheduled and ad hoc backups, restores, and optional roll-forward recoveries of the SAS Metadata Server from the SAS Metadata Server journal.

The server backup facility always performs a full backup of the SAS Metadata Server. A full backup includes all configuration files in the `SASMeta/MetadataServer` directory and all SAS metadata repositories that are registered on the SAS Metadata Server, regardless of their registered access modes.

Regular server backups are configured during the SAS Metadata Server installation by the SAS Deployment Wizard. See the *SAS Intelligence Platform: System Administration Guide* for information about the default backup schedule.

The facility supports automated restores from a backup with optional roll-forward recovery to a specified point in time.

IServer methods implement the server-based facility and can be used to manage backup and recovery.

- Options in the Refresh method are used to set and modify the default backup configuration and backup schedule and to enable clients to execute ad hoc backups and restores.

- XML elements in the Status method return requested information about the backup configuration, backup schedule, backup history, and specific backups and restores.
- A <FORCE/> XML element in the Pause or Resume method enables administrators to regain control of the server in the event that it does not respond during a recovery.

However, the recommended interface for managing backups and performing restores is SAS Management Console. SAS Management Console provides a graphical user interface for managing backups. For information about the SAS Management Console interface, see "Backing Up and Recovering the SAS Metadata Server" on page 17.

For information about IServer backup and recovery options, see "Refresh Method" on page 244, "Status Method" on page 255, "Pause()" on page 241, and "Resume Method" on page 253.

Metadata Server Clustering

Beginning in SAS 9.4, SAS supports single SAS Metadata Server and clustered SAS Metadata Server configurations.

- In a single SAS Metadata Server configuration, all metadata requests are received and processed by one SAS Metadata Server. If this metadata server ceases to function, metadata processing is unavailable until the metadata server can be recovered.
- In a clustered SAS Metadata Server configuration, three or more identical SAS Metadata Servers are linked together to provide for metadata redundancy and make all of the metadata servers available for processing requests. If one metadata server becomes unavailable, its load is transferred to another metadata server.

A SAS Metadata Server cluster is configured by the SAS Deployment Wizard. Additional server nodes can be added by the SAS Deployment Wizard. Metadata servers are permanently removed from the cluster with SAS Deployment Manager. IServer methods cannot be used to define or redefine the cluster.

IServer methods can be used to control client activity on the cluster of metadata servers as a whole, and on an individual metadata server in the cluster.

- When the <CLUSTER/> XML element is specified as an option in the Pause, Refresh, Resume, or Stop method, the pause, refresh, resume, and stop actions are executed simultaneously on all metadata servers in the cluster.
- When the <SINGLE_SERVER/> XML element is specified as an option in the Pause, Refresh, Resume, or Stop method, the pause, refresh, resume, and stop actions are performed only on the metadata server indicated in the server connection options.

To ensure integrity in a clustered metadata server environment, it is recommended that clients specify a behavior and not leave the intended behavior unspecified. If <CLUSTER/> is encountered in single server configuration, the element is ignored.

It is recommended that the <CLUSTER/> element be specified for most Refresh actions. <CLUSTER/> instructs the Refresh method to act on each XML element in the clustered environment. I a clustered metadata server environment, the master

server is responsible for keeping metadata on the slave servers up to date. Backups are made on the master server. The master server automatically recovers slave servers that are removed and returned to the cluster. Administrative intervention is required only when a substantial change to the metadata on the cluster is needed (for example, to reorganize space in metadata repositories or to return the entire cluster to an earlier version of metadata). For these tasks, administrators must stop the cluster and make the changes on the master server in stand-alone mode. When the cluster is restarted, the master server propagates the changes to the slave servers. <CLUSTER/> ensures that Refresh requests that should be performed by the master server are routed accordingly, and that elements that request actions that should be performed by administrators to a master server in stand-alone mode are ignored in the clustered configuration.For more information, see “Refresh Method” on page 244.

The IServer Status method can be used to get information about the cluster. For more information about the cluster XML elements, see “Status Method” on page 255.

In the clustered metadata server configuration, a load-balancing algorithm controls server connections. The load-balancing code is not part of the SAS Open Metadata Interface, so the server cluster cannot be controlled with IServer methods. Instead of using IServer methods to control and get information about the server cluster, use the METAOPERATE procedure and the METADATA procedure. For information about these procedures, see the *SAS Language Interfaces to Metadata*.

Pause()

Short Description

Temporarily limits the availability of the SAS Metadata Server.

Category

IServer interface server control methods

Syntax

```
rc=Pause(options);
```

Table 10.1 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the SAS Metadata Server ran the method call. RC=0 means that it did, RC=1 means that it did not.
options	C	in	<CLUSTER/> Specifies to execute the Pause request on all SAS Metadata Servers in the cluster. In a single SAS Metadata Server configuration, this option is ignored. <FORCE/> Regains control of the SAS Metadata Server in the unlikely event that it does not respond during recovery processing. <PAUSECOMMENT> Enables administrators to set a free-form text comment that is retrieved with server status queries. <SERVER STATE="*value*"/> Specifies to change the availability of the SAS Metadata Server to the specified server state. Valid values are ADMIN, READONLY, or OFFLINE. If a STATE value is not specified or the <SERVER> option is omitted from the Pause request, the default value is <SERVER STATE="OFFLINE"/>. <SINGLE_SERVER/> Specifies to execute the Pause request on the connected SAS Metadata Server only.

Details

A user must have administrative user status on the SAS Metadata Server to issue the Pause method.

The Pause method is issued on a running SAS Metadata Server to temporarily change the server to a more restrictive state. The method downgrades the availability of all repositories on the server, unless a repository is registered with a more restrictive Access value than the one specified. You cannot change the state of a specific repository with Pause. To limit a metadata repository's availability, use the UpdateMetadata method to change the value in the repository's Access attribute.

When the <SINGLE_SERVER/> element is specified, the method affects the SAS Metadata Server on which it is executed. When the <CLUSTER/> element is

specified, the method affects all SAS Metadata Servers that are configured in the server cluster. A user is encouraged to specify one of these elements. If neither element is specified, the behavior is undefined. If <CLUSTER/> is specified but a server cluster is not configured, then the method is executed locally.

Note: Pausing and resuming an individual metadata server in a cluster can result in undesirable behavior. The preferred method for making an individual metadata server temporarily unavailable is to stop the server. For more information, see "Stop Method" on page 271.

Six states are supported for a running SAS Metadata Server:

ONLINE
: This is the normal state of a running SAS Metadata Server. It indicates the server is available for reading and writing to all users.

ADMIN
: The SAS Metadata Server is available for reading and writing, but only to users who have administrative user status on the SAS Metadata Server.

READONLY
: The SAS Metadata Server is available for reading only to all users.

ADMIN(READONLY)
: The SAS Metadata Server is available for reading only to users who have administrative user status on the SAS Metadata Server.

RECOVERY
: The SAS Metadata Server is unavailable for reading and writing to all users, except for the backup recovery process.

OFFLINE
: The SAS Metadata Server is running but is temporarily unavailable to all users.

The ADMIN, READONLY, and OFFLINE states are set with the Pause method.

The ADMIN(READONLY) and RECOVERY states are set by internal server recovery processes. In the unlikely event the metadata server fails to respond during the recovery process, administrators can use the Pause method with the <FORCE/> option to regain control of the server. When the server is in the RECOVERY state, Pause requests without the <FORCE/> option are rejected. By default, <FORCE/> changes the server to an OFFLINE state. The <FORCE/> option should be used with the <SERVER STATE="ADMIN"/> option to enable administrators to examine the server before making it available to users.

The ADMIN(READONLY) state is set by internal server recovery processes that fail if the server was in a READONLY state when the recovery process was started. From this state, administrators can troubleshoot the failure before making the server available to users.

When the server has been changed to another state by the Pause method or by internal server recovery processes, the Resume method must be used to return the server to an ONLINE state. For more information, see "Resume Method" on page 253.

The <PAUSECOMMENT> element enables administrators to set a free-form text comment with the Pause request. The comment is retrieved with server status queries. The <PAUSECOMMENT> element is specified as follows:

```
<PauseComment>This is a test of the Pause method. </PauseComment>
```

The <PAUSECOMMENT> element can be added to any exception returned by a method that rejects a user request because of a paused status. The Resume method clears the text in the <PAUSECOMMENT> element.

When a paused SAS Metadata Server is stopped and restarted, it restarts in an ONLINE state. Neither a server pause nor <PAUSECOMMENT> is persisted between server sessions.

The state of the SAS Metadata Server is obtained by issuing the Status method. For more information, see “Status Method” on page 255.

Examples

The following example pauses all SAS Metadata Servers in the cluster to an OFFLINE state:

```
<!-- Pause the current server to OFFLINE  -->
options='<Cluster/>';
rc=serverObject.Pause(options);
```

OFFLINE is the default value when the Pause method is issued without the <SERVER STATE="*value*"/> element.

The following example pauses all SAS Metadata Servers in the cluster to an ADMIN state and includes a <PAUSECOMMENT> to explain why the cluster is unavailable:

```
<!-- Pause all servers in the cluster to ADMIN  -->
options='<Server State="ADMIN"/><Cluster/><PauseComment>This is a test of the
Pause method.  Client activity on the SAS Metadata Server cluster will be resumed
shortly.</PauseComment>';
rc=serverObject.Pause(options);
```

If the <CLUSTER/> element is encountered in a single server configuration, the method pauses the connected SAS Metadata Server.

Refresh Method

Short Description

Changes certain SAS Metadata Server invocation and configuration options on a running server. This method can also be used to quickly recover memory on the SAS Metadata Server.

Category

IServer interface server control methods

Syntax

```
rc=Refresh(options);
```

Table 10.2 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the SAS Metadata Server ran the method call. RC=0 means that it did, RC=1 means that it did not.
options	C	in	<*alert-email-options*/> Includes XML elements that enable you to send test alert e-mails and temporarily change the alert e-mail recipients and configuration. For more information, see "Alert E-Mail Options" on page 248. <ARM *system-options*/> Invokes specified ARM logging system options. For more information, see "ARM Configuration Options" on page 249. <*Backup-options*/> Set or modify the server backup configuration and server backup schedule for the server-based backup and recovery facility, execute ad hoc backups, and maintain the scheduler thread. For more information, see "Backup Options" on page 249. <CLUSTER/> Specifies to execute the Refresh request on all SAS Metadata Servers in the cluster. In a single SAS Metadata Server configuration, this option is ignored.
			<OMA JOURNALPATH="*filename*"/> Specifies to stop writing journal entries to the journal file in the current location and resume writing journal entries in a new journal file in the specified location. This option is valid only when the metadata server is configured with <OMA JOURNALTYPE="SINGLE"/> in the omaconfig.xml file. If the metadata server is configured with any other journal type value, this option returns an error.

Parameter	Type	Direction	Description
			<RECOVER *options*/> Restores the SAS Metadata Server from a specified server backup and can perform a roll-forward recovery from the metadata server journal file. For more information, see "Recover Option" on page 251. <REPOSITORY Id="*repository-identifier*"/> Specifies to close and reopen the specified metadata repository to recover memory. <RPOSMGR ACCESS="ONLINE"/> Specifies to resume the SAS Repository Manager to an ONLINE state. When the SAS Repository Manager is resumed, metadata repositories are returned to their registered Access values. <SERVER/> Specifies to close and reopen all of the metadata repositories that are managed by the SAS Metadata Server to recover memory. <SINGLE_SERVER/> Specifies to execute the Refresh request on the connected SAS Metadata Server only.

Details

General Operation

A user must have administrative user status on the SAS Metadata Server to issue the Refresh method. The Refresh method accepts instruction from XML elements that are specified in the OPTIONS= parameter.

At least one OPTIONS= XML element must be specified for the Refresh method to have an effect. The method supports a memory recovery option, an alert e-mail testing option, and four categories of optional configuration elements: alert e-mail system options, ARM system options, a journal option, and server backup and recovery options. None of the configuration elements affect memory size.

When the <SINGLE_SERVER/> element is specified with an XML element, the action is performed on the SAS Metadata Server on which it is executed. When the <CLUSTER/> element is specified with an XML element, the action is performed on all SAS Metadata Servers that are configured in the server cluster.

A user is encouraged to specify either <CLUSTER/> or <SINGLE_SERVER/> with the other XML elements. If neither element is specified, the behavior is undefined. If

<CLUSTER/> is specified but a server cluster is not configured, then the method is executed locally.

The memory recovery options are the only XML elements with which a user might want to specify the <SINGLE_SERVER/> element. For all other XML elements, the <CLUSTER/> element is appropriate. The <CLUSTER/> element instructs the Refresh method to take the appropriate action for the element in the server cluster. Specifically, when <CLUSTER/> is specified and the server is running in a clustered environment:

- ARM logging is invoked on the master SAS Metadata Server.
- The server backup facility backs up the master SAS Metadata Server.
- Backup configuration and scheduling information is stored centrally, where all SAS Metadata Servers can access it.
- Alert e-mail settings are temporarily changed for all SAS Metadata Servers in the cluster.
- Options that replace or restructure repository data sets, such as the <RECOVER *options*/> option and the backup REORG option, are not honored unless the SAS Metadata Server on which they are executed is running in stand-alone mode. A server is running in stand-alone mode when it is started with the STARTNOCLUSTER option. For more information about restore and recovery in a server cluster, see *SAS Language Interfaces to Metadata*.
- The <OMA JOURNALPATH="*filename*"/> option is ignored.

The SAS Metadata Server must refresh repositories when there is no other activity on the SAS Metadata Server, so it automatically delays other user requests until the Refresh method completes. This might have a small effect on server performance.

Memory Recovery Options

The memory recovery XML elements are <SERVER/> and <REPOSITORY/>.

When executed with the <SERVER/> element, the Refresh method pauses the SAS Metadata Server to an OFFLINE state, and then resumes it to an ONLINE state, in one step. This process unloads all metadata repositories that are managed by the SAS Metadata Server and the SAS Repository Manager from memory and closes all repository containers on disk. Memory on the SAS Metadata Server host is quickly recovered. The repositories remain in their current Access states. Closed containers are reopened as needed in response to subsequent metadata queries and updates.

The <REPOSITORY Id="*repository-identifier*"/> element unloads the specified metadata repository from memory and closes its container on disk. Memory used by the repository is recovered. The specified repository remains in its current Access state. The repository's container is reopened as needed in response to subsequent metadata queries and updates.

The <REPOSITORY/> element has the syntax:

```
<REPOSITORY Id="Reposid|REPOSMGR|ALL"/>
```

REPOSID
: Specifies the unique 8-character or two-part 17-character metadata identifier of a metadata repository. Multiple repositories can be refreshed at once by stacking <REPOSITORY/> elements in the OPTIONS parameter.

Note: The foundation repository should not be refreshed without also refreshing all metadata repositories that depend on it.

REPOSMGR
Specifies the SAS Repository Manager.

ALL
Includes all metadata repositories, including the SAS Repository Manager. ALL is the default value if <REPOSITORY/> is specified without the Id= attribute and has the same effect as specifying the <SERVER/> element.

Specifying <CLUSTER/> with the <SERVER/> or <REPOSITORY/> element causes the Refresh method to close and reopen the specified repositories in all metadata servers in the cluster. Specifying <SINGLE_SERVER/> with the <SERVER/> or <REPOSITORY/> element causes the specified metadata repositories to close and reopen just in the server that receives the request.

Alert E-Mail Options

The e-mail testing XML element is:

<OMA ALERTEMAILTEST= "*text*"/>
Sends a test e-mail message to the addresses configured in the SAS Metadata Server's omaconfig.xml file, specifically in the <OMA ALERTEMAIL="*email-address*"/> option. The e-mail recipients are displayed on the **General** tab of the active server's Properties window in SAS Management Console. Or, the e-mail recipients can be listed with the Status method.

The e-mail configuration options are:

<OMA ALERTEMAIL="*email-address*"/>
Temporarily modifies the e-mail addresses to which the SAS Metadata Server will send an alert e-mail message. To specify more than one e-mail address, enclose each address in single quotation marks, place a blank space between each address, and enclose the list in parentheses. For example: "('Bill@mycompany.com' 'Susan@mycompany.com')"

The e-mail system options are:

<OMA EMAILAUTHPROTOCOL="NONE | LOGIN"/>
Changes the authentication protocol for SMTP e-mail that is sent by the SAS Metadata Server. When you specify the value "LOGIN", you also need to specify EMAILID and EMAILPW. The value "NONE" specifies that no authentication protocol is used.

<OMA EMAILHOST="*server-network-address*"/>
Changes the network address of the enterprise's SMTP server (for example, mailhost.company.com).

<OMA EMAILID="*server-email-address*"/>
Changes the e-mail address for the From field of alert e-mail messages that are sent by the SAS Metadata Server. The e-mail address can be entered in either of the following forms:

- "*server-name*<*user-account@domain*>"
- "<*user-account@domain*>"

<OMA EMAILPW="*password*"/>
Specifies the logon password to be used with the e-mail address that you specified for the EMAILID option. The password should be encoded with PROC

PWENCODE. We recommend that you use SAS02 as the minimum encryption level. SAS02 is the standard SAS proprietary encryption level that is available with PROC PWENCODE. To specify a higher encryption level, you must have SAS/SECURE software. For more information about PROC PWENCODE, see the *Base SAS Procedures Guide*.

<OMA EMAILPORT="*port-number*"/>
Changes the port number that is used by the SMTP server that you specified for the EMAILHOST option.

If a test e-mail is not received, you can use <OMA ALERTEMAIL="*email-address*"/> to change the alert e-mail recipients, or you can use the <OMA *e-mail-system-options*/> to change the e-mail server's configuration while the metadata server is running and test again until the test succeeds.

The Refresh method changes the alert e-mail settings for the duration of the server session. To permanently change alert e-mail recipients, you must modify the omaconfig.xml file. To permanently change e-mail system options for the alert e-mail server, you must modify the sasv9.cfg file. You must stop the metadata server to modify these files.

To get the current values of the e-mail system options, use the Status method. For more information, see "Status Method" on page 255. For more information about the metadata server's alert e-mail notification system, see the *SAS Intelligence Platform: System Administration Guide*.

ARM Configuration Options

<ARM *system-options*/>
Specifies one or more ARM system options as follows:

```
<ARM ARMSUBSYS="(ARM_NONE|ARM_OMA)" ARMLOC="fileref|filename"/>
```

The ARMSUBSYS system option enables or disables ARM logging on the server. Specify "(ARM_OMA)" to enable ARM logging. Specify "(ARM_NONE)" to disable ARM logging. ARM logging is disabled by default.

The ARMLOC system option specifies a location for the log. ARMLOC should be specified with ARMSUBSYS when you enable ARM logging. If ARM logging is already enabled, specifying just ARMLOC="*filename*" writes the ARM log to a new location. Absolute and relative pathnames are read as different locations. For more information about ARM logging, see "Using the ARM_OMA Subsystem" in the *SAS Intelligence Platform: System Administration Guide*.

Backup Options

The server backup XML elements include:

<BACKUP *optional-attributes*/>
Invokes an ad hoc server backup to the backup location indicated in the MetadataServerBackupConfiguration.xml file. The backup is named with a datetime stamp in a modified ISO-8601 format (server-local datetime value followed by the GMT offset). The <BACKUP/> element supports two optional attributes:

COMMENT= "*text*"
Specifies a text string of an unlimited length to describe the reason for the ad hoc backup. The text string is recorded in the backup history.

REORG="Y|N"
Specifies whether repository data sets should be rebuilt to remove unused disk space. The default value is "N" (No). Use of this option in ad hoc backups is discouraged because the REORG process pauses and resumes repositories, which interrupts the regular operation of the metadata server.

Note: This option is ignored in a running clustered server configuration. To execute REORG in a clustered server configuration, you must stop the cluster and run the process on a server that is running in stand-alone mode.

<BACKUPCONFIGURATION *parameters*/>
Sets or modifies the value of a specified server backup parameter in the MetadataServerBackupConfiguration.xml file. The parameters are:

BACKUPLOCATION="*directory*"
Specifies backups will be stored in a directory of the specified name in the SASMeta/MetadataServer directory of your SAS configuration. The default backup directory is named Backups. If the specified directory does not exist, the metadata server will create it for you. To create the directory in a location other than SASMeta/MetadataServer or on a different drive, specify an absolute pathname that is meaningful to the computer that hosts the SAS Metadata Server. The server will create one directory for backups. If a pathname is specified, the other directory levels must already exist.

RUNSCHEDULEDBACKUPS="Y | N"
Controls the scheduler thread. A value of "Y" activates scheduled backups; a value of "N" turns them off.

DAYSTORETAINBACKUPS="*number*"
Specifies the number of days to keep backups before they are deleted from the SAS Metadata Server host. The default value is "7". To never delete any backups, specify "0" in this attribute. Specifying a value of 0 is not recommended because it can cause disk space issues.

<SCHEDULE EVENT= "Backup" WEEKDAY*n*= "*time-value*R"/>
Sets or modifies the server backup schedule. The server backup schedule is stored in the MetadataServerBackupConfiguration.xml file.

EVENT= "Backup"
Specifies the event to schedule. "Backup" is the valid value.

WEEKDAY*n*= "*time-value*"
The server backup facility supports daily backups, specified on a weekly schedule, where the attribute Weekday1="*time-value*" is Sunday and the attribute Weekday7="*time-value*" is the following Saturday. Appropriately numbered Weekday*n*="*time-value*" attributes represent the other days of the week. Backup times are specified using four-digit time values indicating hours and minutes based on a 24-hour clock in server-local time; for example, 0100 is 1 a.m and 1300 is 1 p.m. To schedule a backup, specify the appropriate Weekday*n*="*time-value*" attribute with a time value. To schedule more than one backup in a day, specify multiple time values and separate them with semicolons.

R
Is optional and requests a REORG operation. The REORG option rebuilds repositories to remove unused disk space. It is a resource-intensive operation that pauses the server, so it should be used sparingly. It is recommended that a REORG be performed no more than once a week.

Note: The REORG option is ignored in a running clustered server configuration. To execute a REORG in a clustered server configuration, you must stop the cluster, and then run the process on a server that is running in stand-alone mode.

FOR INTERNAL USE ONLY: The EVENT="" attribute also supports the values Test, Bogus, and Crash. These are supported for metadata server testing. The Test event enables testers to schedule events on the server that do not actually cause backups to see whether the scheduler works. The Bogus event is recognized by the Refresh method, but not by the scheduler to verify that the scheduler reacts appropriately to unrecognized event names. The Crash event causes the thread to crash. The Crash option works only if the omaconfig.xml option OMA.ENABLE_CRASH_TESTING is set.

<SCHEDULER/>
When server backups are configured, a scheduler thread runs continuously from the time the SAS Metadata Server is started. It buffers the schedule in 48-hour increments. Specifying the <SCHEDULER/> XML element with a <REBUILD/> subelement forces the scheduler thread to rebuild its in-memory linked list of events. Specifying <SCHEDULER/> with a <RESTART/> subelement causes the current scheduler thread to stop and start a new thread.

Recover Option

<RECOVER *required-and-optional-parameters*/>
Restores SAS metadata repositories using the backup specified in a BACKUPNAME="*name*" or in a BACKUPPATH="*pathname*" attribute with the options indicated in optional attributes. The specified backup is recovered to the current metadata server directory. If the server path was different when the backup was made, the repository pathnames in the backup are changed to relative pathnames to enable the server to function in the new location.

BACKUPNAME or BACKUPATH is a required parameter.

BACKUPNAME="*name*"
Specifies the name of a backup to use to recover the server. The backup is assumed to be located in the backup location indicated in the MetadataServerBackupConfiguration.xml file. To use a backup from a different location to recover the server, either use the BACKUPPATH attribute instead of BACKUPNAME or specify the BACKUPLOCATION attribute with the BACKUPNAME attribute.

BACKUPLOCATION="*directory*"
Specifies the name of the directory that contains the backup specified in BACKUPNAME.

BACKUPPATH="*pathname*"
Specifies the full or relative pathname to a backup. Relative pathnames are mapped to the SASMeta/MetadataServer directory.

Optional parameters include:

COMMENT="*text*"
Specifies a character string of unlimited length that contains an explanation for the recovery. The text is recorded in the backup history.

PAUSECOMMENT="*text*"
Specifies a character string of unlimited length that is displayed as a recovery notification to clients.

IGNOREVERIFY= "Y | N"
Specifies whether to perform a validation process to determine whether the content of the backup matches the record of the backup in the METADATASERVERBACKUPMANIFEST.xml file. The recovery is stopped if it does not match. The value "N" instructs the server to perform the validation. This is the recommended setting. The value "Y" instructs the metadata server to perform the recovery and skip the validation.

CAUTION
You should evaluate your backup very carefully before you set IGNOREVERIFY="Y". If the backup is missing a critical component like the repository manager (A0000001) or a repository, your server might not start after recovery.

INCLUDEALLCONFIGFILES= "Y | N"
Specifies whether to replace all configuration files in the server directory with the configuration files present in the directory when the backup was made.

CAUTION
This option should not be set in a clustered environment. Configuration files contains host-specific paths. In a clustered server configuration, a backup might be restored on a different host than the one on which it was made.

CAUTION
When INCLUDEALLCONFIGFILES="Y", all files in the server directory will be deleted and replaced with the configuration files in the backup. Any files added to the directory since the backup will be lost.

ROLL_FORWARD= "blank | _ALL_ | *datetime*"
Specifies whether the metadata server journal should be used to recover metadata server transactions that were recorded beyond those recovered in the restored image. It also specifies whether to recover all transactions or only transactions up to a specified point from the journal. Omission of this parameter or specifying the parameter with a blank value specifies not to recover transactions from the journal.

ALL
Specifies to recover all transactions in the journal.

Datetime
Specifies to recover transactions up to the specified date and time. The ROLL_FORWARD argument requires input in GMT time.

Examples

The following example shows how ARM_OMA logging is enabled on the SAS Metadata Server:

```
options='<ARM armbsubsys="(ARM_OMA)" armloc="myARM.log"/><CLUSTER/>';
rc=serverObject.Refresh(options);
```

The following example shows how ARM_OMA logging is disabled on the SAS Metadata Server:

```
options='<ARM armbsubsys="(ARM_NONE)"/><CLUSTER/>';
rc=serverObject.Refresh(options);
```

The following example shows how to close and reopen all metadata repositories on one SAS Metadata Server to recover memory.

```
options='<SERVER/><SINGLE_SERVER/>';
rc=serverObject.Refresh(options);
```

The following example shows how to close and reopen all metadata repositories on all SAS Metadata Servers in the cluster to recover memory:

```
options='<SERVER/><CLUSTER/>';
rc=serverObject.Refresh(options);
```

The following example shows how to request an ad hoc backup in a clustered server environment:

```
options='<Backup Comment="Ad hoc backup started by the backup.sas job."/><CLUSTER/>';
rc=serverObject.Refresh(options);
```

The following example shows how to modify the server backup schedule on the cluster:

```
options='<Schedule Event="Backup" Weekday1="0010;1700"
Weekday2="0010R;1700" Weekday3="0010;1500" Weekday4="0010;1500" Weekday5="0010;1700"
Weekday6="0010;1700" Weekday7="0010"/><CLUSTER/>'
rc=serverObject.Refresh(options);
```

Resume Method

Short Description

Returns a paused SAS Metadata Server to an ONLINE state.

Category

IServer interface server control methods

Syntax

```
rc=Resume(options);
```

Table 10.3 Method Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the SAS Metadata Server ran the method call. RC=0 means that it did, RC=1 means that it did not.
options	C	in	<CLUSTER/> Specifies to execute the Resume request on all SAS Metadata Servers in the cluster. In a single SAS Metadata Server configuration, this option is ignored. <FORCE/> Regains control of the SAS Metadata Server in the unlikely event that it does not respond during recovery processing. <SERVER/> must be specified with <FORCE/>. <SERVER/> Specifies to return the SAS Metadata Server to an ONLINE state. <SINGLE_SERVER/> Specifies to execute the Resume request on the connected SAS Metadata Server only.

Details

A user must have administrative user status on the SAS Metadata Server to issue the Resume method.

The Resume method returns a SAS Metadata Server that has been paused to the ONLINE state. The ONLINE state makes the server available to process user requests. Individual metadata repositories are returned to the persisted access modes indicated in their Access attributes.

When the <SINGLE_SERVER/> element is specified, the method affects the SAS Metadata Server on which it is executed. When the <CLUSTER/> element is specified, the method affects all SAS Metadata Servers that are configured in the server cluster.

A user is encouraged to specify one of these elements. If neither element is specified, the behavior is undefined. If <CLUSTER/> is specified but a server cluster is not configured, then the method is executed locally.

Note: Pausing and resuming an individual metadata server in a cluster can result in undesirable behavior. The preferred method for making an individual metadata server temporarily unavailable is to stop the server. For more information, see “Stop Method” on page 271.

The Resume method cannot be used to change the availability of a metadata repository. To return a specific metadata repository to an ONLINE state, use the UpdateMetadata method to change the value in the repository's Access attribute.

If the Resume method is issued without options, the method operates as if the <SERVER/> element was specified. However, if you specify the <FORCE/> option, the <SERVER/> element must be specified as well.

The <FORCE/> option regains control of the SAS Metadata Server in the unlikely event that it fails to respond during the recovery process. A server unresponsive if it remains it does not change from the RECOVERY state. For more information about the RECOVERY state, see "Pause()" on page 241.

Example

The following example shows how to execute the Resume method on the current SAS Metadata Server only:

```
<!-- Resume one server to an ONLINE state -->
options='<SINGLE_SERVER/>';
rc=serverObject.Resume(options);
```

The following example shows how to execute the Resume method on all SAS Metadata Servers in a cluster:

```
<!-- Resume all servers in the cluster to an ONLINE state -->
options='<CLUSTER/>';
rc=serverObject.Resume(options);
```

If the <CLUSTER/> element is encountered in a single server configuration, the method resumes the current SAS Metadata Server.

The following example shows how to specify the <FORCE/> option on all SAS Metadata Servers in a cluster:

```
<!-- Resume all servers in the cluster with the <FORCE/> option -->
options='<SERVER/><FORCE/><CLUSTER/>';
rc=serverObject.Resume(options);
```

Status Method

Short Description

Polls the SAS Metadata Server for server availability and configuration information, cluster information, backup information, and journaling and server usage statistics.

Category

IServer interface server control methods

Syntax

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

Table 10.4 *Method Parameters*

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. Indicates whether the SAS Metadata Server ran the method call. RC=0 means that it did, RC=1 means that it did not.
inmeta	string	in	Specifies one or more XML elements that requests information from the SAS Metadata Server. If no elements are specified or the <STATUS/> element is specified, the Status method returns the information described in "Default Status Elements" on page 258. The XML elements include: <*cluster-information-options*> Return information about the availability of the SAS Metadata Server cluster and the clustered metadata server configuration. These elements have no effect in a single SAS Metadata Server configuration. For more information, see "Cluster Information Elements" on page 259.

Parameter	Type	Direction	Description
			<OMA *alert-email-options*/> Return the current value of the specified alert e-mail option. The SAS Metadata Server uses alert e-mail omaconfig.xml options and e-mail system options to send alert e-mails. For more information, see "Alert E-Mail Options" on page 262. <OMA *journal-statistic(s)*/> Return the specified statistic about the SAS Metadata Server journal file. See "Journal Statistics Elements" on page 263 for information about the available statistics. <OMA *server-process-statistic(s)*/> Return the value of the specified server process statistic. For more information, see "Server Process Statistics Elements" on page 265. <*omaconfig-options*/> Return the current value of the specified <OMA> or <RPOSMGR> configuration option whether it was set in the omaconfig.xml configuration file at start-up or whether it was modified after start-up with the Refresh method. For more information, see "Requesting omaconfig.xml Values" on page 266.
			<Scheduler/> Supports an attribute and an optional XML element to check the status of the thread scheduler and alert conditions. For more information, see "<Scheduler/> XML Element" on page 269. <*server-backup-and-recovery-options*/> Return information about the metadata server backup facility. XML elements are available to get information about the current backup or recovery process, the last backup, the last recovery, the backup configuration, the backup schedule, the backup history, and the status of the backup scheduler thread. For more information, see "Server Backup and Recovery Options" on page 267. <STATUS/> Is the default option if the method is issued without options. For more information, see "Default Status Elements" on page 258.
outmeta	string	out	Mirrors the content in the INMETA parameter with the exception that return values are filled in.

Parameter	Type	Direction	Description
options	C	in	<CLUSTER/> in a clustered metadata server configuration, sends the query in the INMETA parameter to the master server. In a single server configuration, this parameter is ignored. Use <CLUSTER/> when you want to list information about all nodes in the cluster. **Note:** This option is available beginning with the third maintenance release for SAS 9.4.

Details

Default Status Elements

The default Status elements are the XML elements that are returned when you issue the Status method without specifying any XML elements in the INMETA parameter or when you specify <STATUS/> in the INMETA parameter. The returned XML elements can be submitted to the server independently to return specific information as well. The default XML elements include:

<MODELVERSION/>
Returns the SAS Metadata Model version number in the form X.XX. For example, 12.04. The model version is incremented when there is a change to the SAS Metadata Model or to the repository format used by metadata repositories. The integer part of the version number is the repository format number. When this number is incremented, it indicates that the underlying data structure has changed and a conversion of the repository tables is highly recommended. It is possible for a SAS Metadata Server that was written for one repository format to use repositories that were created with an earlier repository format. 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 SAS Metadata Model change was made, but there is no need to convert the repository tables. A model change includes the addition or modification of metadata types, attributes, or associations.

<PAUSECOMMENT/>
When the SAS Metadata Server is paused to an ADMIN, READONLY, or OFFLINE state, this option returns a user-defined text comment set by the administrator describing why the server is unavailable. If the SAS Metadata Server is in an ONLINE state, it returns an empty string.

<PLATFORMVERSION/>
Returns the SAS Metadata Server version number in the form X.X.X.X. For example, for a SAS Metadata Server that is running SAS 9.4, the platform version number is 9.4.0.0.

<SERVERLOCALE/>
Returns the active server LOCALE value. For example, for a SAS Metadata Server that is running in the United States, the server locale value is en_US.

<SERVERSTATE/>
Returns the SAS Metadata Server's current state. Valid values are ONLINE, ADMIN, ADMIN(READONLY), READONLY, RECOVERY, or OFFLINE. No response means that the server is down.

<STATUS/>
This is the default option if the method is issued without options. This option returns values for <MODELVERSION/>, <PLATFORMVERSION/>, <SERVERSTATE/>, <PAUSECOMMENT/>, and <SERVERLOCALE/> in one request.

Note: The Status method reports the availability of the SAS Metadata Server to service client requests. It cannot be used to check the state of a specific metadata repository. If you need to know why a specific metadata repository is not available, check the repository's Access and State values in the SAS Management Console Metadata Manager or issue a GetRepositories method that sets the OMI_ALL (1) flag. For more information, see "GetRepositories Method" on page 122.

Cluster Information Elements

Beginning in SAS 9.4, SAS supports a clustered SAS Metadata Server configuration as an alternative to a single SAS Metadata Server configuration. In a clustered SAS Metadata Server configuration, three or more identical SAS Metadata Servers are linked together to provide for metadata redundancy and make all of the metadata servers available for processing requests. The servers have a master-slave relationship. In this relationship, the slave servers receive all client requests. The slave servers forward metadata write requests to the master server and share responsibility for processing read requests. The master server processes write requests and propagates updates to the slaves. If one server fails, its workload is automatically transferred to another server, unless a quorum is lost. A quorum requires that at least half of the servers in the cluster be available to continue to operate. If exactly half are available, the first server defined in the cluster by the SAS Deployment Wizard must be operating.

The Status method supports the following XML elements to return information about the cluster and the cluster's status in relationship to the quorum:

<CLUSTER *attributes*/>
Returns values for specified cluster attributes. The supported attributes are:

CLUSTERGUID=" "
Returns the cluster's unique identifier. This identifier is assigned and recorded in the server definition of every server in the cluster by the SAS Deployment Wizard at installation.

DEFINED_NODES=" "
Is the number of servers that are defined in the cluster.

CURRENT_NODES=" "
Is the number of servers that are known to the server that received the query.

HAS_FIRST_NODE=" "
Returns a YES or NO, indicating whether the server defined as Node 1 is available to the cluster.

HAS_QUORUM=" "
Returns a YES or NO, indicating whether a quorum exists.

LIST=" "
Returns an integer indicating the number of servers that are known to the server that received the query. In addition, it returns <ClusterNode/> XML elements that describe each server. The <ClusterNode/> XML elements include the server name, host name, port number, a Self attribute, and a Flags attribute for each server. The Self attribute identifies the receiving server with a “Y” or “N”. The Flags attribute indicates whether the node is a slave server or the master server.

By default, Status requests are processed by a slave server. To get meaningful information for the CURRENT_NODES= and LIST= attributes, you must direct <CLUSTER *attribute*/> requests to the master server. Beginning in the third maintenance release of SAS 9.4, this can be done by specifying a <CLUSTER/> XML element in the OPTIONS parameter of the Status method call. In previous versions of SAS 9.4, you had to use the METADATA Procedure to direct requests to a specific server. When METHOD=STATUS is used, PROC METADATA supports a NOREDIRECT argument that executes the status request on the connected server. For more information, see METADATA Procedure in the *SAS Language Interfaces to Metadata*.

<CLUSTERSTATE/>
Returns information about the cluster’s availability. Valid return values are STARTING, QUORUM, or LOSTQUORUM. STARTING means that the cluster is waiting for more server nodes to come up and complete the quorum. QUORUM means that a sufficient number of servers are operating for the cluster to remain in service. LOSTQUORUM means that the cluster does not have enough server nodes to remain in service.

<OMA MAXIMUM_CLUSTER_NODES=" "/>
Returns the maximum number of servers that are supported in the cluster as configured in the omaconfig.xml file. Fewer servers are initially defined than are supported. A customer can later add servers up to this number plus one (the additional slot is for the master server). If you have a need to exceed this number, contact SAS Technical Support.

Beginning in the third maintenance release for SAS 9.4, the SAS Metadata Server cluster has a synchronization check feature. This feature is available through the SAS Management Console Metadata Manager Analyze/Repair wizard and the sas-analyze-metadata batch tool. When invoked, this feature compares salient statistics in the metadata type containers of the specified repository on the master server to corresponding statistics for the same repository on each of the slave nodes. When the results do not match, an alert e-mail is sent to the administrator of the node that had the inconsistency, and an alert condition is raised for that repository. The best action for the administrator is to make sure the problem node is not critical to the quorum, and then stop and restart the problem node. The Status method supports the following XML element to return information about a synchronization check. The results of a synchronization check are available until the queried node is restarted.

<SynchCheck><Results *OptionalAttribute(s)=" "*/></SynchCheck>
reports the results of the last synchronization check on the slave node that received the Status query. By default, the query returns the Id value and Name value of all repositories that were examined and a <Container/> XML element that specifies the name of any metadata type container in which an error was found, as follows:

```
<SynchCheck>
   <Results>
     <Repository Id="A0000001" Name="REPOSMGR"/>
```

```
     <Repository Id="A5DST1OY" Name="Foundation">
       <Container Name="testleg" Status="ERROR" RecordCountDifference="-1"
HighIdDifference="Y"/>
     </Repository>
   </Results>
</SynchCheck>
```

The results indicate that the container named TestLeg has one record less on the queried node than on the master server.

The optional attributes are the following:

Detail="FULL"
specifies to return comprehensive information about the synchronization check. The Status query returns <Container/> XML elements similar to the following for every metadata type container in every repository that was examined.

This XML element shows the information that is returned for a container that has a problem:

```
<Container Name="testleg" Status="ERROR" RecordCountDifference="-1"
HighIdDifference="Y" Sample="01Jan1960:00:00:00,06Jan2015:20:24:26"
Master="2,0,AS000001,AS000002,19Dec2014:23:02:40,19Dec2014:23:02:40,3469298726.99,"
Slave="1,0,AS000001,AS000001,19Dec2014:23:02:40,19Dec2014:23:02:40,1734649360.2,"/>
```

This XML element shows the information that is returned for a container that passed the synchronization check:

```
<Container Name="testms" Status="OK" Sample="01Jan1960:00:00:00,06Jan2015:20:24:26"
Master="2,0,AR000001,AR000002,19Dec2014:23:02:40,19Dec2014:23:02:40,3469298726.95,"
Slave="2,0,AR000001,AR000002,19Dec2014:23:02:40,19Dec2014:23:02:40,3469298726.95,"/>
```

The information includes the following:

- The name of the container that was examined in the Name= attribute.
- A value representing the result of the synchronization check for the container in the Status= attribute, which is either OK or ERROR.
- For problem containers only, a positive or negative number identifying the number of records that contain a discrepancy in the RecordCountDifference= attribute. In this example, the container of the queried node has one record less in the TestLeg container than in the master server.
- For problem containers only, a Boolean value indicating whether the discrepancy was detected with the lowest object ID or the highest object ID. A “Y” in the HighIdDifference= attribute indicates that the containers started out with the same object ID, but that the slave node ended with a different object ID from the master server.
- Timestamps representing the beginning and ending of the test period in the Sample attribute. This test examined records that were written between January 1, 1960 and January 6, 2015. January 1, 1960 is the default start value for tests. It ensures that all records processed by the SAS Metadata Server are examined. January 6, 2015 is the date on which this particular synchronization check was performed. The sample period is controlled in the Analyze-Repair wizard.
- The following information for the master server and the slave node in the Master= and Slave= attributes, respectively:
  - the number of records in the metadata type container

- the number of records whose object ID is invalid because it has the wrong repository ID
- the lowest object ID
- the highest object ID
- the lowest modification datetime
- the highest modification datetime
- a simple sum of all of the modification datetime values

ReposName="*repository-name*"
specifies to return information for a named repository. When this attribute is omitted, the Status method returns information for all repositories for which a synchronization check was performed. The Analyze/Repair wizard and sas-analyze-metadata batch tool examine one repository at a time. The information available depends on the number of repositories that were checked before the queried node was restarted.

Note: The <SynchCheck/> element returns information for only the server node that received the request. To get a complete picture of synchronization activity on the cluster, you need to query all nodes. It is a good idea to submit a <SynchCheck/> request to all nodes in the cluster.

Note: In a clustered metadata server configuration, a load-balancing algorithm controls server connections. Therefore, the connected server is rarely the server that processes the requests. The load-balancing algorithm is not part of the SAS Open Metadata Interface, so the server node cannot be controlled with IServer methods. Instead of using the IServer Status method directly to get synchronization information, use the METADATA procedure. In PROC METADATA, specify METHOD=STATUS and specify the NOREDIRECT argument to ensure connection to a specific node. The NOREDIRECT argument temporarily overrides the load balancer and executes the request on the connected server. For more information, see the documentation for the METADATA procedure.

Alert E-Mail Options

The SAS Metadata Server uses an SMTP server to send alert e-mail messages. You can list the e-mail addresses that are configured to receive alert e-mail messages by querying the <OMA ALERTEMAIL=" "/> omaconfig.xml option.

<OMA ALERTEMAIL=" "/>
Returns the e-mail addresses to which the SAS Metadata Server will send an e-mail message in the event of a metadata server backup error, a metadata server recovery error, or an error that prevents the repository data sets from being updated from the journal.

The SAS Metadata Server's e-mail access is configured at server invocation with e-mail system options. To get information about the configuration, query these options:

<OMA EMAILAUTHPROTOCOL=" "/>
Returns the authentication protocol for SMTP that is sent by the SAS Metadata Server. Valid values are LOGIN or NONE. The value "LOGIN" indicates that a user ID and password are used. The value "NONE" indicates that no authentication protocol is used.

<OMA EMAILHOST=" "/>
Returns the network address of the enterprise's SMTP server (for example, mailhost.company.com).

<OMA EMAILID=" "/>
Returns the active e-mail address for the From field of alert e-mail messages that are sent by the SAS Metadata Server.

<OMA EMAILPW=" "/>
Returns the logon password that is used with the e-mail address configured in the EMAILID attribute (in encrypted form).

<OMA EMAILPORT=" "/>
Returns the port number that is used by the SMTP server that is configured in the EMAILHOST attribute.

You can query settings for omaconfig.xml options that configure alert e-mail reminders. Beginning in the second maintenance release for SAS 9.4, the SAS Metadata Server sends alert e-mail reminders in addition to the initial alert e-mail message for the alert condition `the journal commit task stopped running`. A SAS Metadata Server defines a grace period for addressing the alert condition before it terminates itself. The condition and reminders are tracked in a scheduler thread.

<OMA ALERT_CONDITION_FREQUENCY=" "/>
Returns the amount of time that will elapse before the initial and subsequent alert e-mail reminders about the alert condition will be sent. The time value is returned in seconds. The default value is 21,600 seconds (six hours).

<OMA ALERT_CONDITION_GRACE_PERIOD=" "/>
Returns the amount of time that the alert condition will be allowed to persist before the SAS Metadata Server shuts itself down. The time value is returned in seconds. The default value is 259,200 seconds (three days).

The e-mail recipients are defined in the omaconfig.xml file and can be temporarily modified for the duration of a server session by using the Refresh method. The e-mail system options can also be temporarily modified for the duration of a server session by using the Refresh method. The Status method returns the current option values, whether they were set in the omaconfig.xml file or the sasv9.cfg file or modified by using the Refresh method.

When an alert e-mail message for the alert condition `the journal commit task stopped running` is received, submit the <Scheduler/> option on the problem SAS Metadata Server. For more information, see "<Scheduler/> XML Element" on page 269.

Journal Statistics Elements

Journaling is a SAS Metadata Server performance feature that makes updates available in memory before they are written to SAS metadata repositories. The updates are stored in a high-speed journal file on disk until they can be transferred to the metadata repositories. The master server in a clustered server configuration uses the journal file to propagate metadata updates to the slave servers. The server-based backup facility uses the transactions stored in the journal file to perform roll-forward recovery.

The Status journal statistics elements enable you to monitor the size, content, and location of the journal file.

To conserve disk space, the SAS Metadata Server stores journal entries in a compressed format. XML elements that return journal size in bytes report statistics

for the compressed journal entries. To get decompressed values, you must include “DECOMP” in the name of the option. An option name that includes “DECOMP” returns the in-memory size before compression. For more information about SAS Metadata Server journaling, see the *SAS Intelligence Platform: System Administration Guide*.

Although journaling statistics are requested like omaconfig.xml options, they are not documented the same way because, with the exception of <OMA JOURNALPATH=" "/>, they do not configure the SAS Metadata Server—they return statistics only. <OMA JOURNALPATH=" "/> serves a dual purpose. When specified in the omaconfig.xml file with <OMA JOURNALTYPE="SINGLE"/>, it configures the location of the journal file. When specified in the Status method, it reports the location of the journal file. Because of its usage in the omaconfig.xml file, <OMA JOURNALPATH=" "/> must be in all uppercase like other omaconfig.xml options.

The Status journal statistics elements are:

<OMA JOURNALDATAAVAILABLE=" "/>
Returns the number of bytes of data that are in the journal file that have yet to be applied to metadata repositories. To get the decompressed value, submit <OMA JOURNALDECOMPDATAAVAILABLE=" "/>.

<OMA JOURNALENTRYCOUNTER=" "/>
Returns a sequence number that indicates the number of add, update, and delete transactions that have been processed since the SAS Metadata Server was started with journaling enabled. The number includes transactions that are still pending in the journal file and transactions that have been applied to metadata repositories. This sequence number is maintained across server executions.

<OMA JOURNALHISTORICALDATA=" "/>
Returns the total number of bytes of data that have been processed since the journal file was created. This number grows through the life of the journal file. It is not reset when the server is stopped and restarted. To get the decompressed value, submit <OMA JOURNALHISTORICALDECOMPDATA=" "/>.

<OMA JOURNALMAXDATAAVAILABLE=" "/>
Returns the largest number of bytes of data that have been held in the journal file before being committed to metadata repositories. This is the maximum value that JOURNALDATAAVAILABLE has ever attained. To get the decompressed value, submit <OMA JOURNALMAXDECOMPDATAAVAILABLE=" "/>.

<OMA JOURNALMAXFILEENTRYSIZE=" "/>
Returns the number of bytes of data in the largest journal entry that has been written to the journal file. This number can grow through the life of the journal file. It is not reset when the server is stopped and restarted. To get the decompressed value, submit <OMA JOURNALMAXDECOMPFILEENTRYSIZE=" "/>.

<OMA JOURNALPATH=" "/>
Returns the current location of the journal file.

<OMA JOURNALQUEUELENGTH=" "/>
Returns the number of transactions that are waiting in memory to be applied to metadata repositories on disk. The number of transactions in memory reflects the number of transactions that are in the journal file, unless JOURNALSTATE contains the keywords DOTERM, FORCETERM, or TERMDONE. Under these conditions, it is a good idea to check the server log because the journal file might contain entries that are not in the memory queue.

<OMA JOURNALSPACEAVAILABLE=" "/>
Returns either the number of bytes of space left in the journal file if the file is a fixed size, or returns 999999999 if the file is not a fixed size.

<OMA JOURNALSTATE=" "/>
Returns a keyword indicating the internal status of journal-entry processing for troubleshooting. Valid keywords are the following:

Uninitialized
indicates that journaling is not enabled on the SAS Metadata Server.

IDLE
indicates that journaling is enabled. However, there are no journal entries being processed.

BUSY
indicates that journaling is enabled and the SAS Metadata Server is processing journal entries.

DOTERM
indicates that the SAS Metadata Server is accepting journal entries. However, the process that applies the updated transactions to repositories on disk will be terminated after the current transaction is processed. It is a good idea to check journal messages in the server log when you receive this keyword.

FORCETERM
indicates that the SAS Metadata Server is accepting journal entries. However, the process that applies the updated transactions to repositories on disk is in the process of being forcefully terminated, perhaps in the middle of a transaction. It is a good idea to check journal messages in the server log when you receive this keyword.

TERMDONE
indicates that a DOTERM or FORCETERM was successfully processed. The SAS Metadata Server continues to accept entries in the journal file. However, it is not applying the updated transactions to repositories on disk. It is a good idea to check journal messages in the server log when you receive this keyword.

WAIT_IDLE
indicates that the SAS Metadata Server is waiting to perform a request (such as changing a repository's properties) that cannot occur until all outstanding journal entries have been applied to repositories on disk. When all pending transactions have been applied, journaling is returned to an IDLE state.

CRASH_RECOVERY
indicates that the SAS Metadata Server is using journal entries to recover from a server crash.

Server Process Statistics Elements

The following XML elements return SAS Metadata Server process statistics. The metrics are useful for multi-user performance testing.

<OMA USER_CPU_TIME=" "/>
Returns the total seconds consumed by the SAS Metadata Server on non-kernel (user) processing.

<OMA SYSTEM_CPU_TIME=" "/>
Returns the total seconds consumed by the metadata server on kernel (system) processing.

<OMA CURRENT_TIME=" " />
Returns the server's current time as a SAS datetime value (the number of seconds since January 1, 1960).

<OMA CURRENT_MEMORY=" " />
Returns the amount of memory currently being used by the server (in bytes).

<OMA HIGH_WATER_MEMORY=" " />
Returns the highest amount of memory used by the server (in bytes).

<OMA CURRENT_THREAD_COUNT=" " />
Returns the number of active threads on the server.

<OMA HIGH_WATER_THREAD_COUNT=" " />
Returns the highest number of threads used by the server.

<OMA SERVER_STARTED=" "/>
New in SAS 9.4, returns the date and time the metadata server was started as a SAS datetime value (the number of seconds since January 1, 1960).

<OMA SERVER_UPTIME=" "/>
New in SAS 9.4, returns the amount of time that has elapsed since the metadata server was started (in seconds).

<OMA SERVERSTARTPATH=" "/>
Returns the pathname of the directory where the SAS Metadata Server was started. This element must be submitted in uppercase letters.

<OMA TOTAL_IO_COUNT=" "/>
Returns the total number of bytes read and written by the server.

Requesting omaconfig.xml Values

The omaconfig.xml file requests changes to the standard SAS Metadata Server configuration. A configuration option is included in this file only to configure a setting that is a departure from the standard configuration or to provide a value that is required by the standard configuration. The file does not provide all server configuration settings. For information about the options supported in the omaconfig.xml file, see *SAS Intelligence Platform: System Administration Guide*.

The options in the omaconfig.xml file represent permanent choices for the server's configuration. That is, the settings are enforced when the metadata server is started. The settings remain enforced until the server is stopped and the omaconfig.xml file is changed. There are a few exceptions. Some options can be changed on a running server with the Refresh method.

The omaconfig.xml file supports configuration options in three categories.

- General server control, where each option is specified as an XML attribute of an <OMA> XML element.
- Repository manager control, where each option is specified as an XML attribute of an <RPOSMGR> XML element.
- Internal authentication control, where each option is specified as an XML attribute of an <InternalAuthenticationPolicy> XML element.

The Status method obtains values for specified <OMA> and <RPOSMGR> attributes. This is only if the attributes for these XML elements exist in the omaconfig.xml file and if they have been modified with the Refresh method. For information about attributes that can be changed with the Refresh method, see "Refresh Method" on page 244.

The Status method does not return values for <InternalAuthenticationPolicy> attributes. To determine the settings of <InternalAuthenticationPolicy> attributes, use the GetInternalLoginSitePolicies method. See "GetInternalLoginSitePolicies Method" on page 174.

If the requested <OMA> or <RPOSMGR> attribute is missing from the omaconfig.xml file and it has not been modified by the Refresh method, the Status method returns a blank value. A blank value indicates that the option is operating with a standard configuration setting.

When querying omaconfig.xml options, use the case documented for the option in the *SAS Intelligence Platform: System Administration Guide* (usually uppercase). The omaconfig.xml file is case sensitive. If the case does not match, the Status method returns a blank value.

Server Backup and Recovery Options

The SAS Metadata Server supports the ability to back up and recover itself. Server backups are performed in a dedicated server backup thread. As a result, backups do not interrupt the regular operation of the metadata server. Backups are controlled by a scheduler process that runs in a dedicated scheduler thread. Server recovery processes are integrated with the metadata server journal to support roll-forward recovery up to a specified point in time since the restored backup image.

The SAS Metadata Server is installed with a default backup configuration and backup schedule.

The SAS Metadata Server uses four system files to manage backup and recovery processes. In a single SAS Metadata Server configuration, these files are stored in the `SASMeta/MetadataServer` directory. In a clustered SAS Metadata Server configuration, these files are stored in the shared backup location.

MetadataServerBackupConfiguration.xml
: Contains the backup configuration and backup schedule.

MetadataServerBackupHistory.xml
: Contains a history of backup and recovery activity.

MetadataServerBackupManifest.xml
: Contains a record of the repositories and files copied in a backup. This file is created for each backup in the backup's directory and it can be used to validate the backup. A copy of this file for the most recent backup is also kept in the server directory.

MetadataServerRecoveryManifest.xml
: Contains a record of the repositories and files copied in a recovery. This file can be used to validate the recovery. The facility overwrites the last file with the new file each time a recovery is performed.

CAUTION
These four system files should never be opened directly.

The Status method provides the following elements to get information about server backups and about the contents of the files:

<BACKUP *attribute(s)*/>
: queries the progress of an active server backup or server recovery operation. Valid attributes are BYTESTOCOPY=" ", BYTESCOPIED=" ", and ISRUNNING=" ".

<BACKUPCONFIGURATION/>
Returns the active server backup configuration from the MetadataServerBackupConfiguration.xml file.

<METADATASERVERBACKUPCONFIGURATION/>
Returns the active server backup configuration and the backup schedule from the MetadataServerBackupConfiguration.xml file.

<METADATASERVERBACKUPHISTORY/>
Returns a history of the backup and recovery activity on the SAS Metadata Server from the MetadataServerBackupHistory.xml file.

<METADATASERVERBACKUPMANIFEST *optional-attribute*/>
lists the metadata repositories, metadata server journal file, and configuration files copied in the last backup from the MetadataServerBackupManifest.xml file. To get the MetadataServerBackupManifest.xml file from an earlier backup, include an optional BACKUPNAME="*name*" or BACKUPPATH="*pathname*" attribute in <METADATASERVERBACKUPMANIFEST/>. BACKUPNAME="*name*" gets the file for the backup of the specified name from the active backup location. BACKUPPATH="*pathname*" gets the file for a backup that is stored in a location other than the active backup location. Pathnames are considered to be relative to the **`SASMeta/MetadataServer`** directory. To get a backup from another backup location, specify an absolute pathname.

<METADATASERVERRECOVERYMANIFEST/>
lists the metadata repositories, metadata server journal file, and configuration files copied in the last recovery from the MetadataServerRecoveryManifest.xml file.

<SCHEDULE EVENT="Backup" WEEKDAY*n*=" "/>
Returns the backup times stored in the backup schedule for the specified days. The facility has a weekly backup schedule that specifies backup times in WeekDay*n*="*timevalue*" attributes, where *n* represents a day of the week. WeekDay1 is Sunday and WeekDay7 is Saturday. To get the backup schedule for a particular day, specify the WeekDay*n*=" " attribute with an appropriate number. The facility returns four-digit time values based on a 24-hour clock in the time zone in which the server is running. If more than one backup is scheduled in a day, the time values are separated by semicolons. An “R” appended to a time value indicates that a REORG is scheduled to be performed with the backup.

To enable clients to retrieve specific records from the history and manifest files, the XML elements requesting these files support limited XPATH search syntax specified in an XPATH=" " attribute. Two syntax elements are supported: XPATH [POSITION()=LAST()] and XPATH [@ATTRIBUTE='*value*']. The XPATH [POSITION()=LAST()] syntax can be specified in a <METADATASERVERBACKUPHISTORY/> request to get the last backup in the MetadataServerBackupHistory.xml file.

```
<MetadataServerBackupHistory XPath="MetadataServerBackupManifest/Backups/
Backup[POSITION()=LAST()]" />
```

The XPATH [@ATTRIBUTE='*value*'] syntax can return information about specific files in a backup or recovery. For example, the following request gets information about the adminUsers.txt file from the MetadataServerBackupManifest.xml file:

```
<MetadataServerBackupManifest XPath="MetadataServerBackupManifest/Backups/
Backup/ConfigurationFiles/File[@Name='adminUsers.txt'" />
```

You must familiarize yourself with the structure of the history and manifest files to build XPATH queries.

See "<Scheduler/> XML Element" on page 269 for information about how to check the health of the server backup thread.

<Scheduler/> XML Element

A SAS Metadata Server creates a separate process or thread to launch the server backup process and to track and send alert e-mail reminders about the alert condition `the journal commit stopped running`. The <Scheduler/> XML element can be used to get information about the backup process or alert conditions.

<Scheduler Ping=" "/>
: Specify the PING= attribute in the <Scheduler/> XML element to determine whether the scheduler is viable. PING= sends a packet and waits for a response. Possible return values are Alive, TimeOut, Down, or Unconfigured.

<Scheduler><AlertConditions/></Scheduler>
: Specify the <AlertConditions/> subelement to determine whether an alert condition exists on the specified SAS Metadata Server. If an alert condition exists, the subelement returns an <AlertCondition/> XML element and an <ExpirationTime/> XML element. The <AlertCondition/> element includes the error and a datetime value representing the time at which the error occurred. The <ExpirationTime/> element includes the server's scheduled termination time.

Examples

Standard Interface Example

The following example shows how the Status method is issued in the standard interface:

```
<! -- Default values returned by Status method in SAS 9.4  -->
inmeta=' ';
outmeta=' ';
options=' ';

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

The Status method is issued without specifying options in the INMETA parameter to show the default behavior of the method. The following is the output from the request:

```
<ModelVersion>15.01</ModelVersion>
<PlatformVersion>9.4.0.0</PlatformVersion>
<ServerState>ONLINE</ServerState>
<PauseComment/>
<ServerLocale>en_US</ServerLocale>
```

DoRequest Examples

The following examples are formatted for the INMETADATA parameter of the DoRequest interface.

This is the same method call that was issued in the standard interface example.

```
<!-- Get default values returned by the Status method  -->
<Status>
<Metadata/>
<Options/>
</Status>
```

This Status method call requests omaconfig.xml values and <SERVERLOCALE/>:

```
<!-- Get omaconfig.xml values -->
<Status>
<Metadata>
<OMA MAXACTIVETHREADS=""/>
<OMA JOURNALTYPE=""/>
<OMA ALERTEMAIL=""/>
<RPOSMGR
LIBREF=""
ENGINE=""
PATH=""
OPTIONS=""/>
<SERVERLOCALE/>
</Metadata>
<Options/>
</Status>
```

This Status method call requests the value of the omaconfig.xml <OMA JOURNALTYPE="SINGLE | ROLL_FORWARD | NONE"/> server configuration option and journaling statistics:

```
<!-- Get journaling statistics-->

<Status>
<Metadata>
<OMA JOURNALTYPE=""
     JOURNALPATH=""
     JOURNALSTATE=""
     JOURNALQUEUELENGTH=""
     JOURNALDATAAVAILABLE=""
     JOURNALSPACEAVAILABLE=""
     JOURNALENTRYCOUNTER=""/>
</Metadata>
<Options/>
</Status>
```

The following Status request gets the backup parameters and backup schedule that are active on the SAS Metadata Server:

```
<!-- Get server backup configuration and schedule -->
<Status>
  <Metadata>
    <MetadataServerBackupConfiguration/>
  </Metadata>
  <Options/>
</Status>
```

The following Status request gets the backup schedule for Tuesday:

```
<!-- Get the Tuesday backup schedule -->
<Status>
  <Metadata>
```

```
    <Schedule Event="Backup" Weekday3=""/>
  </Metadata>
  <Options/>
</Status>
```

This Status method call gets information about how long the metadata server has been running:

```
<!-- Get server process statistics -->
<Status>
  <Metadata>
     <OMA Server_Started=""/>
     <OMA Server_Uptime=""/>
  </Metadata>
</Status>
```

The following Status method call gets detailed information about the last synchronization check that was performed on the connected server node. It queries the node to see whether any alert conditions have been recorded and whether a termination time has been scheduled for the node:

```
<Status>
  <Metadata>
    <SynchCheck>
      <Results Detail="FULL"/>
    </SynchCheck>
    <Scheduler Ping=" ">
     <AlertConditions/>
    </Scheduler>
  </Metadata>
</Status>
```

Related Methods

- “GetInternalLoginSitePolicies Method” on page 174

Stop Method

Short Description

Shuts down the SAS Metadata Server.

Category

IServer interface server control methods

Syntax

```
rc=Stop(options);
```

Table 10.5 *Method Parameters*

Parameter	Type	Direction	Description
options	C	in	<CLUSTER/> Specifies to execute the Stop request on all SAS Metadata Servers in the cluster. In a single SAS Metadata Server configuration, this option is ignored. <SINGLE_SERVER/> Specifies to execute the Stop request on the connected SAS Metadata Server only.

Details

A user must have administrative user status on the SAS Metadata Server to issue the Stop method.

When the <SINGLE_SERVER/> element is specified, the method affects the SAS Metadata Server on which it is executed. When the <CLUSTER/> element is specified, the method affects all SAS Metadata Servers that are configured in the server cluster.

A user is encouraged to specify one of these elements. If neither element is specified, the behavior is undefined. If <CLUSTER/> is specified but a server cluster is not configured, then the method is executed locally.

The Stop method is the recommended way to make an individual metadata server in a cluster temporarily available. If the metadata server needs maintenance, it can be restarted in stand-alone mode with the -STARTNOCLUSTER option, and then returned to the cluster after maintenance is performed. For more information about maintaining metadata servers in a cluster, see the *SAS Intelligence Platform: System Administration Guide*.

A return code of 0 indicates that the SAS Metadata Server was successfully stopped. A return code other than 0 indicates that the SAS Metadata Server failed to stop.

The Stop method is available only in the standard interface.

Examples

The following example shows how to stop the connected SAS Metadata Server:

```
<!-- Stops the current metadata server -->
options='<SINGLE_SERVER/>'
rc=serverObject.Stop(options);
```

The following example shows how to stop all SAS Metadata Servers in the cluster:

```
<!-- Stop all servers in the cluster-->
options='<CLUSTER/>'
rc=serverObject.Stop(options);
```

If the <CLUSTER/> element is encountered in a single server configuration, the method stops the connected SAS Metadata Server.

PART 4

IOMI Server Interface Usage

11

Adding Metadata Objects

Overview of Adding Metadata

The SAS Open Metadata Interface provides the AddMetadata method to enable you to instantiate SAS Metadata Model metadata types in a SAS Metadata Repository. The AddMetadata method creates SAS Metadata Model metadata objects as defined in an input XML metadata property string. Depending on the content of the input metadata property string, the method can be used to instantiate a single SAS Metadata Model object, or many related or unrelated SAS Metadata Model objects. It is the responsibility of the caller to build an appropriate input metadata property string to represent an object in a SAS Metadata Repository.

Most resources and information assets that are managed in a SAS Metadata Repository are described by a set of associated SAS Metadata Model metadata

types, rather than by just one metadata type. This set of associated metadata types is referred to as a *logical metadata definition*.

SAS uses an object type dictionary, called the SAS type dictionary, to ensure consistency among the logical metadata definitions that represent common and shared objects in SAS applications. The SAS type dictionary is described in Chapter 3, “Using Interfaces That Read and Write Metadata in SAS 9.4,” on page 19.

A logical metadata definition that you create with AddMetadata is considered a custom definition. It might not conform to the SAS type dictionary. To create object definitions that conform to the dictionary, SAS recommends that you use SAS wizards and procedures to create metadata instead.

The information in this chapter is provided as background information about the internal processes that create metadata and for customers who want to create metadata that is not managed by the SAS type dictionary.

Using the AddMetadata Method

Steps to Create a Metadata Object

To create a metadata object with the AddMetadata method, you must provide the following information:

- A metadata property string that defines the object(s) to be created.
- An identifier that indicates the metadata repository in which the object will be stored.
- The namespace in which to process the request.
- The OMI_TRUSTED_CLIENT (268435456) flag. The OMI_TRUSTED_CLIENT flag is required to create and update a metadata object in a SAS Metadata Repository.

This information is passed as AddMetadata parameters. The parameters are displayed here as XML elements that can be submitted to the SAS Metadata Server in the INMETADATA parameter of the DoRequest method:

```
<AddMetadata>

<Metadata>metadata_property_string</Metadata>

<Reposid>repository_identifier</Reposid>
   <Ns>namespace</Ns>
   <Flags>268435456</Flags>
   <Options/>
</AddMetadata>
```

To create a metadata object describing an application element, you specify the following information:

- a metadata property string that defines a SAS Metadata Model metadata object in the <METADATA> element. The property string defines attribute values for the object and associations between this object and important related objects.
- a metadata repository identifier in the <REPOSID> element
- the SAS namespace in the <NS> element
- the OMI_TRUSTED_CLIENT (268435456) flag in the <FLAGS> element

The input metadata property string must be formatted in XML. For instructions to code an XML metadata property string, see "Constructing a Metadata Property String" on page 76. At a minimum, the metadata property string must specify one metadata type and any required attributes and associations for the object to be created. Most metadata types have required attributes. Some metadata types also have required associations. For information about required and optional properties of the metadata types describing application elements, see the metadata type descriptions in "Alphabetical Listing of SAS Namespace Metadata Types" in the *SAS Metadata Model: Reference*.

Although it is not required, it is recommended that you provide values for all of an object's attributes, and define as many associations as possible in the metadata property string. This makes your metadata more meaningful. Before creating objects, see "Selecting Metadata Types to Represent Application Elements" on page 282.

Creating Associations While Creating Objects

The metadata property string that defines a SAS Metadata Model object can also define associations to related SAS Metadata Model objects. The related objects can already exist in the SAS Metadata Repository, or they can be created new along with the main object in the property string.

As discussed in "Constructing a Metadata Property String" on page 76, an association is defined by including XML subelements that specify an association name and an associated metadata type in the property string that defines a SAS Metadata Model object.

The following is an example of a metadata property string that includes XML subelements that define an association:

```
<MetadataType Name="Name-of-primary-object" Desc="New object created using AddMetadata">
     <AssociationName>
        <AssociatedMetadataType Name="Name-of-associated-object"
          Desc="New associated object that is created by AddMetadata" />
     </AssociationName>
   </MetadataType>
```

The AddMetadata method creates a new object for every property string that is submitted in the <METADATA> element, unless the ObjRef attribute is included in the property string. ObjRef is supported only in a nested property string. It specifies that the object instance is an existing metadata object. It instructs the SAS Metadata Server to create an object reference to the specified object without modifying the specified object's other properties.

The SAS Metadata Server creates a new associated object in all of the following cases:

- When you omit the Id attribute from the metadata property string.
- When you specify the Id attribute with a null value (Id=" ").
- When you specify a symbolic name in the Id attribute (Id="$Table").
- When you specify a real value in the Id attribute (Id="A1000001").

Creating Cross-Repository References

The default behavior of the AddMetadata method is to create the main metadata object and all references to new and existing objects in the repository specified in the <REPOSID> element. You can also create associations that reference new and existing objects in other repositories. Associations between objects that exist in different repositories are referred to as cross-repository references. A cross-repository reference is created by including the appropriate repository identifier in the associated object definition.

- To create a cross-repository reference to an existing object, specify its 17-character metadata identifier in the form *Reposid.Instanceid* in the ObjRef attribute of the XML subelement defining the associated object. The *Reposid* portion of the metadata identifier specifies the other repository.
- To create a cross-repository reference and a new object in another repository, specify the target repository identifier and a symbolic name for the new object using the Id attribute. For example, Id="*Reposid*.$Table". Use of the Id attribute with a symbolic name is equivalent to passing a null value in the Id attribute: it indicates to the SAS Metadata Server that a new object is to be created.

The objects and associations that make up an application entity's logical metadata definition should be created in the same metadata repository as the main object. Cross-repository references should be reserved for associations between application entities.

Symbolic Names

A symbolic name is an alias that is preceded by a dollar sign ($). You assign a symbolic name to a metadata object to reference it before it is created. Symbolic names are used to create associations and new associated objects in other repositories. They also enable you to create references between multiple, unrelated metadata objects in a single XML request. For more information about this type of usage, see "Example of an AddMetadata Request That Creates Multiple, Unrelated Metadata Objects" on page 288.

When used to create an association and a new associated object in another repository, the symbolic name has the form *Reposid.$Alias*. It is specified in the Id attribute of the XML subelement that defines the associated object.

When used to create multiple, unrelated metadata objects in the same repository, the simple form *$Alias* is used.

Alias can be any name, as long as it is preceded by a dollar sign ($). After the successful completion of AddMetadata or UpdateMetadata processing, the alias and any references to it are automatically replaced with a real object identifier.

Example Metadata Property Strings

The following is an example of a metadata property string that creates a SAS Metadata Model object and an association to an existing SAS Metadata Model object in another repository:

```
<MetadataType Name="Name-of-primary-object" Desc="New object created using AddMetadata">
   <AssociationName>
     <AssociatedMetadataType Objref="Reposid.Objectid"/>
   </AssociationName>
</MetadataType>
```

Note the use of the ObjRef attribute and the fact that no other attributes are specified. When ObjRef is used, the SAS Metadata Server ignores any additional attributes that might be specified. The object identifier in the ObjRef attribute includes both the repository identifier and the object instance identifier of the target object.

The following is an example of a metadata property string that creates a SAS Metadata Model object, an association, and a new associated SAS Metadata Model object in another repository:

```
<MetadataType Name="Name-of-primary-object"  Desc="New object created using AddMetadata">
   <AssociationName>
     <AssociatedMetadataType Id="Reposid.$SymbolicName" Name="Name-of-associated-object"
     Desc="Associated object that is created by AddMetadata"/>
   </AssociationName>
</MetadataType>
```

The portion of the property string that identifies the associated object specifies the Id attribute with the repository identifier and a symbolic name for the new object. (You can determine the available repositories and their unique identifiers by issuing the GetRepositories method. For more information, see “Using GetRepositories to List the Registered Repositories” on page 311.) Because a new object is created, you must specify at least a Name value for the associated object and you should include values for other attributes.

Creating Multiple Associated Objects

To define multiple associations for an object, stack the associated object definitions within the primary object definition as follows:

```
<Metadata>
   <MetadataType Name="String" Desc="String">
     <AssociationName1>
        <AssociatedMetadataType Name="String" Desc="String"/>
     </AssociationName1>
     <AssociationName2>
        <AssociatedMetadataType Name="String" Desc="String"/>
     </AssociationName2>
  </MetadataType>
</Metadata>
```

Creating Multiple, Unrelated SAS Metadata Model Objects in an AddMetadata Request

To create multiple, unrelated SAS Metadata Model metadata objects in an AddMetadata request, stack the metadata property strings that define the metadata objects in the <METADATA> element as follows:

```
<Metadata>
  <MetadataType1 Name="String" Desc="String">
   <AssociationName>
     <AssociatedMetadataType Name="String" Desc="String"/>
   </AssociationName>
 </MetadataType1>
 <MetadataType2 Name="String" Desc="String">
   <AssociationName>
     <AssociatedMetadataType Name="String" Desc="String"/>
   </AssociationName>
 </MetadataType2>
</Metadata>
```

Selecting Metadata Types to Represent Application Elements

The SAS Metadata Model defines approximately 170 metadata types that represent application elements. The metadata types are intended to be used in combination by clients to create metadata describing application data or entities used by an application.

The metadata programmer begins by selecting a metadata type that most closely describes the entity for which he wants to store metadata. For example, when creating metadata describing a data source, a metadata programmer might select the PhysicalTable metadata type to represent data in a table. This becomes the primary or top-level object in the metadata definition. He then needs to examine the SAS Metadata Model to select metadata types that provide supporting information. For example, to describe each table's columns, he might use the Column metadata type. If the tables have passwords, he might use the SASPassword metadata type, and so on. These are considered secondary objects in the metadata definition.

Once he has identified all of the metadata types necessary to fully describe the data source, he can use the AddMetadata method to create metadata objects representing each metadata type, and to associate the objects with one another.

To assist metadata programmers in building consistent metadata definitions, the SAS Metadata Model categorizes metadata types as being either primary or secondary:

- Metadata types that are subtypes of the PrimaryType supertype are intended to be used as the topmost object in a metadata definition, or to describe an

application element that provides supporting information, but can be referenced, secured, and deleted independently of the primary object that it supports.

- Metadata types that are subtypes of the SecondaryType supertype provide supporting information for a primary type and are never referenced directly within a SAS application.

For a list of the metadata types in each category, see the descriptions of the PrimaryType and SecondaryType metadata types in the *SAS Metadata Model: Reference*.

The PrimaryType metadata type defines attributes and associations that support the management of the entities described by the metadata definitions. For more information about these attributes and associations and how they should be used, see "PrimaryType and SecondaryType Abstract Types" in the model reference.

SAS uses a type dictionary to manage the logical metadata definitions that are used to represent object types that are common to or shared by SAS Intelligence Platform applications. To determine the primary metadata types used to represent common and shared object types, view the type dictionary. The dictionary is in the /System/Types folder of the SAS Folders tree.

Example of an AddMetadata Request That Creates a SAS Metadata Model Object

The following AddMetadata request creates a metadata object describing a SAS library. A SAS library is represented in the SAS Metadata Model by the SASLibrary metadata type.

```
<AddMetadata>
 <Metadata>
   <SASLibrary
      Name="NW Sales"
      Desc="NW region sales data"
      Engine="base"
      IsDBMSLibname="0"
      Libref="nwsales"
      IsPreassigned="0"
      PublicType="Library"/>
 </Metadata>
 <Reposid>A0000001.A53TPPVI</Reposid>
 <NS>SAS</NS>
 <Flags>268435456</Flags>
 <Options/>
</AddMetadata>
```

In this method call, consider the following:

- The <METADATA> element specifies the metadata property string. In this string, SASLibrary is the metadata type and Name, Desc, Engine, IsDBMSLibname, Libref, and PublicType are attributes of the SASLibrary metadata type.
- The <REPOSID> element specifies the unique identifier of the repository in which to create the metadata object.

- The <NS> element identifies the namespace to process the request. The SAS namespace contains metadata types that define application elements.
- The <FLAGS> element specifies the OMI_TRUSTED_CLIENT (268435456) flag.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the ADDMETADATA method. -->

<SASLibrary Name="NW Sales" Desc="NW region sales data" Engine="base"
  IsDBMSLibname="0" Libref="nwsales" IsPreassigned="0" PublicType="Library"
Id="A53TPPVI.A1000001"/>
```

The output string mirrors the input string, with the exception that a two-part object instance identifier is assigned to the new object in the form:

```
Reposid.Objectid
```

Reposid is the unique repository identifier. *Objectid* is the unique object instance identifier. You will use this unique object identifier any time you need to reference the metadata object. In “Example of an AddMetadata Request That Creates an Object and an Association to an Existing Object” on page 284, this identifier is used to create an association to the object.

Example of an AddMetadata Request That Creates an Object and an Association to an Existing Object

The following AddMetadata request creates a ResponsibleParty object, and then associates it with the SASLibrary object created in “Example of an AddMetadata Request That Creates a SAS Metadata Model Object” on page 283. The ResponsibleParty metadata type is used to associate a set of Person objects with a role or responsibility. This ResponsibleParty object is created with the role of "Owner".

```
<AddMetadata>
 <Metadata>
   <ResponsibleParty Name="LibraryAdministrator"
      Desc="NW Region Sales Data"
      Role="Owner">
     <Objects>
       <SASLibrary ObjRef="A53TPPVI.A1000001"/>
     </Objects>
   </ResponsibleParty>
 </Metadata>
 <Reposid>A0000001.A53TPPVI</Reposid>
 <NS>SAS</NS>
 <Flags>268435456</Flags>
 <Options/>
</AddMetadata>
```

In this method call, the <REPOSID>, <NS>, and <FLAGS> elements contain the same values as in “Example of an AddMetadata Request That Creates a SAS

Metadata Model Object" on page 283. In the <METADATA> property string, note the following:

- <RESPONSIBLEPARTY> is the metadata type. Name, Desc, and Role are attributes of the ResponsibleParty metadata type. Name and Role are required attributes.
- <OBJECTS> is the association name and is specified as a subelement of <RESPONSIBLEPARTY>.
- <SASLIBRARY> is the metadata type of the associated metadata object and is specified as a subelement of <OBJECTS>. The ObjRef attribute in the <SASLIBRARY> subelement informs the SAS Metadata Server that the SASLibrary object is an existing object. The server creates the association without modifying any of the object's other properties.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the ADDMETADATA method. -->

<ResponsibleParty Name="LibraryAdministrator" Desc="NW Region Sales Data"
  Role="Owner" Id="A53TPPVI.A2000001">
  <Objects>
    <SASLibrary ObjRef="A53TPPVI.A1000001"/>
  </Objects>
</ResponsibleParty>
```

The output string mirrors the input string with the exception that a two-part metadata object identifier is assigned to the ResponsibleParty object.

Example of an AddMetadata Request That Creates Multiple, Related Metadata Objects

The following AddMetadata request creates a table object, column objects, and an association to the previously defined SASLibrary object in a single method call. The SAS Metadata Model supports several metadata types for describing tables. In this example, the PhysicalTable metadata type is used to represent a table that is materialized in a file system. A PhysicalTable object is associated to a Column object with a Columns association. A PhysicalTable object is associated to a SASLibrary object with a TablePackages association.

```
<AddMetadata>
 <Metadata>
   <PhysicalTable Name="Sales Offices" Desc="Sales offices in NW region"
     PublicType="Table">
     <TablePackages>
       <SASLibrary ObjRef="A53TPPVI.A1000001"/>
     </TablePackages>
     <Columns>
        <Column
           Name="City"
           Desc="City of Sales Office"
           ColumnName="City"
           SASColumnName="City"
           ColumnType="12"
```

```
                SASColumnType="C"
                ColumnLength="32"
                SASColumnLength="32"
                SASFormat="$Char32."
                SASInformat="$32."
              PublicType="Column"/>
            <Column
                Name="Address"
                Desc="Street Address of Sales Office"
                ColumnName="Address"
                SASColumnName="Street_Address"
                ColumnType="12"
                SASColumnType="C"
                ColumnLength="32"
                SASColumnLength="32"
                SASFormat="$Char32."
                SASInformat="$32."
              PublicType="Column"/>
            <Column
                Name="Manager"
                Desc="Name of Operations Manager"
                ColumnName="Manager"
                SASColumnName="Manager"
                ColumnType="12"
                SASColumnType="C"
                ColumnLength="32"
                SASColumnLength="32"
                SASFormat="$Char32."
                SASInformat="$32."
              PublicType="Column"/>
            <Column
              Name="Employees"
              Desc="Number of employees"
              ColumnName="Employees"
              SASColumnName="Employees"
              ColumnType="6"
              SASColumnType="N"
              ColumnLength="3"
              SASColumnLength="3"
              SASFormat="3.2"
              SASInformat="3.2"
              PublicType="Column"/>
            </Columns>
        </PhysicalTable>
    </Metadata>
    <Reposid>A0000001.A53TPPVI</Reposid>
    <NS>SAS</NS>
    <Flags>268435456</Flags>
    <Options/>
  </AddMetadata>
```

In the request, the <REPOSID>, <NS>, and <FLAGS> elements contain the same values as in "Example of an AddMetadata Request That Creates a SAS Metadata Model Object" on page 283. In the <METADATA> property string, note the following:

- <PHYSICALTABLE> is the metadata type. Name, Desc, and PublicType are attributes of the PhysicalTable metadata type.

- <TABLEPACKAGES> is the association name that creates the association to the SASLibrary object. The ObjRef attribute in the <SASLIBRARY> subelement informs the SAS Metadata Server that the association is being created to an existing object.
- <COLUMNS> is the association name that creates the associations to the Column objects. The column definitions are nested under the Columns association name.
- Four Column objects are created. For each object, consider the following:
  - Name is a required attribute.
  - The ColumnName, ColumnType, and ColumnLength attributes describe the names and values of the items in a DBMS.
  - The SASColumnName, SASColumnType, and SASColumnLength attributes indicate their corresponding values in a SAS table.
  - A ColumnType value of 12 indicates VARCHAR. A ColumnType value of 6 indicates FLOAT.

  For more information about the properties for the PhysicalTable and Column metadata types, see the SAS Metadata Model documentation.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the ADDMETADATA method. -->

<PhysicalTable Name="Sales Offices" Desc="Sales offices in NW region"
 Id="A53TPPVI.A4000001">
 <TablePackages>
  <SASLibrary ObjRef="A53TPPVI.A1000001"/>
 </TablePackages>
 <Columns>
  <Column Name="City" Desc="City of Sales Office" ColumnName="City"
    SASColumnName="City"  ColumnType="12" SASColumnType="C" ColumnLength="32"
    SASColumnLength="32" SASFormat="$Char32." SASInformat="$32."
    PublicType="Column" Id="A53TPPVI.A5000001">
    <Table>
     <PhysicalTable ObjRef="A53TPPVI.A4000001"/>
    </Table>
  </Column>
  <Column Name="Address" Desc="Street Address of Sales Office"
    ColumnName="Address" SASColumnName="Street_Address" ColumnType="12"
    SASColumnType="C" ColumnLength="32" SASColumnLength="32" SASFormat="$Char32."
    SASInformat="$32." PublicType="Column"  Id="A53TPPVI.A5000002">
    <Table>
     <PhysicalTable ObjRef="A53TPPVI.A4000001"/>
    </Table>
  </Column>
  <Column Name="Manager" Desc="Name of Operations Manager" ColumnName="Manager"
    SASColumnName="Manager" ColumnType="12" SASColumnType="C" ColumnLength="32"
    SASColumnLength="32" SASFormat="$Char32." SASInformat="$32."
    PublicType="Column" Id="A53TPPVI.A5000003">
    <Table>
     <PhysicalTable ObjRef="A53TPPVI.A4000001"/>
    </Table>
  </Column>
  <Column Name="Employees" Desc="Number of employees" ColumnName="Employees"
    SASColumnName="Employees" ColumnType="6" SASColumnType="N" ColumnLength="3"
```

```
    SASColumnLength="3" SASFormat="3.2" SASInformat="3.2" PublicType="Column"
    Id="A53TPPVI.A5000004">
    <Table>
     <PhysicalTable ObjRef="A53TPPVI.A4000001"/>
    </Table>
  </Column>
</Columns>
</PhysicalTable>
```

The output string mirrors the input string, with the exception that a two-part metadata object identifier is assigned to each new metadata object.

Example of an AddMetadata Request That Creates Multiple, Unrelated Metadata Objects

The following method call shows another way to format an AddMetadata request that creates multiple objects. The request creates a second table object, named Sales Associates, and creates objects representing the table's columns by stacking their metadata property strings. A Column object cannot be created without an association to a table object. Therefore, a symbolic name is assigned to the PhysicalTable object to enable the Column objects to reference the PhysicalTable object before it is created.

```
<AddMetadata>
 <Metadata>
 <PhysicalTable Id="$Employees" Name="Sales Associates"
     Desc="Sales associates in NW region" PublicType="Table">
     <TablePackages>
       <SASLibrary ObjRef="A53TPPVI.A1000001"/>
     </TablePackages>
  </PhysicalTable>

  <Column
     Name="Name"
     Desc="Name of employee"
     ColumnName="Employee_Name"
     SASColumnName="Employee"
     ColumnType="12"
     SASColumnType="C"
     ColumnLength="32"
     SASColumnLength="32"
     SASFormat="$Char32."
     SASInformat="$32."
     PublicType="Column" >
     <Table>
        <PhysicalTable ObjRef="$Employees"/>
     </Table>
  </Column>

  <Column
     Name="Address"
     Desc="Home Address"
```

```
      ColumnName="Employee_Address"
      SASColumnName="Home_Address"
      ColumnType="12"
      SASColumnType="C"
      ColumnLength="32"
      SASColumnLength="32"
      SASFormat="$Char32."
      SASInformat="$32."
      PublicType="Column">
      <Table>
         <PhysicalTable ObjRef="$Employees"/>
      </Table>
   </Column>

   <Column
      Name="Title"
      Desc="Job grade"
      ColumnName="Title"
      SASColumnName="Title"
      ColumnType="12"
      SASColumnType="C"
      ColumnLength="32"
      SASColumnLength="32"
      SASFormat="$Char32."
      SASInformat="$32."
      PublicType="Column">
      <Table>
         <PhysicalTable ObjRef="$Employees"/>
      </Table>
   </Column>

 </Metadata>
 <Reposid>A0000001.A53TPPVI</Reposid>
 <NS>SAS</NS>
 <Flags>268435456</Flags>
 <Options/>
</AddMetadata>
```

In this method call, the <REPOSID>, <NS>, and <FLAGS> elements contain the same values as in “Example of an AddMetadata Request That Creates a SAS Metadata Model Object” on page 283. In the <METADATA> element, note the following:

- There are multiple metadata property strings, stacked one on top of the other.
- The metadata property string defining the PhysicalTable object is the topmost string. This property string includes an Id attribute that assigns the symbolic name $Employees. Name and PublicType are required attributes.
- Separate metadata property strings define each Column object. Each string defines the unique attributes of the column and the global Name and PublicType attributes.
- Each Column definition defines a Table association to the PhysicalTable object by specifying the ObjRef attribute and referencing the symbolic name $Employees.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the ADDMETADATA method. -->
```

```
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates"
  Desc="Sales associates in NW region" PublicType="Table">
 <TablePackages>
  <SASLibrary ObjRef="A53TPPVI.A1000001"/>
 </TablePackages>
</PhysicalTable>
<Column Name="Name" Desc="Name of employee" ColumnName="Employee_Name"
  SASColumnName="Employee" ColumnType="12" SASColumnType="C" ColumnLength="32"
  SASColumnLength="32" SASFormat="$Char32." SASInformat="$32." PublicType="Column"
  Id="A53TPPVI.A5000005">
  <Table>
   <PhysicalTable ObjRef="A53TPPVI.A4000002"/>
  </Table>
</Column>
<Column Name="Address" Desc="Home Address" ColumnName="Employee_Address"
  SASColumnName="Home_Address" ColumnType="12" SASColumnType="C" ColumnLength="32"
  SASColumnLength="32" SASFormat="$Char32." SASInformat="$32."
  PublicType="Column" Id="A53TPPVI.A5000006">
  <Table>
   <PhysicalTable ObjRef="A53TPPVI.A4000002"/>
  </Table>
</Column>
<Column Name="Title" Desc="Job grade" ColumnName="Title" SASColumnName="Title"
  ColumnType="12" SASColumnType="C" ColumnLength="32" SASColumnLength="32"
  SASFormat="$Char32." SASInformat="$32." PublicType="Column" Id="A53TPPVI.A5000007">
  <Table>
   <PhysicalTable ObjRef="A53TPPVI.A4000002"/>
  </Table>
</Column>
```

The symbolic name is replaced with the PhysicalTable object's unique object identifier in the output everywhere that it was used.

Example of an AddMetadata Request That Creates an Association to an Object in Another Repository

This example contains two AddMetadata method calls:

- The first method call creates a second repository.
- The second method call creates a Document object in the new repository and associates the Document object with the SASLibrary object created in “Example of an AddMetadata Request That Creates a SAS Metadata Model Object” on page 283. A Document object has an Objects association to a SASLibrary object.

This method call creates the second repository:

```
<AddMetadata>
  <Metadata>
    <RepositoryBase
```

```
      Name="Test repository 2"
      Desc="Second test repository."
      Path="C:\testdat\repository2"
      RepositoryType="Custom">
    </RepositoryBase>
   </Metadata>
   <Reposid>A0000001.A0000001</Reposid>
   <NS>REPOS</NS>
   <!-- OMI_TRUSTED_CLIENT flag -->
   <Flags>268435456</Flags>
   <Options/>
</AddMetadata>
```

In the method call, note the following:

- The <METADATA> element submits a property string that defines a RepositoryBase object. Path and RepositoryType are required attributes.
- The <REPOSID> element specifies the SAS Repository Manager identifier.
- The <NS> element specifies the REPOS namespace.
- The <FLAGS> parameter specifies the OMI_TRUSTED_CLIENT flag (268435456).

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the ADDMETADATA method. -->

<RepositoryBase Name="Test repository 2" Desc="Second test repository."
  Path="C:\testdat\repository2" Id="A0000001.A5KD78HW" Access="0"
 RepositoryType="Custom"/>
```

Test repository 2 is assigned the unique repository identifier A0000001.A5KD78HW.

This AddMetadata method call creates the Document object in Test repository 2:

```
<AddMetadata>
 <Metadata>
    <Document  Name="Sales Summary" Desc="Summary of Sales Activity in the NW Region"
  PublicType="Document">
       <Objects>
        <SASLibrary ObjRef="A53TPPVI.A1000001"/>
       </Objects>
    </Document>
 </Metadata>
 <Reposid>A0000001.A5KD78HW</Reposid>
 <NS>SAS</NS>
 <Flags>268435456</Flags>
 <Options/>
</AddMetadata>
```

In this method call, note the following:

- The <METADATA> subelement that defines the Objects association to the SASLibrary object specifies the ObjRef attribute. The value in the ObjRef attribute is in the form *Reposid.ObjectId*, where *Reposid* is the repository identifier of Test repository 1 and *ObjectId* is the SASLibrary object's 8–character identifier.
- The <REPOSID> element specifies the unique repository identifier assigned to Test repository 2.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the ADDMETADATA method. -->

<Document Name="Sales Summary" Desc="Summary of Sales Activity in the NW Region"
  PublicType="Document" Id="A5KD78HW.A1000001">
 <Objects>
   <SASLibrary ObjRef="A53TPPVI.A1000001"/>
 </Objects>
</Document>
```

Document object A5KD78HW.A1000001 was successfully created in repository A0000001.A5KD78HW.

12

Updating Metadata Objects

Overview of Updating Metadata

The SAS Open Metadata Interface provides the UpdateMetadata method for updating existing metadata objects.For reference information, see "UpdateMetadata Method" on page 139. With UpdateMetadata, you can do the following:

- modify an existing metadata object's attributes
- add an association between two existing metadata objects
- add an association between an existing metadata object and a new metadata object
- modify an associated object's properties
- remove an association

The UpdateMetadata method does not allow you to create new objects. For information about creating a metadata object, see Chapter 11, "Adding Metadata Objects," on page 277. UpdateMetadata can create associated objects indirectly as a result of defining an association.

The UpdateMetadata method cannot delete a metadata object. To delete a metadata object, you must use the DeleteMetadata method. For more information, see Chapter 19, "Deleting Metadata Objects," on page 395. The UpdateMetadata method might indirectly delete dependent objects to enforce cardinality rules when

an association is deleted. For example, if a table is updated to remove an association to a column, then the Column object, which is dependent on the table, is deleted as well. However, a Column object cannot be updated to remove its association to a table and, as a result, be deleted.

Using the UpdateMetadata Method

Required Attributes and Associations

The UpdateMetadata method can be issued in the REPOS namespace to modify the properties of a metadata repository or in the SAS namespace to modify the properties of an object describing an application element.

You can add or modify any attribute or association that is not designated as required in the metadata type documentation. For information about which metadata types have required attributes and associations, see “Required Attributes and Associations” in the *SAS Metadata Model: Reference*. An association that is designated as required typically indicates that the object is a dependent object. To remove a required association, you must delete the dependent object with the DeleteMetadata method. However, a dependent object’s other attributes and associations can be modified with UpdateMetadata.

Using UpdateMetadata to Modify Attribute Values

To modify an object’s attributes, specify the metadata type, object ID, and the names of the attributes that you want to modify and their new values in a metadata property string. Submit the metadata property string to the UpdateMetadata method in the INMETADATA parameter. Here is an example of such an UpdateMetadata method call that is formatted for the DoRequest interface:

```
<UpdateMetadata>
  <Metadata>
    <Metadata_Type Id="reposid.objectid" Attribute1="new_value"
                   Attribute2="new_value" Attribute3="new_value"/>
  </Metadata>
  <NS>SAS</NS>
  <--- OMI_TRUSTED_CLIENT Flag --->
  <Flags>268435456</Flags>
  <Options/>
</UpdateMetadata>
```

In this method call, note the following:

- The <METADATA> element specifies the metadata type, the object instance, and the attributes that you want to modify. The first part of the two-part object identifier in the object definition identifies the repository in which to execute the request. *Attribute1*, *Attribute2*, and *Attribute3* are metadata type attributes. The new values specified for these attributes replace any existing values in the repository. Unmodified attributes remain unchanged.

- The OMI_TRUSTED_CLIENT (268435456) flag enables the SAS Metadata Server to write to the metadata object.

Using UpdateMetadata to Add or Modify Associations

To add or modify an association for an object, include an association name subelement and associated object definition in the metadata property string that describes the target object. In the association name subelement, specify an appropriate Function attribute value. In the associated object definition, specify an appropriate associated object identifier and value. For example:

```
<Metadata>
  <MetadataType Id="reposid.objectid">
     <AssociationName Function="directive">
       <AssociatedMetadataType Id
|ObjRef="value"/>
     </AssociationName>
  </MetadataType>
</Metadata>
```

The <METADATA> parameter identifies the metadata type, object instance, association name, and associated object affected. In this metadata property string, note the following:

- The association name subelement specifies a Function attribute and a directive. The Function attribute specifies how the SAS Metadata Server should process the associated object definition.
- The associated object subelement supports a choice of associated object identifiers. The associated object identifier that you choose to specify indicates whether a new association will be made to an existing object, whether a new association and a new associated object will be created, or whether an existing associated object will be modified.

Understanding the Function Attribute

Directives Supported by the Function Attribute

The Function attribute specifies how the SAS Metadata Server should process the associated object definition submitted with the association name. The supported values, referred to as directives, specify the order in which the specified associated object is ordered in the target object's association list, when multiple associations are supported. Support for multiple associations is denoted by a 0:* cardinality figure for the association name in the metadata type documentation. Association names that support a single association have a 0:1 or 1:1 cardinality figure defined for them in the metadata type documentation.

The Function attribute supports the directives shown in the following table. If the Function attribute is omitted from an UpdateMetadata request, MODIFY is the default directive for a single association, and MERGE is the default directive for a multiple association.

Table 12.1 *Function Directives Supported by the UpdateMetadata Method*

Directive	Supported for Single Associations?	Supported for Multiple Associations?	Description
Append	No	Yes	Adds the specified associations to the specified object's association list without modifying any of the other associations.
Merge	Yes	Yes	Adds or modifies associations in the specified object's association list.
Modify	Yes	No	Modifies an existing association or adds the association if the association does not exist.
Replace	Yes	Yes	Single: Overwrites an existing association with the specified association. Multiple: Replaces the existing association list with the specified association list, listing any new associations first. Any existing associations that are not represented in the new association list are deleted.
Remove	Yes	Yes	Deletes the specified associations from the specified object's association list without modifying any of the other associations.

Examples of How the Function Directives Affect Association Ordering

Assume that a Job object exists that has an Extensions association to several Extension objects. The association list contains the following associations:

```
<Job Id="A5RPBASE.AZ000001" Name="Test01" PublicType="Job" UsageVersion="">
  <Extensions>
    <Extension Id="A5RPBASE.AJ0000SD" Name="Extension1" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SE" Name="Extension2" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SF" Name="Extension3" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SG" Name="Extension4" Desc=""/>
  </Extensions>
</Job>
```

Submit the following UpdateMetadata method call with different Function= directives to illustrate the effect of each directive on the association list:

```
<UpdateMetadata>
   <Metadata>
     <Job Id="A5RPBASE.AZ000001" PublicType="Job" UsageVersion="">
       <Extensions Function="directive">
         <Extension Id="A5RPBASE.AJ0000SD" Name="Extension1" Desc="directive"/>
         <Extension Id="" Name="Extension5"/>
       </Extensions>
    </Job>
   </Metadata>
   <Reposid>A0000001.A5RPBASE</Reposid>
   <NS>SAS</NS>
   <--- OMI_TRUSTED_CLIENT flag --->
   <Flags>268435456</Flags>
  </UpdateMetadata>
```

The UpdateMetadata method call specifies to modify Extension1 to include the name of the directive in the Desc= attribute. It adds an association to an Extension5 object. The blank Id value in the Extension5 XML element indicates to create the associated object. For the purpose of this example, the association list is reset back to the four original associations in between each UpdateMetadata request.

Results of the UpdateMetadata method call with Function="APPEND":

```
<Extensions>
    <Extension Id="A5RPBASE.AJ0000SD" Name="Extension1" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SE" Name="Extension2" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SF" Name="Extension3" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SG" Name="Extension4" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SH" Name="Extension1" Desc="Append"/>
    <Extension Id="A5RPBASE.AJ0000SI" Name="Extension5" Desc=""/>
  </Extensions>
```

The APPEND directive never modifies existing associations. Any associations that you attempt to modify are treated as new associations. The existing object, Extension1, is duplicated with a new Id value and given the specified Desc= value. The new associations are added to the end of the existing association list.

Results of the UpdateMetadata method call with Function="MERGE":

```
<Extensions>
    <Extension Id="A5RPBASE.AJ0000SE" Name="Extension2" Desc=""/>
```

```
    <Extension Id="A5RPBASE.AJ0000SF" Name="Extension3" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SG" Name="Extension4" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SJ" Name="Extension5" Desc=""/>
    <Extension Id="A5RPBASE.AJ0000SD" Name="Extension1" Desc="Merge"/>
  </Extensions>
```

MERGE modifies and moves existing associations instead of duplicating them. Modified associations are added to the association list after new associations in the order in which they are specified.

Function="MODIFY" has no effect on the association list. MODIFY is not supported for multiple associations.

Results of the UpdateMetadata method call with Function="REPLACE":

```
<Extensions>
   <Extension Id="A5RPBASE.AJ0000SK" Name="Extension5" Desc=""/>
   <Extension Id="A5RPBASE.AJ0000SD" Name="Extension1" Desc="Replace"/>
</Extensions>
```

With REPLACE, the previous association list is deleted and replaced with a new association list. New associations are added to the association list before modified associations.

When the UpdateMetadata method is executed with Function="REMOVE", the method call fails with the error:

```
Subelement Extension :  cannot be removed from Job : A5RPBASE.AZ000001.
```

The method call fails because Extension5 cannot be found. To prevent a removal request from failing when an association cannot be found, you can set the OMI_IGNORE_NOTFOUND (134217728) flag with the OMI_TRUSTED_CLIENT (268435456) flag in the method call.

Results of the UpdateMetadata method call with Function="REMOVE" and the OMI_IGNORE_NOTFOUND flag:

```
<Extensions>
   <Extension Id="A5RPBASE.AJ0000SM" Name="Extension2" Desc=""/>
   <Extension Id="A5RPBASE.AJ0000SN" Name="Extension3" Desc=""/>
   <Extension Id="A5RPBASE.AJ0000SO" Name="Extension4" Desc=""/>
</Extensions>
```

The directive removes Extension1 from the association list.

Summary of Function Directives

- To add associations to an existing association list without modifying the properties of other associations in the association list, specify Function="APPEND".
- To add new associations and modify the properties of an existing association in the association list, specify Function="MERGE".
- To replace an existing association list with a new association list, specify Function="REPLACE".
- To remove an association from an association list without affecting any of the other associations in the list, specify Function="REMOVE".
- To modify the properties of the associated object in a single association, specify Function="MODIFY".

Understanding the Associated Object Identifiers

Identifiers and Supported Values

The following table lists the identifiers and values that are supported in an associated object definition and their behaviors when used in the UpdateMetadata method.

Table 12.2 *Identifiers and Values Supported in an Associated Object Definition*

Identifier and Value	Result
Id="" Id="*Reposid.$SymbolicName*" or no identifier	Create an association and the associated object. For more information about symbolic names, see "Symbolic Names" on page 280. When a symbolic name is used to create a new object in UpdateMetadata, the form Id="*Reposid.$SymbolicName*" must be used whether the object is in the same repository or in a different repository than the top-level object.
Id="*real_value*"	Modifies the specified object with the specified properties, if the object is found. If the object is not found, the update fails.
ObjRef="*real_value*"	Creates an association to, but does not modify the properties of the specified object, if the object is found. If the object is not found, the update fails.

Usage Examples

The following is an example of an association name and an associated object definition that adds an association to an existing object in the same repository:

```
<AssociationName Function="Directive">
  <AssociatedMetadataType ObjRef="Objectid"/>
</AssociationName>
```

Note the use of the ObjRef attribute and an object identifier in the associated object definition. If you specify additional attributes, the method ignores them.

Here is an example of an association name and an associated object definition that adds an association and a new object in the same repository:

```
<AssociationName Function="Directive">
  <AssociatedMetadataType Id="" Name="Name"/>
</AssociationName>
```

Note the use of the Id attribute with a null value in the associated object definition. Another way to create the associated object is to omit an object identifier from the associated object definition.

Here is an example of an association name and an associated object definition that modifies an existing associated object in the same repository:

```
<AssociationName Function="Directive">
  <AssociatedMetadataType Id="Objectid" Name="Name"
    Desc="This is a new description for this associated object."/>
</AssociationName>
```

Note the use of the Id attribute with a real object identifier in the associated object definition.

To create an association to an existing object in another repository using the UpdateMetadata method, specify the ObjRef attribute and include the object's repository identifier in the object instance identifier of the associated object definition. For example:

```
<AssociationName Function="Directive">
  <AssociatedMetadataType ObjRef="Reposid.Objectid"/>
</AssociationName>
```

To create an association and a new object in another repository, specify the Id attribute, a repository identifier, and a symbolic name in the object instance identifier of the associated object definition. For example:

```
<AssociationName Function="Directive">
  <AssociatedMetadataType Id="Reposid.$SymbolicName" Name="Name"/>
</AssociationName>
```

Deleting Associations

Deleting an association with the REMOVE directive in the Function attribute does not delete the associated object, unless the associated object is a dependent object. To delete an associated object, you must delete each associated object individually using the DeleteMetadata method. For more information, see Chapter 19, "Deleting Metadata Objects," on page 395.

The UpdateMetadata method does not print dependent objects that it might have deleted in its output by default. To include the dependent objects deleted by an Update operation in the output, set the OMI_RETURN_LIST (1024) flag in the UpdateMetadata method call.

For an example of deleting an association, see "Example of an UpdateMetadata Request That Deletes an Association" on page 304.

Example of an UpdateMetadata Request That Modifies an Object's Attributes

The following is an example of an UpdateMetadata request that modifies an object's attributes. The specified attributes and values replace attribute values stored for the object of the specified metadata type and object instance identifier. Examples in this section are formatted for submission in the INMETADATA parameter of the DoRequest method.

```
<UpdateMetadata>
   <Metadata>
      <SASLibrary
        Id="A53TPPVI.A1000001"
        Engine="oracle"
        IsDBMSLibname="1"/>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TRUSTED_CLIENT flag -->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

In the request, note the following:

- The Id attribute is used in the main element and specifies a real value. If the object is not found, the request fails.
- The request submits new values for SASLibrary object A53TPPVI.A1000001's Engine and IsDBMSLibname attributes. Unmodified attributes remain unchanged.

Example of an UpdateMetadata Request That Modifies an Association

The following is an example of an UpdateMetadata request that adds a single association (one that has a 0:1 or 1:1 cardinality in the metadata type documentation). PhysicalTable object A53TPPVI.A4000001 is updated to add a PrimaryPropertyGroup association to a PropertyGroup object. A PhysicalTable object has a 0:1 cardinality to a PropertyGroup object in a PrimaryPropertyGroup association.

```
<UpdateMetadata>
   <Metadata>
      <PhysicalTable Id="A53TPPVI.A4000001">
         <PrimaryPropertyGroup Function="Modify">
            <PropertyGroup Id="" Name="Read Options"/>
         </PrimaryPropertyGroup>
      </PhysicalTable>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TRUSTED_CLIENT Flag -->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

In the request, note the following:

- The Id attribute is used in the main element and specifies a real value.
- The association name element (PrimaryPropertyGroup) specifies the Function attribute with the MODIFY directive, which is required to modify single associations.

- Use of the Id attribute in the associated object definition with a null value instructs the SAS Metadata Server to create the PrimaryPropertyGroup association and the required PropertyGroup object if they do not exist, and to modify the properties of the PropertyGroup object if it does exist.

To replace an existing PrimaryPropertyGroup association with a new association, you need to specify Function="REPLACE".

Example of an UpdateMetadata Request That Merges Associations

The following UpdateMetadata examples illustrate the use of the MERGE directive. MERGE is the default directive for multiple associations when the Function attribute is omitted from an UpdateMetadata request. MERGE adds and modifies associations without overwriting existing associations.

The first example adds UniqueKeys and ForeignKeys associations to the table objects created in Chapter 11, “Adding Metadata Objects,” on page 277. The UpdateMetadata request consists of two main parts:

- It adds a UniqueKeys association to PhysicalTable A53TPPVI.A4000002 and identifies the table’s Name column (A53TPPVI.A5000005) as the key column.
- It associates the UniqueKey object with PhysicalTable A53TPPVI.A4000001 by creating a ForeignKeys association. The ForeignKey object identifies the table’s Employees column (A53TPPVI.A5000004) as its key column.

```
<UpdateMetadata>
   <Metadata>
      <PhysicalTable Id="A53TPPVI.A4000002">
         <UniqueKeys Function="Merge">
            <UniqueKey Id="" Name="Sales Associates in
NW Region">
              <KeyedColumns Function="Merge">
                <Column ObjRef="A53TPPVI.A5000005"/>
              </KeyedColumns>
              <ForeignKeys Function="Merge">
                <ForeignKey Id="" Name="Link to Sales
Associates table">
                  <Table>
                    <PhysicalTable ObjRef="A53TPPVI.A4000001"
                             Name="Sales offices in NW Region"/>
                  </Table>
                  <KeyedColumns Function="Merge">
                    <Column
ObjRef="A53TPPVI.A5000004"/>
                  </KeyedColumns>
                </ForeignKey>
              </ForeignKeys>
            </UniqueKey>
         </UniqueKeys>
      </PhysicalTable>
   </Metadata>
   <NS>SAS</NS>
```

```
   <!-- OMI_TRUSTED_CLIENT Flag -->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

In the request, note the following:

- The Id attribute in the main element specifies a real value.
- The Function directives in the <UNIQUEKEYS>, <KEYEDCOLUMNS>, and <FOREIGNKEYS> association elements specify to merge the new associations with any existing associations defined in the association lists of the specified tables and columns. MERGE is the default directive for multiple associations, so the directives could have been omitted from the request, and MERGE would be used.
- The null Id values in the <UNIQUEKEY> and <FOREIGNKEY> subelements instruct the SAS Metadata Server to create new associated objects.
- The ObjRef attribute in the <COLUMN> element specifies to create an association to an existing object.
- The Table association is a single association. The default directive for single associations is MODIFY, which modifies an existing association or adds it if the association does not exist. Use of the ObjRef attribute prevents the table's other attributes from being modified.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the UPDATEMETADATA method. -->

<PhysicalTable Id="A53TPPVI.A4000002">
<UniqueKeys>
<UniqueKey Id="A53TPPVI.A8000001" Name="Sales
Associates in NW Region">
<KeyedColumns>
<Column ObjRef="A53TPPVI.A5000005"/>
</KeyedColumns>
<ForeignKeys>
<ForeignKey Id="A53TPPVI.A9000001" Name="Link to Sales
Associates table">
<Table>
<PhysicalTable ObjRef="A53TPPVI.A4000001"/>
</Table>
<KeyedColumns>
<Column ObjRef="A53TPPVI.A5000004"/>
</KeyedColumns>
<PartnerUniqueKey>
<UniqueKey ObjRef="A53TPPVI.A8000001"/>
</PartnerUniqueKey>
</ForeignKey>
</ForeignKeys>
<Table>
<PhysicalTable ObjRef="A53TPPVI.A4000002"/>
</Table>
</UniqueKey>
</UniqueKeys>
</PhysicalTable>
```

The request created seven associations and two new objects (UniqueKey and ForeignKey).

The following example updates the UniqueKey object created in the previous request. It modifies the Name attribute of the key column and adds an association to a second key column.

```
<UpdateMetadata>
   <Metadata>
      <UniqueKey Id="A53TPPVI.A8000001">
         <KeyedColumns>
             <Column Id="A53TPPVI.A5000005"
Name="EmployeeName"/>
             <Column ObjRef="A53TPPVI.A5000006"/>
          </KeyedColumns>
      </UniqueKey>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TRUSTED_CLIENT Flag -->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

In the request, note the following:

- A Function directive is omitted from the request because KeyedColumns is a multiple association and the default value of MERGE is appropriate for the operation.
- Use of the Id attribute to identify the original keyed column allows the column's properties to be updated.
- Use of the ObjRef attribute to identify the newly associated column creates the association and does not modify any of the column's other attributes.

Here are the results of a GetMetadata request that lists the UniqueKey object's KeyedColumns associations:

```
<!-- Using the GETMETADATA method. -->

<UniqueKey Id="A53TPPVI.A8000001" Name="Sales Associates in NW Region">
<KeyedColumns>
<Column Id="A53TPPVI.A5000005" Name="EmployeeName"
Desc="Name of employee"/>
<Column Id="A53TPPVI.A5000006" Name="Address" Desc="Home Address"/>
</KeyedColumns>
</UniqueKey>
```

Two Column objects are returned. Column A53TPPVI.A5000005's Name attribute was changed from Name to EmployeeName.

Example of an UpdateMetadata Request That Deletes an Association

The following UpdateMetadata request deletes the KeyedColumns association added in the second example in "Example of an UpdateMetadata Request That Merges Associations" on page 302:

```
<UpdateMetadata>
```

```
  <Metadata>
     <UniqueKey Id="A53TPPVI.A8000001">
       <KeyedColumns Function="Remove">
           <Column Id="A53TPPVI.A5000006" Name="Address"/>
       </KeyedColumns>
     </UniqueKey>
  </Metadata>
  <NS>SAS</NS>
  <!-- OMI_TRUSTED_CLIENT Flag -->
  <Flags>268435456</Flags>
  <Options/>
</UpdateMetadata>
```

The Function="REMOVE" directive instructs the SAS Metadata Server to remove the specified association from the UniqueKey object's KeyedColumns association list. If Function="REPLACE" had been specified, the existing KeyedColumns association list would have been replaced with the specified association.

Here are the results of a GetMetadata call that gets a revised KeyedColumns associations list:

```
<!-- Using the GETMETADATA method. -->

<UniqueKey Id="A53TPPVI.A8000001" Name="Sales Associates in NW Region">
<KeyedColumns>
<Column Id="A53TPPVI.A5000005" Name="EmployeeName" Desc="Name of employee"/>
</KeyedColumns>
</UniqueKey>
```

For more information about the use of REMOVE versus REPLACE, see "Deleting Associations" on page 300.

Example of an UpdateMetadata Request That Appends Associations

The following UpdateMetadata request adds an association and a new Column object to PhysicalTable A53TPPVI.A4000002 using the APPEND directive:

```
<UpdateMetadata>
   <Metadata>
      <PhysicalTable Id="A53TPPVI.A4000002">
         <Columns Function="Append">
             <Column Id="" Name="Salary"
PublicType="Column"/>
         </Columns>
      </PhysicalTable>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TRUSTED_CLIENT Flag -->
   <Flags>268435456</Flags>
   <Options/>
</UpdateMetadata>
```

In the example, the null Id value in the associated object definition indicates the associated object is to be created. Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the UPDATEMETADATA method. -->

<PhysicalTable Id="A53TPPVI.A4000002">
<Columns Function="Append">
<Column Id="A53TPPVI.A500002U" Name="Salary">
<Table>
<PhysicalTable ObjRef="A53TPPVI.A4000002"/>
</Table>
</Column>
</Columns>
</PhysicalTable>
```

The association to the new Column object is added to the existing association list without affecting other associated objects. Here are the results of a GetMetadata request that lists the Column objects associated with PhysicalTable A53TPPVI.A4000002:

```
<!-- Using the GETMETADATA method. -->

<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates">
<Columns>
<Column Id="A53TPPVI.A5000005" Name="EmployeeName" Desc="Name of employee"/>
<Column Id="A53TPPVI.A5000006" Name="Address" Desc="Home Address"/>
<Column Id="A53TPPVI.A5000007" Name="Title" Desc="Job grade"/>
<Column Id="A53TPPVI.A500002U" Name="Salary" Desc=""/>
</Columns>
</PhysicalTable>
```

13

Overview of Querying Metadata

Supported Queries

Overview

The SAS Open Metadata Interface provides methods to get information about the SAS Metadata Server's availability and configuration, namespaces, metadata repositories, and metadata objects.

Querying Server Availability and Configuration

In SAS 9.4, the IServer server interface Status method has been enhanced to get more information about the SAS Metadata Server. Using the Status method, you can get the following information:

- the server's current state
- SAS Metadata Model and platform version numbers
- the server locale
- the value of specific server configuration options that are set in the omaconfig.xml file and server invocation command

- journaling statistics
- server performance statistics
- information about server backups
- information about the server's alert e-mail notification subsystem
- information about the configuration and availability of the servers in a clustered SAS Metadata Server configuration.

For more information, see "Status Method" on page 255.

Querying Namespaces

A namespace refers to the set of metadata types that can be accessed by the SAS Metadata Server. The SAS Open Metadata Interface defines two namespaces:

- The REPOS namespace contains metadata types that describe metadata repositories.
- The SAS namespace contains all metadata types that describe application elements.

SAS provides the GetNamespaces method to enable you to get the namespaces programmatically. For more information, see "GetNamespaces Method" on page 120. The GetTypes method gets all of the metadata types in a namespace. For more information, see "GetTypes Method" on page 132. The GetTypeProperties method gets all possible properties for a specified metadata type. For more information, see "GetTypeProperties Method" on page 130.

The SAS Metadata Model is a hierarchical model. The GetSubtypes method gets subtypes of a specified metadata type. For more information, see "GetSubtypes Method" on page 128. The IsSubtypeOf method determines whether on metadata type is a subtype of another metadata type.For more information, see "IsSubtypeOf Method" on page 137. All of these methods are referred to as management methods because they enable you to get information about the metadata environment. This information provides useful background information when preparing to create metadata objects.

See also:"Using GetTypes to Get the Metadata Types in a Namespace" on page 309.

Querying Repositories

When a repository is created, it is registered in a SAS Repository Manager. The SAS Repository Manager is itself a repository, which maintains information that enables the SAS Metadata Server to access the repositories, metadata programmers to create metadata in the repositories, and administrators to administer the availability of the repositories.

You can determine the repositories that have been registered in a SAS Repository Manager by using the GetRepositories method. For more information, see "GetRepositories Method" on page 122. The GetRepositories method lists the Id, Name, Desc, and DefaultNS attributes of the repositories registered in the SAS Repository Manager. A repository identifier is required to add metadata to a

repository. For usage information, see “Using GetRepositories to List the Registered Repositories” on page 311.

The SAS Repository Manager also stores Path, RepositoryType, RepositoryFormat, Access, PauseState, and CurrentAccess attributes for a repository. To get the values of these attributes, you can issue the GetRepositories method with the OMI_ALL (1) flag set. For usage information, see “Using GetRepositories to Get All Repository Attributes” on page 312.

A repository is described by a metadata type just like any other metadata object. To get values for global attributes that are stored for all metadata types, you can use the same methods that you use to read application objects. For more information, see “Querying Metadata Objects” below. Issue the GetMetadata and GetMetadataObjects method calls on the REPOS namespace and specify the RepositoryBase metadata type. For more information, see “Using GetMetadata to Get Only a Repository's Regular Attributes” on page 313.

Querying Metadata Objects

A metadata object consists of attributes and associations that uniquely describe the object instance. The object instance can be an application element or a repository. The SAS Open Metadata Interface provides two methods for reading metadata objects:

- The GetMetadata method gets specified properties of a specific metadata object.
- The GetMetadataObjects method gets all metadata objects of a specified metadata type from the specified repository.

The methods support flags and options that enable you to expand or to filter your requests.

For reference information about the methods, see “GetMetadata Method” on page 109 and “GetMetadataObjects Method” on page 115.

For usage information, see Chapter 14, “Getting the Properties of a Specified Metadata Object,” on page 315, Chapter 15, “Using Templates,” on page 337, Chapter 16, “Getting All Metadata of a Specified Metadata Type,” on page 355, and Chapter 17, “Filtering a GetMetadataObjects Request,” on page 371.

SAS uses an object type dictionary to ensure consistency among common and shared object definitions. For objects that are persisted in metadata, the SAS type dictionary standardizes the logical metadata definitions which represent the objects in a SAS Metadata Repository. See Chapter 3, “Using Interfaces That Read and Write Metadata in SAS 9.4,” on page 19 for information about how the SAS type dictionary affects GetMetadata and GetMetadataObjects requests.

Using GetTypes to Get the Metadata Types in a Namespace

The SAS Open Metadata Interface provides the GetTypes method to get all of the metadata types defined in a namespace. The following is an example of a GetTypes request that gets the metadata types that are defined in the SAS namespace of the

SAS Metadata Model. The request is formatted for the INMETADATA parameter of the DoRequest method:

```
<GetTypes>
   <Types/>
   <NS>SAS</NS>
   <Flags/>
   <Options/>
</GetTypes>
```

The <NS>, <FLAGS>, and <OPTIONS> elements are input parameters. This example does not specify any flags or options. The <TYPES> element is an output parameter. The <TYPES> element lists the metadata types in the specified namespace.

To get all of the metadata types in the SAS Metadata Model, issue the GetTypes method on both the SAS and REPOS namespaces.

The following is an example of the method output. Only the first line of the output is shown:

```
<Type Id="AbstractExtension" Desc="Abstract Extension"
HasSubtypes="1"/>
```

In the output, note the following:

- Id is a metadata type name.
- Desc is a system-supplied description of the metadata type.
- HasSubtypes is a Boolean indicator that identifies whether a metadata type has subtypes. A value of 1 indicates that the type has subtypes. A value of 0 indicates that it does not.

Using GetTypes to Get Actual Metadata Types in a Repository

After adding metadata objects, you can get all metadata types defined in a repository by using the GetTypes method with the OMI_SUCCINCT (2048) flag set. When used with OMI_SUCCINCT and its <REPOSID> element, the GetTypes method returns the metadata types for which metadata has been defined in a specific repository.

Here is an example of a GetTypes request that sets the OMI_SUCCINCT flag:

```
<GetTypes>
   <Types/>
   <NS>SAS</NS>
   <!-- specify the OMI_SUCCINCT flag -->
   <Flags>2048</Flags>
   <Options>
     <!-- include <REPOSID> XML element and a repository identifier -->
     <Reposid>A0000001.A53TPPVI</Reposid>
   </Options>
</GetTypes>
```

The <NS>, <FLAGS>, <OPTIONS>, and <REPOSID> elements are input parameters.

- The <NS> element specifies the namespace.
- The <FLAGS> element sets the OMI_SUCCINCT flag (2048).
- The <OPTIONS> element passes the <REPOSID> element to the SAS Metadata Server.
- The <REPOSID> element specifies the target repository identifier.

The <TYPES> element is an output parameter. Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETTYPES method. -->

<Types>
<Type Id="Column" Desc="Columns" HasSubtypes="0"/>
<Type Id="PhysicalTable" Desc="Physical Table" HasSubtypes="1"/>
<Type Id="ResponsibleParty" Desc="Responsible Party" HasSubtypes="0"/>
<Type Id="SASLibrary" Desc="SAS Library" HasSubtypes="0"/>
</Types>
```

The repository contains metadata objects of four metadata types: Column, PhysicalTable, ResponsibleParty, and SASLibrary.

- Id specifies the metadata type.
- Desc returns a system-supplied description of the metadata type.
- When OMI_SUCCINCT is set, the HasSubtypes attribute has no meaning.

To list the actual metadata objects of each metadata type, you must use the GetMetadataObjects method. For more information, see “GetMetadataObjects Method” on page 115.

Using GetRepositories to List the Registered Repositories

You can get the repositories that are registered on a SAS Metadata Server by issuing the GetRepositories method. For more information, see “GetRepositories Method” on page 122. The following is an example of a GetRepositories request that is formatted for the INMETADATA parameter of the DoRequest method.

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

The request gets the Id, Name, Desc and DefaultNS attributes of all repositories registered on the SAS Metadata Server. Here is an example of the output returned by the SAS Metadata Server, formatted for readability:

```
<GetRepositories>
<Repositories>
 <Repository Id="A0000001.A529QDG2" Name="Foundation" Desc="" DefaultNS="SAS"/>
```

```
 <Repository Id="A0000001.A5M41IFE" Name="BILineage" Desc="BILineage" DefaultNS="SAS"/
</Repositories>
<Flags>0</Flags>
<Options/>
</GetRepositories>
```

The current SAS Repository Manager has two repositories registered in it: Foundation and BILineage.

Using GetRepositories to Get All Repository Attributes

To get a complete list of a repository's attributes for all registered repositories, issue the GetRepositories method with the OMI_ALL (1) flag set. When OMI_ALL is set, GetRepositories returns values for each repository's regular metadata attributes (Engine, MetadataCreated, MetadataUpdated, and Options) in addition to general, identifying information and repository attributes (Access, CurrentAccess, Path, PauseState, RepositoryFormat and RepositoryType). See "GetRepositories Method" on page 122 for a description of the attributes.

The following is an example of a method call that sets this flag:

```
<GetRepositories>
 <Repositories/>
<!-- OMI_ALL flag  -->
 <Flags>1</Flags>
 <Options/>
</GetRepositories>
```

Here is an example of the output returned by the SAS Metadata Server, formatted for readability:

```
<GetRepositories>
<Repositories>
 <Repository Id="A0000001.A0000001" Name="REPOSMGR" Desc="The Repository Manager"
DefaultNS="REPOS" RepositoryType="" RepositoryFormat="15" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState="" Path="rposmgr" Engine="" Options=""
MetadataCreated="01Jan1960:00:00:00" MetadataUpdated="01Jan1960:00:00:00"/>
 <Repository Id="A0000001.A529QDG2" Name="Foundation" Desc="" DefaultNS="SAS"
RepositoryType="FOUNDATION" RepositoryFormat="15" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState="" Path="MetadataRepositories/Foundation"
Engine="BASE Options="" MetadataCreated="26Jul2012:20:12:12"
MetadataUpdated="26Jul2012:20:12:12"/>
 <Repository Id="A0000001.A5M41IFE" Name="BILineage" Desc="BILineage" DefaultNS="SAS"
RepositoryType="CUSTOM" RepositoryFormat="15" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState="" Path="MetadataRepositories/BILineage"
Engine="BASE" Options="" MetadataCreated="27Jul2012:00:05:28"
MetadataUpdated="27Jul2012:00:05:28"/>
</Repositories>
<Flags>1</Flags>
<Options/>
</GetRepositories>
```

In the output, note the following:

- There are two metadata repositories registered in addition to the SAS Repository Manager. The SAS Repository Manager (REPOSMGR) is the first repository listed.
- A blank value in the SAS Repository Manager's PauseState attribute indicates the SAS Metadata Server is online. The Pause method affects all repositories uniformly, so you can expect all of the repositories to be available, depending on their registered access values.
- The foundation repository is listed next and is registered with an Access value of OMS_FULL (full access). Following is the custom repository, named BILineage, which also has an access value of OMS_FULL.
- All of the repositories have the same format (15).
- The fact that the CurrentAccess value for each repository matches the Access value indicates that the SAS Metadata Server is able to access all repositories without a problem. The CurrentAccess value changes only if the SAS Metadata Server cannot access a repository in its intended state for a reason other than a server pause. For example, if the repository's format is not compatible with the current SAS Metadata Server. In that case, CurrentAccess might return a value of OMS_READONLY or OMS_OFFLINE.

Using GetMetadata to Get Only a Repository's Regular Attributes

To get only general information about a repository, such as its Id, name, description, engine, repository format, repository type, and metadata created and updated dates, use the GetMetadata method and set the OMI_ALL (1) flag.

The following GetMetadata method call, which is formatted for the INMETADATA parameter of the DoRequest method, requests information about the Foundation repository:

```
<GetMetadata>
   <Metadata>
         <RepositoryBase Id="A0000001.A55WR3E8" Name="Foundation" />
   </Metadata>
  <NS>REPOS</NS>
  <Flags>1</Flags>
   <Options/>
</GetMetadata>
```

In the method call, note the following:

- the <METADATA> element specifies the metadata type RepositoryBase, not Repository.
- the <NS> element specifies the REPOS namespace.
- the <FLAGS> element specifies a 1, setting the OMI_ALL flag. If this flag were not set, the GetMetadata method would return only the Id and Name attributes of the repository, which we already know.

Here is an example of the output returned from the request:

```
<GetMetadata>
 <Metadata>
 <RepositoryBase Id="A0000001.A529QDG2" Name="Foundation" Access="0" Desc=""
Engine="BASE" MetadataCreated="26Jul2012:20:12:12" MetadataUpdated="26Jul2012:20:12:12"
Options="" Path="MetadataRepositories/Foundation" RepositoryFormat="14"
RepositoryType="FOUNDATION">
  <DependencyUsedBy/>
  <DependencyUses/>
 </RepositoryBase>
</Metadata>
<NS>REPOS</NS>
<Flags>1</Flags>
<Options/></GetMetadata>
```

A GetMetadataObjects method that is issued in the REPOS namespace on the RepositoryBase metadata type with the OMI_GET_METADATA (256) flag and OMI_ALL (1) flag set will get this same information for all registered repositories.

14

Getting the Properties of a Specified Metadata Object

Introduction to the GetMetadata Method

Overview

To get the attributes and associations for a metadata object, the SAS Open Metadata Interface provides the GetMetadata method. The default behavior of the GetMetadata method is to get the specified metadata object with whatever attributes and associations are specified in the INMETADATA parameter.

The GetMetadata method also supports flags. Flags are provided that:

- get all of the attributes and associations that are documented for the specified metadata type.
- get all of the attributes of the specified metadata object and any associated objects that are returned by the GetMetadata request.
- get specified attributes and associations for subtypes of the specified object.
- get the logical metadata definition for the specified metadata object as defined in the SAS type dictionary. The specified object must be a PublicType subtype, and it must store the name of a type definition in the PublicType attribute.
- enable you to submit templates which specify attributes and associations to get for associated objects.

GetMetadata and Cross-Repository References

When you submit the GetMetadata method, you identify the object to get by specifying its metadata type and 17-character metadata identifier. The first eight characters of the 17-character identifier represent a repository identifier. A GetMetadata method call that requests associated objects will get all associated objects that are in the same repository. In addition, it will get cross-repository references to objects that are in repositories of a compatible type.

The SAS Metadata Server supports three types of repositories. Two of the repository types (the foundation and custom repositories) are considered public repositories; they hold metadata that is available for production use. The third type, project repositories, are private; they contain copies of objects for making changes that might or might not be promoted for production use.

- A GetMetadata request that is issued on an object in a public repository returns associated objects that are in other public repositories by default. GetMetadata will not retrieve cross-repository references to objects that are in project repositories unless you specify a flag. For more information, see “Including Objects from Project Repositories in a Public Query” on page 333.
- A GetMetadata request that is issued on an object in a project repository returns associated objects that are in the project repository. In addition, it returns cross-repository references from all of the public repositories that the project repository services.

Cross-repository references involving project repositories are managed by a change management facility. The change management facility is used exclusively by SAS Data Integration Studio.

GetMetadata and Logical Type Definitions

The GetMetadata method treats all metadata objects as independent objects. That is, it gets the specified metadata object and specified attributes and associations of the specified object. In the initial releases of the software, the only way to get attributes and associations for associated objects in a request was to set the OMI_TEMPLATE (4) flag, and submit user-defined templates that specified the association names and secondary metadata types to expand.

SAS 9.3 introduced the OMI_FULL_OBJECT (2) flag. If the specified object is a PublicType subtype in the SAS Metadata Model, and if it stores a valid value in the PublicType attribute, the OMI_FULL_OBJECT flag gets all direct and nested associations in that object type's logical metadata definition as defined in the SAS type dictionary.For more information about the SAS type dictionary, see Chapter 3, "Using Interfaces That Read and Write Metadata in SAS 9.4," on page 19. For more information about the OMI_FULL_OBJECT flag, see "Using the OMI_FULL_OBJECT Flag" on page 334.

Basic GetMetadata Request

The following is an example of a basic GetMetadata request. The request is formatted for the DoRequest interface:

```
<GetMetadata>
 <Metadata>
   <Column Id="A5TJRDIT.B700002E" Name="" Desc="" SASColumnType="" IsNullable="">
    <Table/>
   </Column>
 </Metadata>
 <NS>SAS</NS>
 <Flags>0</Flags>
 <Options/>
</GetMetadata>
```

In the request, note the following:

- The property string in the <METADATA> element specifies to get a Column object with the metadata identifier A53TPPVI.A5000001. The A53TPPVI portion of the identifier indicates the repository to look in. A5000001 is the unique object instance identifier.
- Name, Desc, SASColumnType, and IsNullable in the property string are XML attributes of the Column metadata type for which we are requesting that the GetMetadata method return values.
- The <Table/> subelement within the <Column> property string requests that GetMetadata return objects that are associated with the specified Column object via the Table association name. A Column object can have one table object associated with it. For a list of the table types supported under the Table association name, as well as other association names defined for the Column metadata type, see Column in the "Alphabetical Listing of SAS Namespace Metadata Types" in the *SAS Metadata Model: Reference*.

Here is an example of the output returned by the SAS Metadata Server:

```
<Column Id="A5TJRDIT.B700002E" Name="IDNUM" Desc="Identification Number"
SASColumnType="N" IsNullable="1">
    <Table>
        <PhysicalTable Id="A5TJRDIT.B200000J" Name="EMPINFO" Desc=""/>
</Table>
</Column>
```

The SAS Metadata Server returns values for the requested attributes of the specified Column object, and general, identifying information (Id, Name, and Desc)

about the associated PhysicalTable object. See the model documentation for a description of the attribute values.

To get additional properties for the specified Column object and its associated objects, the GetMetadata method supports the following flags:

- OMI_ALL_SIMPLE (8)—Gets all of the XML attributes of the specified object and of any associated objects that are returned. For more information, see "Expanding a GetMetadata Request to Get All Attributes" on page 318.
- OMI_ALL (1)—Gets all of the attributes and associations that are documented for the specified metadata type in the SAS Metadata Model. For any associated objects that are found, it returns general, identifying information. For more information, see "Expanding a GetMetadata Request to Get All Attributes and Associations" on page 319.
- OMI_SUCCINCT (2048)—Omits attributes that do not contain a value or that contain a null value from the returned XML string.
- OMI_TEMPLATE (4)—Instructs the SAS Metadata Server to check the OPTIONS parameter for one or more user-defined templates that specify additional attributes or associations to return. The templates can request additional attributes and associations for the primary metadata type in the INMETADATA parameter. Templates can also be used to return attributes and associations for associated objects in the INMETADATA parameter or in another template. Templates are specified in a <TEMPLATES> element. For more information, see "Getting Attributes and Associations of Associated Objects" on page 321 and Chapter 15, "Using Templates," on page 337.
- OMI_FULL_OBJECT (2)—Specifies to expand the associations for the specified object based on the type definition for that object type in the SAS type dictionary. The request is valid only if the specified object is a PrimaryType subtype in the SAS Metadata Model, and if it stores a valid value in the PublicType attribute. For more information, see "Using the OMI_FULL_OBJECT Flag" on page 334.

Expanding a GetMetadata Request to Get All Attributes

To get all of XML attributes of the requested objects, set the OMI_ALL_SIMPLE (8) flag in the GetMetadata request. The OMI_ALL_SIMPLE flag gets only an object's attributes; it does not get any associations. However, it will get the attributes for the specified object and all associated objects requested in the INMETADATA parameter and by other GetMetadata flags. The following is an example of how the OMI_ALL_SIMPLE flag is specified:

```
<GetMetadata>
 <Metadata>
   <Column Id="A5TJRDIT.B700002E">
     <Table/>
   </Column>
 </Metadata>
 <NS>SAS</NS>
 <!--OMI_ALL_SIMPLE flag -->
 <Flags>8</Flags>
```

```
 <Options/>
</GetMetadata>
```

Note the similarity of this request to example shown in "Basic GetMetadata Request" on page 317.

- The <METADATA> element specifies a metadata type (Column), an object instance identifier, and an association name (Table) to expand. The association name is optional; it is not a required part of the request.
- The <NS> parameter specifies the SAS namespace.
- The <FLAGS> element specifies the OMI_ALL_SIMPLE (8) flag.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATA method. -->

<Column Id="A5TJRDIT.B700002E" Name="IDNUM" Desc="Identification Number"
SASColumnType="N" IsNullable="1" BeginPosition="0" ChangeState="" ColumnLength="8"
ColumnName="IDNUM" ColumnType="0" EndPosition="0" IsDiscrete="1" IsHidden="0"
LockedBy="" MetadataCreated="10Jan2011:21:20:36" MetadataUpdated="14Jan2011:22:37:17"
PublicType="Column" SASAttribute="" SASColumnLength="8" SASColumnName="IDNUM"
SASExtendedColumnType="" SASExtendedLength="0" SASFormat="SSN11." SASInformat="F11."
SASPrecision="0" SASScale="0" SortOrder="" SummaryRole="" UsageVersion="1000000">
    <Table>
<PhysicalTable Id="A5TJRDIT.B200000J" Name="EMPINFO" Desc="" ChangeState=""
DBMSType="" IsCompressed="0" IsDBMSView="0" IsEncrypted="0" IsHidden="0"
LockedBy="" MemberType="DATA" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="14Jan2011:22:37:17" NumRows="-1" PublicType="Table"
SASTableName="EMPINFO" TableName="EMPINFO" UsageVersion="1000000"/>
</Table>
</Column>
```

The GetMetadata method gets all of the attributes of the specified Column object, including the names of attributes for which values have not been defined. It also gets all of the attributes of the PhysicalTable object that is returned through the Table association name.

To limit the output to attributes that have values defined, you can also set the OMI_SUCCINCT (2048) flag. Add the value of OMI_SUCCINCT to OMI_ALL_SIMPLE (2048 + 8 = 2056) and specify the sum in the FLAGS parameter. The OMI_SUCCINCT flag instructs the SAS Metadata Server to omit any attributes that do not contain a value or that contain a null value from the output.

Expanding a GetMetadata Request to Get All Attributes and Associations

To get all of a metadata object's attributes and associations, set the OMI_ALL (1) flag in the GetMetadata request. The flag gets all attributes and direct associations of the specified metadata object. It does not get associations of associated objects. The following is an example of a GetMetadata request that sets the OMI_ALL flag:

```
<GetMetadata>
 <Metadata>
   <Column Id="A53TPPVI.A5000001"/>
```

```
 </Metadata>
 <NS>SAS</NS>
 <!--OMI_ALL flag -->
 <Flags>1</Flags>
 <Options/>
</GetMetadata>
```

In the request, note the following:

- The <METADATA> element specifies a metadata type and an object instance identifier.
- The <NS> parameter specifies the SAS namespace.
- The <FLAGS> element specifies the OMI_ALL (1) flag.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATA Method -->
<Column Id="A5TJRDIT.B700002E" BeginPosition="0" ChangeState="" ColumnLength="8"
ColumnName="IDNUM" ColumnType="0" Desc="Identification Number" EndPosition="0"
IsDiscrete="1" IsHidden="0" IsNullable="1" LockedBy=""
MetadataCreated="10Jan2011:21:20:36" MetadataUpdated="14Jan2011:22:37:17" Name="IDNUM"
PublicType="Column" SASAttribute="" SASColumnLength="8" SASColumnName="IDNUM"
SASColumnType="N" SASExtendedColumnType="" SASExtendedLength="0" SASFormat="SSN11."
SASInformat="F11." SASPrecision="0" SASScale="0" SortOrder="" SummaryRole=""
UsageVersion="1000000">
    <AccessControls/>
    <AnalyticColumns/>
    <Changes/>
    <CustomAssociations/>
    <DisplayForKeys/>
    <Documents/>
    <Extensions/>
    <ExternalIdentities/>
    <FavoritesContainers/>
    <ForeignKeyAssociations/>
    <Groups/>
    <Implementors/>
    <Indexes>
        <Index Id="A5TJRDIT.BI000003" Name="IDNUM" Desc=""/>
</Indexes>
    <Keys>
        <UniqueKey Id="A5TJRDIT.BH000003" Name="EMPINFO.ic_id" Desc=""/>
</Keys>
    <Keywords/>
    <LocalizedAttributes/>
    <Notes/>
    <PrimaryPropertyGroup/>
    <Prompts/>
    <Properties/>
    <PropertySets/>
    <QueryClauses/>
    <ReferencedObjects/>
    <ResponsibleParties/>
    <SourceFeatureMaps/>
    <SourceTransformations/>
    <SpecSourceTransformations/>
    <SpecTargetTransformations/>
```

```
        <Table>
            <PhysicalTable Id="A5TJRDIT.B200000J" Name="EMPINFO" Desc=""/>
    </Table>
        <TargetFeatureMaps/>
        <TargetTransformations/>
        <Timestamps/>
        <Trees/>
        <TSObjectNamespace/>
        <UniqueKeyAssociations/>
        <UsedByPrototypes/>
        <UsingPrototype/>
        <Variables/>
        <XPaths/></Column>
```

The output includes all attributes and associations that are documented for the Column metadata type, including attributes and associations for which values have not been defined. To limit the output to attributes and associations that have values, also set the OMI_SUCCINCT (2048) flag. The OMI_SUCCINCT flag instructs the SAS Metadata Server to omit any attributes and associations that do not contain a value or that contain a null value from the output.

This Column object has three associated objects defined: an Index object is associated to the Column through the Indexes association name; a UniqueKey object is associated with the Column through the Keys association name; and a PhysicalTable object is associated to the Column through the Table association name. The OMI_ALL flag returns the Id, Name, and Desc values for associated objects.

Getting Attributes and Associations of Associated Objects

By default, the GetMetadata method gets only the Id, Name and Desc attributes of any associated objects that are returned by a request. If you set the OMI_ALL_SIMPLE (8) flag, you can get all of the XML attributes of the specified object and any associated objects. However, if you want to get specific attributes of associated objects, or associations of associated objects, you must set the OMI_TEMPLATE (4) flag and specify a template in the GetMetadata request.

A template is one or more additional property strings that you specify in the OPTIONS parameter of the GetMetadata method within a <TEMPLATES> element. The property strings specify additional attributes and associations to retrieve for the primary object in the INMETADATA parameter, an associated object that is requested in the INMETADATA parameter, or an associated object that is requested in another template. The attributes that are requested in the template are retrieved in addition to any attributes that are requested in the INMETADATA parameter or that are requested by other GetMetadata flags. For information about how to create a template, see “Creating Templates for the Get Methods” on page 340.

The following is an example of a GetMetadata request that sets the OMI_TEMPLATE flag and specifies a template. Templates request additional attributes for both the specified object and associated objects requested in the INMETADATA parameter (<METADATA> element):

```
<GetMetadata>
```

```
<Metadata>
  <Column Id="A53TPPVI.A5000001" Name="" Desc="" ColumnType="" SASFormat="">
   <Table/>
  </Column>
</Metadata>
<NS>SAS</NS>
<!-- OMI_TEMPLATE -->
<Flags>4</Flags>
<Options>
 <Templates>
   <Column Id="" ColumnLength="" BeginPosition="" EndPosition=""/>
   <PhysicalTable Id="" Name="" Desc="" DBMSType="" MemberType=""/>
 </Templates>
</Options>
</GetMetadata>
```

In the request, note the following:

- The <METADATA> element specifies the metadata type, an object instance identifier, four metadata type attributes, and the association name Table.
- The <NS> element specifies the SAS namespace.
- The <FLAGS> element specifies the number representing the OMI_TEMPLATE flag (4).
- The <OPTIONS> element contains a <TEMPLATES> element and two property strings. Each property string is a template. The first template specifies additional attributes to retrieve for the Column object identified in the <METADATA> element. The second template specifies attributes to retrieve for the PhysicalTable object that is associated with the Column object through the Table association name that was requested in the <METADATA> element.

Here is an example of the output that is returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATA method. -->

<Column Id="A53TPPVI.A5000001" Name="City" Desc="City of Sales Office"
  ColumnType="12" SASFormat="$Char32." ColumnLength="32" BeginPosition="0"
  EndPosition="0">
<Table>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"
  Desc="Sales offices in NW region" DBMSType="" MemberType=""/>
</Table>
</Column>
```

The GetMetadata method gets the values of the Name, Desc, ColumnType, SASFormat, ColumnLength, BeginPosition, and EndPosition attributes of the specified Column object. In addition, it gets the Id, Name, Desc, DBMSType, and MemberType attributes of the Column object's associated PhysicalTable object.

Filtering the Associated Objects That Are Returned by a GetMetadata Request

Introduction to the Search Attribute

The GetMetadata method supports a Search attribute on association names to filter the associated objects that it returns. The Search attribute can be specified in both the property string in the <METADATA> element and in a template in the <TEMPLATES> element. The Search attribute enables you to retrieve only associated objects that are of a specified metadata type, or are of a specified metadata type and meet specified criteria:

The Search attribute is specified in the association name element of a metadata property string in one of the following forms:

```
<AssociationName search="Object"/>

<AssociationName search="Object[Criteria]"/>
```

- *Object* can be an * (asterisk) or a SAS Metadata Model metadata type that is valid for the specified association name.

  Specifying an * instructs the SAS Metadata Server to get objects of all metadata types that are valid for *AssociationName*.

  Specifying a metadata type instructs the SAS Metadata Server to get only associated objects of the specified metadata type.

- [*Criteria*] is a search specification that conforms to the syntax documented for the <XMLSELECT> option. For information about the syntax, see "<XMLSELECT search="criteria"/> Syntax" on page 373. When search criteria are specified, GetMetadata retrieves only associated objects indicated by *Object* that also meet the specified criteria.

Example of Specifying Search Criteria in the <METADATA> Element

When specified in the <METADATA> element, the search criteria string looks like one of the following:

```
<Metadata>
  <MetadataType>
    <AssociationName search="Object"/>
  </MetadataType>
</Metadata>

<Metadata>
  <MetadataType>
```

```
    <AssociationName search="Object[AttributeCriteria]"/>
  </MetadataType>
</Metadata>
```

To understand the filtering that occurs, consider the following requests. In the first request, the <METADATA> element specifies to get the Document metadata object that has Id="A52WE4LI.AT0000RZ" and all objects that are associated with it through the Objects association name (all metadata types):

```
<GetMetadata>
   <Metadata>
     <Document Id="A52WE4LI.AT0000RZ">
      <Objects search="*"/>
     </Document>
   </Metadata>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadata>
```

The request is the same as specifying the association name without search criteria:

```
<GetMetadata>
   <Metadata>
     <Document Id="A52WE4LI.AT0000RZ">
      <Objects/>
     </Document>
   </Metadata>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadata>
```

In both requests, the GetMetadata method specifies to get requested attributes for the specified Document object (Id only in this case) and all objects that are associated with it through the Objects association name. Here is an example of the output from the requests:

```
<!-- Using the GETMETADATA method. -->

<Document Id="A52WE4LI.AT0000RZ">
  <Objects>
   <PhysicalTable Id="A52WE4LI.B60000RT" Name="Table1" Desc="Sales table"/>
   <PhysicalTable Id="A52WE4LI.B60000RU" Name="Table2" Desc="Human Resources table"/>
   <ExternalTable Id="A52WE4LI.BA000001" Name="Oracle Sales" Desc="Sales information
from  Oracle database"/>
   <ExternalTable Id="A52WE4LI.BA000002" Name="Oracle HR" Desc="Human Resources
 information from Oracle database"/>
  </Objects>
</Document>
```

The specified Document object has four objects associated with it through the Objects association name: two PhysicalTable objects and two ExternalTable objects. By default, the GetMetadata method returns the Id, Name and Desc values of the associated objects.

In this second request, a search string is used in the Objects association name of the <METADATA> element to filter the request to get only PhysicalTable objects that are associated with the specified Document object through the Objects association:

```
<GetMetadata>
```

```
   <Metadata>
     <Document Id="A52WE4LI.AT0000RZ">
      <Objects search="PhysicalTable"/>
     </Document>
   </Metadata>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadata>
```

Here is an example of the output from the request:

```
<!-- Using the GETMETADATA method. -->

<Document Id="A52WE4LI.AT0000RZ">
 <Objects>
   <PhysicalTable Id="A52WE4LI.B60000RT" Name="Table1" Desc="Sales table"/>
   <PhysicalTable Id="A52WE4LI.B60000RU" Name="Table2" Desc="Human Resources table"/>
 </Objects>
</Document>
```

The ExternalTable objects that were returned in the first example are excluded from the output of this example.

In this third request, attribute criteria are added to the search criteria string in the <METADATA> element to further filter the request. The request specifies to get PhysicalTable objects that are associated with the specified Document object through the Objects association whose Desc attribute value has the word Sales in it:

```
<GetMetadata>
   <Metadata>
    <Document Id="A52WE4LI.AT0000RZ">
      <Objects search="PhysicalTable[@Desc ? 'Sales']"/>
    </Document>
   </Metadata>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetMetadata>
```

Here is an example of the output from the request:

```
<!-- Using the GETMETADATA method. -->

<Document Id="A52WE4LI.AT0000RZ">
 <Objects>
   <PhysicalTable Id="A52WE4LI.B60000RT" Name="Table1" Desc="Sales table"/>
 </Objects>
</Document>
```

Example of Specifying Search Criteria in the <TEMPLATES> Element

Templates are submitted to the GetMetadata method in the OPTIONS parameter in a <TEMPLATES> element. To submit a template, you must set the OMI_TEMPLATE (4) flag.

To understand how search criteria are processed in a template, consider the following request.

```
<GetMetadata>
   <Metadata>
     <Document Id="A52WE4LI.AT0000RZ"/>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TEMPLATE -->
   <Flags>4</Flags>
   <Options>
     <Templates>
       <Document MetadataCreated="" MetadataUpdated="">
         <Objects search="ExternalTable[@Desc= ?  'Human Resources']"/>
       </Document>
    </Templates>
 </Options>
</GetMetadata>
```

In the method call, note the following:

- The <METADATA> element specifies to get the Document metadata object that has Id="A52WE4LI.AT0000RZ".
- The OMI_TEMPLATE flag (4) instructs the SAS Metadata Server to check for a template in the <OPTIONS> element.
- The <TEMPLATES> element includes a template for the Document metadata type that specifies to get the Name, Desc, MetadataCreated and MetadataUpdated attributes of the Document object, and any ExternalTable objects that are associated to the specified Document through the Objects association and whose Desc attribute has the words Human Resources in it.

Here is an example of the output from the request:

```
<!-- Using the GETMETADATA method. -->

<Document Id="A52WE4LI.AT0000RZ" MetadataCreated="22Aug2008:14:52:24"
MetadataUpdated="22Aug2008: 16:08:45">
  <Objects>
      <ExternalTable Id="A52WE4LI.BA000002"/>
   </Objects>
</Document>
```

Note: When the OMI_TEMPLATE flag is set, GetMetadata gets only the Id attribute for associated objects. If you want to get additional attributes, you need to specify them in another template, or set a flag such as OMI_ALL_SIMPLE (8), which gets all attributes for the specified object and all associated objects.

Example of Specifying Search Criteria in Both Elements

When search criteria are specified in both the <METADATA> and <TEMPLATES> elements, the following rules apply:

- If the search criteria strings specify different association names, both are applied.

- If the search criteria strings specify the same association name, the search criteria string in the <TEMPLATES> element is ignored.

For example, consider this request:

```
<GetMetadata>
   <Metadata>
     <Document Id="A52WE4LI.AT0000RZ">
      <Objects search="ExternalTable[@Desc ? 'Sales']"/>
     </Document>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TEMPLATE flag -->
   <Flags>12</Flags>
   <Options>
    <Templates>
     <Document>
      <Objects search="PhysicalTable[@Desc ? 'Sales']"/>
     </Document>
   </Templates>
 </Options>
</GetMetadata>
```

Before any metadata is retrieved, the properties in the <METADATA> and <TEMPLATES> elements are merged into one list that is used to get the metadata objects. The properties in the <METADATA> element take priority over properties in the <TEMPLATES> element. As a result, additional properties that are specified in the template are added to the list. However, any properties that are duplicated in the template are ignored.

In this example, although the search criteria in the <METADATA> element and in the <TEMPLATES> element specify different metadata types, they specify the same association name (Objects), so the search criteria in the <TEMPLATES> element is ignored. Because the OMI_TEMPLATE flag is set, the SAS Metadata Server returns only the Id value of the specified object and its associated objects.

Now, consider the following request:

```
<GetMetadata>
   <Metadata>
     <Document Id="A52WE4LI.AT0000RZ">
         <ResponsibleParties  search="ResponsibleParty[@Name  ? 'Writer']"/>
     </Document>
   </Metadata>
   <NS>SAS</NS>
   <!-- OMI_TEMPLATE + OMI_ALL_SIMPLE + OMI_SUCCINCT flags -->
   <Flags>2060</Flags>
   <Options>
     <Templates>
      <ResponsibleParty>
          <Persons search="*"/>
      </ResponsibleParty>
   </Templates>
 </Options>
</GetMetadata>
```

In this request, note the following:

- The <METADATA> element specifies to get Document A52WE4LI.AT0000RZ and a search criteria string on the ResponsibleParties association. The search

criteria string specifies to get ResponsibleParty objects that are associated to the specified Document object. The search criteria further specify to get only ResponsibleParty objects that have the word Writer in their Name attribute.

- The sum of the OMI_TEMPLATE (4) + OMI_ALL_SIMPLE (8) + OMI_SUCCINCT (2048) flags instructs the SAS Metadata Server to check for a <TEMPLATES> element in the <OPTIONS> element, get all attributes for the specified object and associated objects, and to omit attributes that do not contain a value or that contain a null value from the results.
- The <OPTIONS> element includes a template in the <TEMPLATES> element that specifies to get objects associated with the returned ResponsibleParty objects through the Persons association.

Here is an example of the output returned by the SAS Metadata Server:

```
 <!-- Using the GETMETADATA method. -->

<Document Id="A52WE4LI.AT0000RZ" Name="AugustPerformance" Desc="Summary report of
NW Region production activity, HR expense, and sales" MetadataCreated=
"22Aug2008:14:52:24" MetadataUpdated="22Aug2008: 16:08:45" PublicType="Document"
URI="Text" UsageVersion="1000000">
  <ResponsibleParties search="ResponsibleParty[@Name  ? 'Writer']">
   <ResponsibleParty Id="A52WE4LI.BN000003"  Name="Technical Writer" MetadataCreated=
"22Aug2008:14:52:24" MetadataUpdated="22Aug2008: 16:08:45" UsageVersion="1000000" >
     <Persons SEARCH="*">
      <Person Id="A52WE4LI.AR0002BE" Desc="Primary Writer" MetadataCreated=
"22Aug2008:14:52:24"  MetadataUpdated="22Aug2008: 16:08:45" Name="Melissa Mark"
PublicType="User" UsageVersion="1000000" />
     </Persons>
  </ResponsibleParty>
 </ResponsibleParties>
</Document>
```

The results show that one ResponsibleParty object is associated with Document A52WE4LI.AT0000RZ through the ResponsibleParties association that has the word Writer in its Name attribute. In turn, this ResponsibleParty object has one object associated with it through the Persons association that describes a person named Melissa Mark.

Using GetMetadata to Get Common Properties for Sets of Objects

If you have a set of objects for which you want to get common properties, you can use the GetMetadata method and set the OMI_INCLUDE_SUBTYPES (16) and OMI_TEMPLATE (4) flags.

In the request:

- Specify the metadata type and Id values of the objects for which you want to get properties in the <METADATA> element.
- Specify additional properties that you want to get in one or more templates in a <TEMPLATES> element within the <OPTIONS> element. The templates that you specify must reference a metadata type that either matches or is a

supertype of the metadata types specified in the <METADATA> element, as defined in the SAS Metadata Model.

The OMI_TEMPLATE flag instructs the GetMetadata method to get the properties specified in the templates for the objects specified in the <METADATA> element. The OMI_INCLUDE_SUBTYPES flag applies the templates to objects that are subtypes of the metadata types specified in the templates.

The following is an example of a GetMetadata method call that requests common properties from multiple objects. It is simple because it specifies one template:

```
<GetMetadata>
   <Metadata>
    <PhysicalTable Id="A58SW16P.B1000001"/>
    <Person Id="A58SW16P.AP0001JL"/>
    <Event Id="A58SW16P.B3000001"/>
    <WorkTable Id="A58SW16P.B4000001"/>
    <Document Id="A58SW16P.AY0000RT"/>
   </Metadata>
<NS>SAS</NS>
<!-- OMI_TEMPLATE + OMI_INCLUDE_SUBTYPES -->
<Flags>20</Flags>
<Options>
  <Templates>
    <Root Name="" Desc="" UsageVersion="">
     <Extensions/>
    </Root>
  </Templates>
 </Options>
</GetMetadata>
```

In the method call, note the following:

- The <METADATA> element specifies five metadata objects from which to get properties.
- The <NS> element specifies the SAS namespace.
- The <FLAGS> element specifies the OMI_TEMPLATE and OMI_INCLUDE_SUBTYPES (4 +16 = 20) flags.
- The <OPTIONS> element includes a <TEMPLATES> element and specifies a template that instructs the method to get attributes and associations for the Root metadata type. The Root metadata type is the supertype of all of the metadata types defined in the SAS Metadata Model. Therefore, the properties requested for the Root object are returned for all of the objects specified in the <METADATA> element.

The following is an example of the output returned by the SAS Metadata Server:

```
<Metadata>
<PhysicalTable Id="A58SW16P.B1000001" Name="Patient Information"
Desc="Information describing an individual patient." UsageVersion="0">
  <Extensions/>
</PhysicalTable>
<Person Id="A58SW16P.AP0001JL" Name="Created Person 1" Desc="Person created for
GetMetadata" UsageVersion="0">
  <Extensions>
     <Extension  Id="A58SW16P.AC0001JL" Name="Attrib 1" Desc="First attribute"
     UsageVersion="0"><Extensions/></Extension>
     <Extension Id="A58SW16P.AC0001JM" Name="Attrib 2" Desc="Second attribute"
```

```
      UsageVersion="0"><Extensions/></Extension>
    </Extensions>
</Person>
<Event Id="A58SW16P.B3000001" Name="Event 1 for GetMetadata" Desc="Event added"
UsageVersion="0">
  <Extensions>
     <Extension Id="A58SW16P.AC0001JN" Name="Attrib 1" Desc="First attribute"
     UsageVersion="0"><Extensions/></Extension>
     <Extension Id="A58SW16P.AC0001JO" Name="Attrib 2" Desc="Second attribute"
     UsageVersion="0"><Extensions/></Extension>
  </Extensions>
</Event>
<WorkTable Id="A58SW16P.B4000001" Name="WorkTable 1 for getmet"
Desc="WorkTable added" UsageVersion="0">
  <Extensions>
     <Extension Id="A58SW16P.AC0001JP" Name="Attrib 1" Desc="First attribute"
     UsageVersion="0"><Extensions/></Extension>
     <Extension Id="A58SW16P.AC0001JQ" Name="Attrib 2" Desc="Second attribute"
     UsageVersion="0"><Extensions/></Extension>
  </Extensions>
</WorkTable>
<Document Id="A58SW16P.AY0000RT" Name="Document 1 for getmet" Desc="doc added"
UsageVersion="0">
  <Extensions>
     <Extension Id="A58SW16P.AC0001JR" Name="Attrib 1" Desc="First attribute"
     UsageVersion="0"><Extensions/></Extension>
     <Extension Id="A58SW16P.AC0001JS" Name="Attrib 2" Desc="Second attribute"
     UsageVersion="0"><Extensions/></Extension>
  </Extensions>
</Document>
</Metadata>
```

In the output, note the following:

- The GetMetadata method returned the Name, Desc, and UsageVersion attribute values for all of the objects specified in the <METADATA> element.
- The PhysicalTable object had no Extension objects associated with it through the Extensions association. The Person, Event, WorkTable, and Document objects each had two Extension objects defined through the Extensions association.
- The method attempted to list Extension objects that were associated to the Extension objects (because Extension is a subtype of Root), but none were found.
- The values of Name, Desc, and UsageVersion attributes were also returned for the Extension objects (because Extension is a subtype of Root).

The following is an example of a GetMetadata method call that sets the OMI_INCLUDE_SUBTYPES flag and specifies multiple templates in the <TEMPLATES> element. When you specify more than one template in the <TEMPLATES> element, the order in which the templates are specified is important. The templates are applied in the order specified. If two or more templates apply to the same metadata type, the first template found is applied. The other templates are ignored. For example:

```
<GetMetadata>
  <Metadata>
    <PhysicalTable Id="A58SW16P.B1000001"/>
    <Person Id="A58SW16P.AP0001JL"/>
```

```
    <Event Id="A58SW16P.B3000001"/>
    <WorkTable Id="A58SW16P.B4000001"/>
    <Document Id="A58SW16P.AY0000RT"/>
  </Metadata>
  <NS>SAS</NS>
  <!-- OMI_TEMPLATE + OMI_INCLUDE_SUBTYPES -->
<Flags>20</Flags>
<Options>
  <Templates>
    <DataTable Name="" UsageVersion="">
      <Documents/>
      <Columns/>
    </DataTable>
    <Root Name="" Desc="" UsageVersion="">
     <Extensions/>
    </Root>
    <Document Name="" Desc=""/>
  </Templates>
 </Options>
</GetMetadata
```

This method call specifies three templates in the <TEMPLATES> element. One template is for the DataTable metadata type. One template is for the Root metadata type. And, one template is for the Document metadata type. Because OMI_INCLUDE_SUBTYPES is set, the GetMetadata method processes the templates as follows:

1. DataTable is the supertype of the PhysicalTable and WorkTable metadata types. Therefore, the method returns the requested Name and UsageVersion attributes for all of these objects, and any objects associated to them through the Documents and Columns associations.
2. Because the first template did not specify what properties to get for any associated Document and Column objects, the method consults the second template. Because the Root metadata type is the supertype of all metadata types, the properties requested for the Root object are retrieved for the Document and Column objects that were retrieved by the first template. The properties are also retrieved for the remaining objects identified in the <METADATA> element.
3. Because the third template specifies a metadata object that has already been processed by the second template, it is ignored.

The following is an example of the output returned by the SAS Metadata Server:

```
<Metadata>
<PhysicalTable Id="A58SW16P.B1000001" Name="Patient Information"
UsageVersion="0">
  <Documents/>
  <Columns>
    <Column Id="A58SW16P.B2000001" Name="Patient ID" Desc="Patient Information"
    UsageVersion="0"><Extensions/></Column><Column Id="A58SW16P.B2000002"
    Name="Initials" Desc="Patient Initials" UsageVersion="0">
    <Extensions/>
  </Column>
  <Column Id="A58SW16P.B2000003" Name="Sex" Desc="Sex of Patient"
  UsageVersion="0"><Extensions/></Column>
  <Column Id="A58SW16P.B2000004"  Name="Date Of Birth" Desc="Date Of Birth"
```

```
UsageVersion="0">
   <Extensions/>
  </Column>
  <Column Id="A58SW16P.B2000005" Name="Sponsor Patient ID"
  Desc="Sponsor Patient Information" UsageVersion="0">
    <Extensions/>
  </Column>
  <Column Id="A58SW16P.B2000006" Name="Weight In Pounds" Desc="Patient
  Weight In Pounds" UsageVersion="0">
    <Extensions/>
  </Column>
  <Column Id="A58SW16P.B2000007" Name="Weight In Kilograms" Desc="Patient
  Weight In Kilograms" UsageVersion="0">
    <Extensions>
      <Extension Id="A58SW16P.AC0000RU" Name="Algorithm" Desc="Algorithm
      for column." UsageVersion="0">
        <Extensions/>
      </Extension>
    </Extensions>
  </Column>
  </Columns>
</PhysicalTable>
<Person Id="A58SW16P.AP0001JL" Name="Created Person 1 for getmet"
Desc="getmet07" UsageVersion="0">
   <Extensions>
     <Extension Id="A58SW16P.AC0001JL" Name="Attrib 1" Desc="First attribute"
     UsageVersion="0">
       <Extensions/>
     </Extension>
     <Extension Id="A58SW16P.AC0001JM" Name="Attrib 2" Desc="Second attribute"
     UsageVersion="0">
       <Extensions/>
     </Extension>
  </Extensions>
</Person>
<Event Id="A58SW16P.B3000001" Name="Event 1 for getmet" Desc="Event added"
UsageVersion="0">
  <Extensions>
    <Extension Id="A58SW16P.AC0001JN" Name="Attrib 1" Desc="First attribute"
    UsageVersion="0">
      <Extensions/>
    </Extension>
    <Extension Id="A58SW16P.AC0001JO" Name="Attrib 2" Desc="Second attribute"
    UsageVersion="0">
     <Extensions/>
    </Extension>
  </Extensions>
</Event>
<WorkTable Id="A58SW16P.B4000001" Name="WorkTable 1 for getmet"
UsageVersion="0">
  <Documents/>
  <Columns/>
</WorkTable>
<Document Id="A58SW16P.AY0000RT" Name="Document 1 for getmet" Desc="doc added"
UsageVersion="0">
  <Extensions>
```

```
    <Extension  Id="A58SW16P.AC0001JR" Name="Attrib 1" Desc="First attribute"
    UsageVersion="0">
      <Extensions/>
    </Extension>
    <Extension Id="A58SW16P.AC0001JS" Name="Attrib 2" Desc="Second attribute"
    UsageVersion="0">
      <Extensions/>
    </Extension>
  </Extensions>
</Document>
</Metadata>
<Ns>SAS</Ns>
<Flags>20</Flags>
<Options>
<Templates>
 <DataTable Name="" usageVersion=""><Documents/><Columns/></DataTable>
 <Root Name="" Desc="" usageVersion=""><Extensions/></Root>
 <Document Name="" Desc=""/>
</Templates>
</Options>
</GetMetadata>
```

In the output, the PhysicalTable object has no associated Document objects and has seven associated Column objects. One of the Column objects has an Extension object defined for it. The WorkTable object has no associated Document or Column objects. The method returned the properties requested for the Root metadata type for all remaining objects. If the OMI_INCLUDE_SUBTYPES flag had not been set in the method call, the results would have been different. In that case, the templates would have been applied in the order given, but the templates that specify the DataTable and Root metadata types would have been ignored.

SAS includes a second template form that enables you to map templates to the metadata types that they are meant to expand. For more information, see “New Get Template Form” on page 344 and “Templates and the OMI_INCLUDE_SUBTYPES Flag” on page 352.

Including Objects from Project Repositories in a Public Query

A GetMetadata method call that requests associated objects and is issued in a public (foundation or custom) repository returns information about cross-repository references to objects in other public repositories, by default. If you have a need to include cross-repository references to objects in project repositories (for example, to determine whether any of an object's associated objects are checked out for development), you can set the OMI_DEPENDENCY_USED_BY (16384) flag in the method call. The following is an example of a GetMetadata request that gets cross-repository references to objects in project repositories:

```
<GetMetadata>
   <Metadata>
     <PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"/>
   </Metadata>
   <NS>SAS</NS>
```

```
   <!-- OMI_ALL (1) + OMI_DEPENDENCY_USED_BY (16384)
    + OMI_SUCCINCT (2048) flags -->
   <Flags>18433</Flags>
   <Options/>
</GetMetadata>
```

In the request, note the following:

- The OMI_ALL flag instructs the method to return all properties (attributes and associations) that are defined for PhysicalTable A53TPPVI.A4000001.
- The OMI_DEPENDENCY_USED_BY flag instructs the method to include cross-repository references to objects in project repositories in the results. The method returns associated objects from all project repositories. There is no way to search specific project repositories.
- The OMI_SUCCINCT flag instructs the method to include only properties that contain a value in them in the results.

Combining GetMetadata Flags

When OMI_SUCCINCT (2048) is added to any flag combination, only attributes that are not null and only associations that have associated objects defined are returned.

When OMI_ALL (1) and OMI_ALL_SIMPLE (8) are set together, the SAS Metadata Server gets all attributes and direct associations for the requested object, and all attributes of the associated objects returned by OMI_ALL.

When OMI_ALL_SIMPLE (8) and OMI_TEMPLATE (4) are set together, the SAS Metadata Server gets all attributes for the specified object and all attributes for associated objects that are returned by the template.

When OMI_ALL (1) and OMI_TEMPLATE (4) are set together, the SAS Metadata Server gets all possible attributes and associations for the specified object and the Id attribute of associated objects returned by the OMI_ALL flag or template.

When OMI_INCLUDE_SUBTYPES (16) is set without OMI_TEMPLATE and a template, it is ignored. When OMI_INCLUDE_SUBTYPES and OMI_TEMPLATE (4) are set together, the SAS Metadata Server gets the properties requested in the <TEMPLATES> element for the objects specified in the <METADATA> element if they are the same metadata type or a subtype of the metadata type specified in the template.

Using the OMI_FULL_OBJECT Flag

The following is an example of a GetMetadata method call that sets the OMI_FULL_OBJECT (2) flag. The OMI_FULL_OBJECT flag instructs the metadata server to use a type definition from the SAS type dictionary to identify the association names to expand when getting the specified object. The flag is effective only if the specified object is a PrimaryType subtype in the SAS Metadata Model and stores the name of a type definition in the PublicType attribute. If the specified

object is not a PrimaryType subtype or does not store a PublicType value, the flag has no effect.

```
<GetMetadata>
 <Metadata>
   <Column Id="A5TJRDIT.B700002E"/>
 </Metadata>
 <NS>SAS</NS>
<!-- OMI_FULL_OBJECT -->
<Flags>2</Flags>
 <Options/>
</GetMetadata>
```

Here is an example output from the request:

```
<!-- Using the GETMETADATA method -->
<Column Id="A5TJRDIT.B700002E">
    <AccessControls/>
    <Table>
        <PhysicalTable ObjRef="A5TJRDIT.B200000J" Name="EMPINFO" Desc=""
PublicType="Table" UsageVersion="1000000"/>
</Table>
    <Notes/>
    <Extensions/>
    <Documents/>
    <Properties/>
    <PropertySets/>
    <ForeignKeyAssociations/>
    <Keys>
        <UniqueKey ObjRef="A5TJRDIT.BH000003" Name="EMPINFO.ic_id" Desc=""/>
</Keys>
    <Keywords/>
</Column>
```

Although a Column object cannot stand alone in a SAS Metadata Repository, it is categorized as a PrimaryType subtype in the model, because Column objects can have different security defined on them than the security defined on their owning table (for example, to make some columns available to some users and not others).

When OMI_FULL_OBJECT is set with no other GetMetadata flags, the method retrieves the Id, Name, and Desc values for all secondary objects retrieved by the type definition, unless an associated object is designated as a connection point. A connection point is an association to another common or shared object that is external to this object definition. When a connection point is encountered, the ObjRef attribute is used to identify the associated object in the output XML instead of the Id attribute, and the object's PublicType and UsageVersion values are returned in addition to Name and Desc.

Any attributes that are explicitly specified in the GetMetadata request's INMETADATA parameter are returned only for the object specified in the INMETADATA parameter.

OMI_FULL_OBJECT interacts with other GetMetadata flags as follows:

- OMI_ALL_SIMPLE: Returns all simple attributes for all objects returned by the type definition, unless they are connection points. For connection points, only ObjRef, Name, Desc, PublicType, and UsageVersion are returned.
- OMI_ALL: Behaves the same as OMI_ALL_SIMPLE, unless OMI_TEMPLATE is also set.

- OMI_TEMPLATE: When OMI_TEMPLATE and OMI_FULL_OBJECT are set together, the SAS Metadata Server behaves as if OMI_FULL_OBJECT were not set, and looks for a template in the <TEMPLATES> element of the OPTIONS parameter for information to process the request. Because OMI_FULL_OBJECT is set, however, any embedded or nested objects in the override template are expanded as well. These nested definitions can be overridden by a named template. For more information, see "User-Defined Template Forms" on page 338.

  An embedded object is an object that can stand alone in the SAS Metadata Repository or be part of another object. A nested object is an object that is part of another object and cannot exist in the SAS Metadata Repository without an association to the owning object. Prompt is an example of an embedded object. Column is an example of a nested object.
- OMI_LOCK: Locks all of the objects returned by the object's type definition, including embedded or nested objects.
- OMI_UNLOCK and OMI_UNLOCK_FORCE: Unlock all of the objects returned by the object's type definition, including embedded or nested objects.
- OMI_INCLUDE_SUBTYPES: Has no affect on the information returned by the type definition.
- OMI_SUCCINCT, OMI_NOEXPAND_DUPS, OMI_NOFORMAT, and OMI_DEPENDENCY_USEDBY: Behave normally on all objects (embedded or nested) that are returned by the type definition.

15

Using Templates

Understanding Templates

Description of a Template

A template is a metadata property string that you specify in the <TEMPLATES> element in the OPTIONS parameter of a GetMetadata, GetMetadataObjects, or DeleteMetadata method call. The template is submitted to the SAS Metadata Server with the OMI_TEMPLATE (4) flag. In GetMetadataObjects, you must set the OMI_GET_METADATA (256) flag in addition to OMI_TEMPLATE.

Purpose of a Template in the Get Methods

In a GetMetadata or GetMetadataObjects method call, a template specifies attributes or associations to retrieve for the main metadata type of the method call, one of its subtypes, or an associated metadata type. This is beyond attributes and associations that are requested in the INMETADATA parameter and by the method's flags. While the OMI_ALL flag returns all attributes and directly associated objects for the specified object, and the OMI_SIMPLE flag returns all simple attributes for the specified object and any associated objects that are requested, neither flag affects the associations of the associated objects.

A template enables you to exactly specify the attributes and associations that you want to retrieve for a specified metadata type. The specified metadata type can be the main metadata type in the request or an associated metadata type. To request associations of associated objects, you specify additional templates.

Prior to SAS 9.3, user-defined templates were the only way to retrieve information about the associations of associated objects. Then, SAS added the OMI_FULL_OBJECT (2) flag. The OMI_FULL_OBJECT flag returns the full logical metadata definition of objects that are managed by the SAS type dictionary. For more information, see “Using the OMI_FULL_OBJECT Flag” on page 334.

User-defined templates can be used to expand the logical metadata definitions of objects that are not managed by the SAS type dictionary. You can also use templates to get a portion of the information that would be returned by the OMI_FULL_OBJECT flag.

Purpose of a Template in DeleteMetadata

In a DeleteMetadata method call, a user-defined template enables you to specify association names that should be traversed to locate associated objects to be deleted in addition to the specified metadata object. If the specified metadata object is managed by the SAS type dictionary, the user-defined template overrides information about the object’s logical metadata definition from the SAS type dictionary, which would normally be used to delete the metadata object and its associated objects.

User-Defined Template Forms

You can submit templates using one of two template forms:

- In the legacy form, one or more metadata property strings are specified directly in the <TEMPLATES> element in the OPTIONS parameter.
- In a newer form, metadata property strings are submitted in one and optionally more <TEMPLATE> subelements in the <TEMPLATES> element in the OPTIONS parameter. Each <TEMPLATE> subelement specifies a TemplateName attribute, which corresponds to a matching TemplateName value in the metadata property string that the template will be expanding. The TemplateName attribute is supported in the INMETADATA property string and in a template.

Using multiple named templates is useful when you need to access many different logical groupings of metadata objects in the same request. For example, when multiple metadata property strings are submitted in the GetMetadata or DeleteMetadata method’s INMETADATA parameter, the TemplateName attribute can direct the SAS Metadata Server to the appropriate template to expand each request. When associated objects are requested in a template, the TemplateName attribute can redirect the SAS Metadata Server to another template for processing, before returning to the initial template.

The new form also supports attributes that enable you to specify different handling of object instances of the same metadata type. These template attributes are described in “Template Attributes” on page 339.

When a request specifies multiple templates, it is easy (and undesirable) to specify associations that refer back to each other. The OMI_NOEXPAND_DUPS (524288) flag is provided to track the objects that are expanded by templates by object ID, and prevent the server from expanding an object more than once. See “Templates and the OMI_NOEXPAND_DUPS Flag” on page 352.

For information about creating user-defined templates, see the following:

- “Creating Templates for the Get Methods” on page 340
- “Creating a Template for DeleteMetadata” on page 396

Template Attributes

The following table shows attributes that are supported in a template. These attributes filter the objects and information about the objects that are returned.

Attribute	Location*	Description
Match="*criteria*"	AssociatedMetadataType	Available in the new form only, enables you to select object instances of the indicated metadata type for different handling based on the value of a key attribute or association. The attribute or association criteria must be valid for the indicated metadata type. Object instances that do not meet the match criteria are still returned by the request. However, the SAS Metadata Server returns only their Id values. Object instances that meet the match criteria have their attributes expanded.
Search="*criteria*"	AssociationName	Available in both the legacy and new forms, enables you to specify search criteria to limit the associated object instances that are returned for the specified association name. The SAS Metadata Server returns only associated object instances that meet the specified search criteria. Other valid object types are ignored.

Attribute	Location*	Description
TemplateExpand="Yes | No"	Associated MetadataType	Available in the new form only, specifies to allow (Yes) or suppress (No) the expansion of objects of the indicated associated metadata type. In GetMetadata, it controls whether a template is applied to the objects. The default value when TemplateExpand is omitted is Yes. When TemplateExpand="No", the SAS Metadata Server returns only the Id values of any objects that are found. In DeleteMetadata, TemplateExpand="No" specifies to ignore (not delete) the associated objects.
TemplateName="*name*"	MetadataType in INMETADATA and AssociatedMetadataType in a template	Available in the new form only, specifies the name of an alternative template to use for the specified metadata type.

* The **Location** field refers to these locations within a template:

```
<MetadataType>
   <AssociationName Search="">
     <AssociatedMetadataType Match="" TemplateExpand="" TemplateName=""/>
   </AssociationName>
</MetadataType>
```

The Match and Search attributes support the full syntax that is available for the GetMetadataObjects <XMLSELECT search="*criteria*"/> option. For more information, see “<XMLSELECT search="criteria"/> Syntax” on page 373.

Creating Templates for the Get Methods

Basic Structure of a Get Template

In a GetMetadata or GetMetadataObjects method call, a template is a metadata property string that specifies attributes and associations to retrieve for a specified metadata type.

- A template that requests simple attributes looks like this:

  ```
  <MetadataType Attribute1="" Attribute2="" Attribute3=""/>
  ```

  MetadataType can be the same metadata type as what is specified in the INMETADATA parameter, an associated metadata type that is requested by an association name in the INMETADATA parameter, or, if the

OMI_INCLUDE_SUBTYPES (16) flag is set, a supertype of the metadata type in the INMETADATA parameter. *Attributen* are attributes that are defined for the specified metadata type in the SAS Metadata Model.

- A template that requests associated objects looks like this:

```
<MetadataType>
  <AssociationName/>
</MetadataType>
```

MetadataType has the same definition as above. *AssociationName* is an association name that is valid for *MetadataType* as defined in the SAS Metadata Model. This request returns associated object instances of all metadata types that are valid for the specified *AssociationName*. You can specify as many association name elements as you need to, as long as each name is valid for *MetadataType*.

- A template can specify to retrieve both attributes and associations in the same request. The attributes and associations must be valid for *MetadataType*. For example:

```
<MetadataType Id="" Name="" Attributen="">
  <AssociationName1/>
  <AssociationName2/>
</MetadataType>
```

- When templates are used, the GetMetadata method returns only the Id attribute of associated objects. To get additional attributes for associated objects, set the OMI_ALL_SIMPLE (8) flag, which gets all simple attributes (*Attributen*) for all requested objects and associated objects. Or, specify additional templates that request specific attributes, as follows:

```
<MetadataType>
   <AssociationName/>
 </MetadataType>
 <AssociatedMetadataType1 Id="" Name="" Attributen=""/>
 <AssociatedMetadataType2 Id="" Name="" Attributen=""/>
```

AssociatedMetadataType1 and *AssociatedMetadataType2* are metadata types that are valid for *AssociationName* in the SAS Metadata Model. Specifying an associated metadata type does not prevent GetMetadata from returning information about objects of other metadata types that are valid for *AssociationName*. It just specifies attributes to retrieve for objects of the specified associated metadata type. The request returns the Id values of any other valid metadata types that are found.

- You can specify associated object requests as deeply within an association path as you would like. For example:

```
<MetadataType>
   <AssociationName1/>
 </MetadataType>
 <AssociatedMetadataType Name="" Attributen="">
   <AssociationName2/>
 </AssociatedMetadataType>
<AssociatedMetadataType2 Name="" Attributen="">
   <AssociationName3/>
 </AssociatedMetadataType2>
<AssociatedMetadataType3  Name="" Attributen="">
```

```
   <AssociationName4/>
 </AssociatedMetadataType3>
<AssociatedMetadataType4 Name="" Attributen=""/>
```

In this example, *AssociatedMetadataType* is a valid metadata type for *AssociationName1* in the SAS Metadata Model. *AssociatedMetadataType2* is a valid metadata type for *AssociationName2* in the SAS Metadata Model. *AssociatedMetadataType3* is a valid metadata type for *AssociationName3* in the SAS Metadata Model, and so on.

- In the preceding request, the SAS Metadata Server returns the specified information for each specified associated metadata type, and the Id values of object instances of other metadata types that are valid for each *AssociationName*. To filter the objects that are returned for an association, you can use the Search attribute in *AssociationName*. When Search is used, the SAS Metadata Server retrieves only objects that meet the criteria specified in the Search attribute. The Search attribute supports filtering by metadata type, attribute, and association path criteria, alone or combined. For example:
  - To get only associated objects of a specified metadata type, specify Search as follows:

    ```
    <MetadataType>
       <AssociationName Search="AssociatedMetadataType"/>
    </MetadataType>
    ```

  - To get associated objects of a specified metadata type that have Attribute="X", specify Search as follows:

    ```
    <MetadataType>
       <AssociationName Search="AssociatedMetadataType[@Attribute='X']"/>
    </MetadataType>
    ```

  - To get only associated objects of a specified metadata type that have a specified association, specify Search as follows:

    ```
    <MetadataType>
       <AssociationName Search="AssociatedMetadataType[AssociationName/
    AssociatedMetadataType2]"/>
    </MetadataType>
    ```

Examples of Basic Output

Here are some simple examples of how templates are applied with real metadata types. Here is an example of a template that requests attribute values for a PhysicalTable object:

```
<PhysicalTable IsDBMSView="" IsEncrypted="" SASTableName=""/>
```

This is sample output from the request:

```
<PhysicalTable Id="A5TJRDIT.B200000J" IsDBMSView="0" IsEncrypted="0"
SASTableName="EMPINFO"/>
```

Here is an example of a template that requests objects associated to the table through the Columns association:

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

This is sample output from the request, reformatted for readability:

```
<PhysicalTable Id="A5TJRDIT.B200000J">
   <Columns>
     <Column Id="A5TJRDIT.B700002A"/>
     <Column Id="A5TJRDIT.B700002B"/>
     <Column Id="A5TJRDIT.B700002C"/>
     <Column Id="A5TJRDIT.B700002D"/>
     <Column Id="A5TJRDIT.B700002E"/>
     <ColumnRange Id="A5TJRDIT.BK000002"/>
   </Columns>
</PhysicalTable>
```

The SAS Metadata Server returns the Id values of all associated objects of all metadata types that are valid for the Columns association name. The Columns association name supports associations to objects of the Column and ColumnRange metadata types.

The following example requests additional attributes for the Column and ColumnRange object:

```
<PhysicalTable>
   <Columns/>
</PhysicalTable>
 <Column Id="" Name="" MetadataCreated="" MetadataUpdated=""/>
```

This is sample output from the request, reformatted for readability:

```
<PhysicalTable Id="A5TJRDIT.B200000J">
 <Columns>
   <Column Id="A5TJRDIT.B700002A" Name="DEPTCODE" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="10Jan2011:21:20:36"/>
   <Column Id="A5TJRDIT.B700002B" Name="NAME" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="10Jan2011:21:20:36"/>
   <Column Id="A5TJRDIT.B700002C" Name="ADDR1" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="10Jan2011:21:20:36"/>
   <Column Id="A5TJRDIT.B700002D" Name="ADDR2" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="10Jan2011:21:20:36"/>
   <Column Id="A5TJRDIT.B700002E" Name="IDNUM" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="14Jan2011:22:37:17"/>
   <ColumnRange Id="A5TJRDIT.BK000002"/>
 </Columns>
</PhysicalTable>
```

The request continues to return the Id value of the one ColumnRange object.

Here is an example of a template that requests only associated ColumnRange objects, and a template that requests basic attributes for the ColumnRange object:

```
<PhysicalTable>
  <Columns search="ColumnRange"/>
</PhysicalTable>
<ColumnRange Id="" Name="" MetadataCreated="" MetadataUpdated=""/>
```

Here is sample output from the request, reformatted for readability:

```
<PhysicalTable Id="A5TJRDIT.B200000J">
 <Columns SEARCH="ColumnRange">
  <ColumnRange Id="A5TJRDIT.BK000002" Name="Test" MetadataCreated="15Mar2011:19:37:31"
MetadataUpdated="15Mar2011:19:37:31"/>
 </Columns>
</PhysicalTable>
```

Here is an example of a template that requests only associated Column objects that have Name="DEPTCODE", and a template that requests basic attributes for the Column object:

```
<PhysicalTable>
  <Columns search="Column[@Name='DEPTCODE']"/>
</PhysicalTable>
<Column Id="" Name="" MetadataCreated="" MetadataUpdated=""/>
```

Here is sample output from the request, reformatted for readability:

```
<PhysicalTable Id="A5TJRDIT.B200000J">
 <Columns SEARCH="Column[@Name='DEPTCODE']">
  <Column Id="A5TJRDIT.B700002A" Name="DEPTCODE" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="10Jan2011:21:20:36"/>
 </Columns>
</PhysicalTable>
```

Here is an example of a template that requests only Column objects that have indexes defined for them, and a template that requests basic attributes for the Column object:

```
<PhysicalTable>
  <Columns search="Column[Indexes/Index]"/>
</PhysicalTable>
<Column Id="" Name="" MetadataCreated="" MetadataUpdated=""/>
```

Here is sample output from the request, reformatted for readability:

```
<PhysicalTable Id="A5TJRDIT.B200000J">
  <Columns SEARCH="Column[Indexes/Index]">
   <Column Id="A5TJRDIT.B700002E" Name="IDNUM" MetadataCreated="10Jan2011:21:20:36"
MetadataUpdated="14Jan2011:22:37:17"/>
  </Columns>
</PhysicalTable>
```

The GetMetadata method supports the Search attribute in the association name subelements of the INMETADATA property string and the <TEMPLATES> element. When search criteria are specified in both the INMETADATA parameter and in the <TEMPLATES> element, the criteria in the INMETADATA parameter takes precedence. For more information, see “Filtering the Associated Objects That Are Returned by a GetMetadata Request” on page 323.

The GetMetadataObjects method supports the Search attribute in a template only. For more information, see “Filtering the Associated Objects That Are Returned by a GetMetadataObjects Request” on page 389.

New Get Template Form

In GetMetadata, the new form has the following basic format:

```
<GetMetadata>
 <Metadata>
  <MetadataType TemplateName="1"/>
 </Metadata>
 <Ns>SAS</Ns>
 <Flags>4</Flags>
```

```
<Options>
<Templates>
<Template TemplateName="1"> 1
<MetadataType Name="" Attribute1="" Attribute2="" Attribute3="">
 <AssociationName1 Search="AssociatedMetadataType1"/> 2
  <AssociatedMetadataType1 Match="criteria1" TemplateExpand="Yes" TemplateName="2"/>
  <AssociatedMetadataType1 Match="criteria2" TemplateExpand="Yes" TemplateName="3"/>
  <AssociatedMetadataType1 TemplateExpand="No"/>
 </AssociationName1>
 <AssociationName2/> 3
</MetadataType>
<AssociatedMetadataType2 Name="" Attribute1="">  4
  <AssociationName/>
</AssociatedMetadataType2>
</Template>
<Template TemplateName="2"> 5
<AssociatedMetadataType Name="" Attribute1="" Attribute2="" >
  <AssociationNameX/>
</AssociatedMetadataType>
</Template>
<Template TemplateName="3">
<AssociatedMetadataType Name="" Attribute1="" Attribute2="" >
  <AssociationNameY/>
</AssociatedMetadataType>
</Template>
</Templates>
</Options>
</GetMetadata>
```

1 Template property strings are submitted in a <TEMPLATE> subelement of the <TEMPLATES> element in the OPTIONS parameter. The TemplateName attribute value in the first <TEMPLATE> subelement maps to the TemplateName value in the metadata property string in the INMETADATA parameter of the method call. The template property string specifies the primary metadata type and the attributes and associations that will be expanded in the request.

Note: The <TEMPLATE> subelement, which expands the primary object, is shown first to simplify the explanation. But, it does not need to be specified first. Named templates can be listed in any order in the <TEMPLATES> element.

2 The first *AssociationName* subelement specifies the Search attribute to filter the request to return only objects of metadata type *AssociatedMetadataType1*. It is followed by subelements to indicate that different handling of objects of *AssociatedMetadataType1* is needed, based on criteria specified in the Match attribute.

- Object instances that meet Match= "*criteria1*" will be expanded by the template indicated in TemplateName="2".
- Object instances that meet Match="*criteria2*" will be expanded by the template indicated in TemplateName="3".
- Object instances that meet neither criteria will have only their Id values returned.

When a TemplateName attribute is encountered in another template, the SAS Metadata Server deactivates the current template, and makes the new template

active. When processing of the named template completes, it reactivates and continues with the initial template.

3 The second *AssociationName* subelement behaves the same as it did in the legacy template. The SAS Metadata Server will return object instances of all metadata types that are valid for the specified association name. It returns only the Id values of any objects that are found unless you specify additional templates to indicate the attributes and associations that you would like returned for each object type. When reading additional templates, the server makes an exact metadata type match, unless the OMI_INCLUDE_SUBTYPES flag is set. For more information, see "Templates and the OMI_INCLUDE_SUBTYPES Flag" on page 352.

4 This is an additional template. You can specify as many additional templates as you need. You can invoke the TemplateName attribute in an additional template as well.

5 This is the second of the three named templates in the request. *AssociatedMetadataType* is the same metadata type as what is in the property string containing the corresponding TemplateName attribute value, or it is a supertype if the OMI_INCLUDE_SUBTYPES flag is set.

In GetMetadataObjects, the new form has the following basic format:

```
<GetMetadataObjects>
<Reposid>A0000001.XXXXXXXX</Reposid>
<Type>MetadataType</Type>
<Objects/>
<Ns>SAS</Ns>
<Flags>260</Flags>
<Options>
<Templates>
<Template TemplateName="MetadataType"> 1
<MetadataType>
 <AssociationName1 Search="AssociatedMetadataType1"> 2
  <AssociatedMetadataType1 Match="criteria1" TemplateExpand="Yes" TemplateName="1"/>
  <AssociatedMetadataType1 Match="criteria2" TemplateExpand="Yes" TemplateName="2"/>
  <AssociatedMetadataType1 TemplateExpand="No"/>
 </AssociationName1>
 <AssociationName2/>
</MetadataType>
<AssociatedMetadataType2 Name="" Attribute1=""> 3
  <AssociationName/>
</AssociatedMetadataType2>
</Template>
<Template TemplateName="1">
<AssociatedMetadataType Name="" Attribute="">
  <AssociationNameA/>
</AssociatedMetadataType>
</Template>
<Template TemplateName="2">
<AssociatedMetadataType Name="" Attribute="">
  <AssociationNameB/>
</AssociatedMetadataType>
</Template>
</Templates>
</Options>
</GetMetadataObjects>
```

1. Because GetMetadataObjects does not accept an input metadata property string, you specify the metadata type from the TYPE parameter in the TemplateName attribute of the <TEMPLATE> subelement. The metadata property string in the <TEMPLATE> subelement specifies the metadata type, attributes, and association names to expand.
2. The Search, Match, TemplateName, and TemplateExpand attributes behave as they do in GetMetadata.
3. Additional templates behave as they do in GetMetadata.

Using the TemplateName attribute to redirect processing to a new template should be reserved for complex requests. Additional templates are adequate for expanding associated objects most of the time.

Examples of How the New Template Form Can Be Used

The following example uses the new template form in a GetMetadata request to query the contents of a folder. A folder is represented in a SAS Metadata Repository by the Tree metadata type. A Tree has a Members association to the objects that it contains. The initial template, named “MyTree”, uses the TemplateName attribute to specify different templates for expanding PhysicalTable objects, ClassifierMap objects, and Job objects. The Match attribute specifies different handling of the ClassifierMap objects that have a PublicType attribute value of “StoredProcess” versus those that do not. ClassifierMap objects that do not meet the Match attribute requirements, and other metadata objects found that are valid for the Members association name will be expanded with the attributes defined in the named template specified for the Root object. The request sets the OMI_INCLUDE_SUBTYPES (16) flag, in addition to the OMI_TEMPLATES (4) flag. The OMI_INCLUDE_SUBTYPES flag causes the server to return the properties specified for the Root metadata type for all subtypes.

Template “StoredProcess” specifies attributes to retrieve, and specifies to retrieve any objects associated through the ComputeLocations and Prompts association names. Template “Job” specifies attributes to retrieve for each Job object, and specifies to retrieve any objects associated through the JobActivities, TransformationSources, and TransformationTargets association names. An additional template requests attributes and associations to expand for any TransformationActivity objects that are found. Template “Table” specifies attributes to retrieve for each PhysicalTable object that is found, and requests any objects associated to PhysicalTable objects through the Columns, Indexes, and UniqueKeys association names. Additional templates request attributes for any returned Column, Index, and UniqueKey objects.

```
<GetMetadata>
 <Metadata>
 <Tree Id="A5MFRU33.AI0001JM" TemplateName="MyTree"/>
 </Metadata>
 <Ns>SAS</Ns>
 <Flags>20</Flags>
 <Options>
  <Templates>
   <Template TemplateName="MyTree">
   <Tree Id="" Name="" Desc="" PublicType="">
   <Members>
   <PhysicalTable TemplateName="Table"/>
```

```
<ClassifierMap Match="@PublicType='StoredProcess'"
TemplateName="StoredProcess"/>
<Job TemplateName="Job"/>
<Root Templatename="dflt"/>
</Members>
</Tree>
</Template>
<Template TemplateName="dflt">
<Root Id="" Name=""/>
</Template>
<Template TemplateName="StoredProcess">
<ClassifierMap Id="" Name="" Desc="" PublicType=""
IsActive="" IsUserDefined="" TransformRole="">
<ComputeLocations/>
<Prompts/>
</ClassifierMap>
<Root Id="" Name=""/>
</Template>
<Template TemplateName="Job">
<Job Id="" Name="" Desc="" PublicType="" IsActive=""
IsUserDefined="" TransformRole="">
<JobActivities/>
<TransformationSources/>
<TransformationTargets/>
</Job>
<TransformationActivity Id="" Name="" Desc="">
<Steps/>
</TransformationActivity>
<Root Name="" Desc=""/>
</Template>
<Template TemplateName="Table">
<PhysicalTable Id="" Name="" Desc="" PublicType=""
IsCompressed="" DBMSType="">
<Columns/>
<Indexes/>
<UniqueKeys/>
</PhysicalTable>
<Column Id="" Name="" Desc="" SASColumnType="" SASColumnLength=""
SASFormat="" SASInformat=""/>
<Index Id="" Name="" Desc="" IndexName="" IsClustered=""
IsUnique=""/>
<UniqueKey Id="" Name="" Desc=""/>
</Template>
</Templates>
</Options>
</GetMetadata>
```

Here is sample output from the request, reformatted for readability:

```
<Tree Id="A5MFRU33.AI0001JM" TemplateName="MyTree" Name="Untitled"
Desc="" PublicType="Folder">
<Members>
<ClassifierMap Id="A5MFRU33.BI000001" Name="Test" Desc="" PublicType="StoredProcess"
IsActive="1" IsUserDefined="0" TransformRole="StoredProcess">
<ComputeLocations>
<ServerContext Id="A5MFRU33.AV000001" Name="SASApp"/>
</ComputeLocations>
```

```
<Prompts>
<PromptGroup Id="A5MFRU33.BJ000001" Name="Parameters"/>
</Prompts>
</ClassifierMap>
<Job Id="A5MFRU33.BF0000RT" Name="DailyJob" Desc="" PublicType="Job" IsActive="1"
IsUserDefined="0" TransformRole="">
<JobActivities>
<TransformationActivity Id="A5MFRU33.BH0000RT" Name="New Transformation Activity"
 Desc="">
<Steps>
<TransformationStep Id="A5MFRU33.BK000001" Name="SAS Splitter" Desc=""/>
<TransformationStep Id="A5MFRU33.BK000002" Name="Transpose" Desc=""/>
<TransformationStep Id="A5MFRU33.BK000003" Name="DateTime_ISO8601conv" Desc=""/>
</Steps>
</TransformationActivity>
</JobActivities>
<TransformationSources/>
<TransformationTargets/>
</Job>
<PhysicalTable Id="A5MFRU33.B50007PU" Name="ACCOUNT_CLOSE_REASON" Desc=""
PublicType="Table" IsCompressed="0" DBMSType="REMOTE">
<Columns>
<Column Id="A5MFRU33.B600099J" Name="PROCESSED_DTTM" Desc="The timestamp
for the last time a record was processed, typically by ETL load processing,
but could also be updated when inter ETL cycle modifications are made to
a record." SASColumnType="N" SASColumnLength="8" SASFormat="NLDATM21."
SASInformat=""/>
<Column Id="A5MFRU33.B600099K" Name="CLOSE_REASON_CD" Desc="" SASColumnType="C"
SASColumnLength="3" SASFormat="" SASInformat=""/>
<Column Id="A5MFRU33.B600099L" Name="VALID_FROM_DTTM" Desc="" SASColumnType="N"
SASColumnLength="8" SASFormat="NLDATM21." SASInformat=""/>
</Columns>
<Indexes>
<Index Id="A5MFRU33.B70007PV" Name="XPKACCOUNT_CLOSE_REASON" Desc=""
IndexName="XPKACCOUNT_CLOSE_REASON" IsClustered="0" IsUnique="1"/>
<Index Id="A5MFRU33.B70007PW" Name="CLOSE_REASON_CD" Desc="" IndexName=""
IsClustered="0" IsUnique="1"/>
</Indexes>
<UniqueKeys>
<UniqueKey Id="A5MFRU33.B80007PV" Name="XPKACCOUNT_CLOSE_REASON" Desc=""/>
<UniqueKey Id="A5MFRU33.B80007PW" Name="XAK1ACCOUNT_CLOSE_REASON" Desc=""/>
</UniqueKeys>
</PhysicalTable>
<Document Id="A5MFRU33.BC0003UX" Name="note for column"/>
<Document Id="A5MFRU33.BC0003UY" Name="rating.Note"/>
</Members>
</Tree>
```

The request returns information about one stored process, one job, one table, and all requested information for each object. In addition, it returns two Document objects that are in the folder.

The following is an example of a GetMetadataObjects request that uses the new template form. The request sets the OMI_GET_METADATA (256) and OMI_TEMPLATES (4) flags, which are required to activate templates processing in GetMetadataObjects. This request specifies to list tables that have columns that are not numeric that have indexes defined for them, and requests attributes for the

columns and indexes. For more information about the NOT logical operator, see "NOT Logical Operator in AttributeCriteria Component" on page 377.

```
<GetMetadataObjects>
<Reposid>A0000000.A5TJRDIT</Reposid>
<Type>PhysicalTable</Type>
<Objects/>
<Ns>SAS</Ns>
<Flags>260</Flags>
<Options>
<Templates>
<Template TemplateName="PhysicalTable">
<PhysicalTable Name="">
<Columns Search="Column[Not(@SASColumnType='N')]">
<Column Match="Column[Indexes/Index]"/>
<Column TemplateExpand="No"/>
</Columns>
<Indexes/>
</PhysicalTable>
<Column SASColumnName="" SASColumnType="" SASColumnLength="">
 <Indexes/>
</Column>
<Index IndexName="" IsClustered="" IsNoMiss="" IsUnique=""/>
</Template>
</Templates>
</Options>
</GetMetadataObjects>
```

Here is sample output from the request, reformatted for readability:

```
<GetMetadataObjects>
<Reposid>A0000000.A5TJRDIT</Reposid>
<Type>PhysicalTable</Type>
<Objects >
   <PhysicalTable Id="A5TJRDIT.B2000001" Name="ODSSTYLE">
   <Columns SEARCH="Column[Not(@SASColumnType='N')]">
   <Column Id="A5TJRDIT.B7000001"/>
   <Column Id="A5TJRDIT.B7000002"/>
   </Columns>
   <Indexes/>
   </PhysicalTable>
   <PhysicalTable Id="A5TJRDIT.B2000002" Name="AUTO">
   <Columns SEARCH="Column[Not(@SASColumnType='N')]">
   <Column Id="A5TJRDIT.B7000003"/>
   </Columns>
   <Indexes/>
   </PhysicalTable >
   <PhysicalTable Id="A5TJRDIT.B2000003" Name="CENSUS">
   <Columns SEARCH="Column[Not(@SASColumnType='N')]">
   <Column Id="A5TJRDIT.B700000A" SASColumnName="State" SASColumnType="C"
SASColumnLength="14">
   <Indexes >
   <Index Id="A5TJRDIT.BI000001" IndexName="State" IsClustered="0"
IsNoMiss="0" IsUnique="1"/>
   </Indexes>
   </Column>
   <Column Id="A5TJRDIT.B700000B"/>
   </Columns>
```

```
   <Indexes>
   <Index Id="A5TJRDIT.BI000001"/>
   </Indexes>
   </PhysicalTable>
   <PhysicalTable Id="A5TJRDIT.B2000004" Name="CENSUS1990">
   <Columns SEARCH="Column[Not(@SASColumnType='N')]">
   <Column Id="A5TJRDIT.B700000E"/>
   <Column Id="A5TJRDIT.B700000F"/>
   </Columns>
   <Indexes/>
   </PhysicalTable>
 <PhysicalTable Id="A5TJRDIT.B200000K" Name="EMPLOYEE">
   <Columns SEARCH="Column[Not(@SASColumnType='N')]">
   <Column Id="A5TJRDIT.B700002R" SASColumnName="EMPNO" SASColumnType="C"
SASColumnLength="5">
   <Indexes>
   <Index Id="A5TJRDIT.BI000006" IndexName="EMPNO" IsClustered="0" IsNoMiss="0"
IsUnique="1"/>
   </Indexes>
   </Column>
   <Column Id="A5TJRDIT.B700002S"/>
   <Column Id="A5TJRDIT.B700002T"/>
   <Column Id="A5TJRDIT.B700002U"/>
   </Columns>
   <Indexes>
   <Index Id="A5TJRDIT.BI000006"/>
   </Indexes>
   </PhysicalTable>
 <PhysicalTable Id="A5TJRDIT.B200000J" Name="EMPINFO">
   <Columns SEARCH="Column[Not(@SASColumnType='N')]">
   <Column Id="A5TJRDIT.B700002A"/>
   <Column Id="A5TJRDIT.B700002B"/>
  </Columns>
   <Indexes>
   <Index Id="A5TJRDIT.BI000003"/>
   </Indexes>
   </PhysicalTable>
 </Objects>
<Ns>SAS</Ns>
<Flags>260</Flags>
<Options>
<Templates>
<Template TemplateName="PhysicalTable">
<PhysicalTable Name="">
<Columns Search="Column[Not(@SASColumnType=''N'')]">
<Column Match="Column[Indexes/Index]" TemplateName="column"/>
<Column TemplateExpand="No"/>
</Columns>
<Indexes/>
</PhysicalTable>
</Template>
<Template TemplateName="column">
<Column SASColumnName="" SASColumnType="" SASColumnLength="">
 <Indexes>
  <Index TemplateName="index"/>
 </Indexes>
```

```
</Column>
</Template>
<Template TemplateName="index">
<Index IndexName="" IsClustered="" IsNoMiss="" IsUnique=""/>
</Template>
</Templates>
</Options>
</GetMetadataObjects>
```

The results return information about six tables. Only two of the tables have character columns that have indexes. Column objects for which only Id values are shown are character columns that do not have indexes. Indexes for which only Id values are shown indicate an index that is associated with the table, rather than defined on a character column.

Templates and the OMI_INCLUDE_SUBTYPES Flag

In both the legacy form and the new form, the order of the additional templates that request associations of associated objects is not important, unless the OMI_INCLUDE_SUBTYPES (16) flag is set.

When processing a metadata object, the SAS Metadata Server searches the metadata types listed in the <TEMPLATES> element or the <TEMPLATE> subelement to find a type match for that object. If a match is found, that metadata property string determines the specified attributes and associations to return. If the OMI_INCLUDE_SUBTYPES flag is not specified, an exact type match is required. This exact-match requirement makes the order of the metadata property strings in the template unimportant.

When the OMI_INCLUDE_SUBTYPES flag is specified, the server searches the metadata types just as before, but the conditions for the type match are different. With OMI_INCLUDE_SUBTYPES, the object being processed matches a metadata property string if the object type exactly matches, or if it is a subtype of a metadata property string's type. Once either type of match is made, the search stops. Any remaining matching property strings are ignored. Therefore, if there are metadata property strings for any supertype in a template, they should be listed after any metadata property strings that are subtypes of that supertype.

The OMI_INCLUDE_SUBTYPES flag does not affect selecting named templates. A named template is always a direct match.

Templates and the OMI_NOEXPAND_DUPS Flag

The SAS Metadata Model defines two association names for every relationship in the SAS Metadata Model. For example, a table object has a Columns association with its Column objects, and a Column object has a Table association with its table. When using templates to expand the associations of associated objects, avoid specifying both sides of a relationship in the template. The server has logic that prevents visiting the same object more than once when a two-sided reference occurs. However, when multiple associations refer to the same metadata types, the server can revisit those same objects over and over again.

This can happen easily in templates that expand objects of the PhysicalTable, Column, and ForeignKey metadata types, which are related to each other through the Columns/Table, ForeignKeys/Table, and Keys/KeyedColumns associations.

To prevent objects from being revisited, you can set the OMI_NOEXPAND_DUPS (524288) flag. However, this flag has memory overhead associated with it, so it is better to avoid specifying two-sided references.

An example of the incorrect way to expand PhysicalTable, Column, and ForeignKey objects is the following:

```
<PhysicalTable>
   <Columns/>
   <Keys/>
</PhysicalTable>
<Column>
  <Table/>
  <ForeignKeys/>
</Column>
<ForeignKey>
  <Table/>
  <KeyedColumns/>
</ForeignKey>
```

The correct way to expand them is the following:

```
<PhysicalTable>
   <Columns/>
</PhysicalTable>
<Column>
 <ForeignKey/>
</Column>
```

16

Getting All Metadata of a Specified Metadata Type

Introduction to the GetMetadataObjects Method

To get all metadata objects of a specified metadata type, the SAS Open Metadata Interface provides the GetMetadataObjects method. The default behavior of the GetMetadataObjects method is to get general, identifying information for each object of the metadata type specified in the TYPE parameter from the repository specified in the REPOSID parameter. The method supports flags and options that enable you to expand the request to get additional properties for each object, to search additional repositories, and to filter the objects that are returned by the request.

The following is an example of a GetMetadataObjects request that does not contain flags or options. The request gets a list of all objects of the PhysicalTable metadata type in Test repository 1, and their Id and Name attributes. The method call is formatted for the INMETADATA parameter of the DoRequest method.

```
<GetMetadataObjects>
<!--Reposid specifies Test repository 1 -->
 <Reposid>A0000001.A53TPPVI</Reposid>
 <Type>PhysicalTable</Type>
 <Objects/>
 <NS>SAS</NS>
 <Flags>0</Flags>
 <Options/>
</GetMetadataObjects>
```

In the request, note the following:

- The <REPOSID> element specifies the repository from which to get the objects.
- The <TYPE> element specifies the metadata type whose objects you want to list.
- The <NS> element specifies the namespace.
- The <FLAGS> and <OPTIONS> elements, although blank in this request, support flags and additional XML elements that expand or filter the GetMetadataObjects request.
- The <OBJECTS> element is an output parameter. Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"/>
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates"/>
</Objects>
```

  Test repository 1 has two objects of metadata type PhysicalTable defined.

A GetMetadataObjects request can be expanded to get additional attributes, to get associated objects, to include subtypes, and to get objects from additional repositories.

A GetMetadataObjects request can be filtered to get only objects that have specific attributes and associations. You can also filter the associated objects that are returned in a request.

For more information, see the following:

- “Expanding a GetMetadataObjects Request to Return Additional Properties” on page 357
- “Expanding a GetMetadataObjects Request to Include Subtypes” on page 365
- “Expanding a GetMetadataObjects Request to Include Additional Repositories” on page 366
- Chapter 17, “Filtering a GetMetadataObjects Request,” on page 371
- “Filtering the Associated Objects That Are Returned by a GetMetadataObjects Request” on page 389

The SAS type dictionary identifies which metadata type to use to list metadata describing resources and information assets that are common to or shared by applications. For more information about the SAS type dictionary, see Chapter 3, “Using Interfaces That Read and Write Metadata in SAS 9.4,” on page 19.

The GetMetadataObjects method is typically used to list application objects in a repository. But, it can also be used to list repositories. For more information, see “Using GetMetadataObjects to List Repositories” on page 369.

Expanding a GetMetadataObjects Request to Return Additional Properties

Introduction to the OMI_GET_METADATA Flag

You can expand a GetMetadataObjects method call to get additional properties by setting the OMI_GET_METADATA (256) flag and specifying flags defined for the GetMetadata method in the GetMetadataObjects request.

The OMI_GET_METADATA flag issues a GetMetadata request for each metadata object that is returned by the GetMetadataObjects method. Specifying one or more other GetMetadata flags with OMI_GET_METADATA enables you to get all properties or specific categories of properties for each metadata object.

The GetMetadataObjects method supports the following GetMetadata flags for requesting additional properties:

- OMI_ALL (1)—Gets all of the attributes and associations of the specified object, and general, identifying information about any associated objects. For more information, see “Example of Retrieving All Properties for All Objects” on page 358.
- OMI_SUCCINCT (2048)—Omits all properties that do not contain a value or that contain a null value from the output. For more information, see “Suppressing Properties That Do Not Store Values from GetMetadataObjects Output” on page 361.
- OMI_ALL_SIMPLE (8)—Gets all of the attributes of the specified object and any associated objects requested by other flags. For more information, see “Example of Retrieving Only Attributes of Objects” on page 361.
- OMI_TEMPLATE (4) — Instructs the SAS Metadata Server to check the <OPTIONS> element for user-defined templates that define which metadata properties to return. The templates can request additional properties for the specified metadata objects, as well as attributes and associations for associated metadata objects. Templates are specified in a <TEMPLATES> element. For more information, see “Example of Retrieving Specified Attributes of All Objects” on page 363. Also see “Example of Retrieving Associated Objects for All Objects” on page 364.

Specifying OMI_GET_METADATA and Other GetMetadata Flags

To specify a GetMetadata flag in a GetMetadataObjects request, add the flag's value to the OMI_GET_METADATA flag and to any other GetMetadataObjects flags that you have set. For example, if OMI_XMLSELECT (128) is already set, and you want to specify OMI_GET_METADATA (256) and OMI_ALL_SIMPLE (8) to get all of the attributes of each object, add their values together (128+256+8=392) and specify the sum in the FLAGS parameter.

Combining GetMetadata and GetMetadataObjects Flags

The flags in this section can be combined with other GetMetadataObjects flags.

- When GetMetadata flags are used with the OMI_INCLUDE_SUBTYPES (16) flag, the GetMetadataObjects method gets the specified properties for all subtypes of the specified metadata type, in addition to all objects of the specified metadata type.
- When GetMetadata flags are used with the OMI_XMLSELECT (128) flag, the GetMetadataObjects method gets the specified properties only for metadata objects that meet <XMLSELECT> search criteria.
- When GetMetadata flags are used with the OMI_DEPENDENCY_USES (8192) flag, the GetMetadataObjects method gets the specified properties for objects of the specified metadata type in all public repositories (the foundation repository and all custom repositories). When GetMetadata flags are used with the OMI_DEPENDENCY_USED_BY (16384) flag, the GetMetadataObjects method gets the specified properties for objects of the specified metadata type in the current repository and all project repositories.

Example of Retrieving All Properties for All Objects

The following is an example of a GetMetadataObjects request that sets the OMI_GET_METADATA (256) and OMI_ALL (1) flags. The OMI_ALL flag lists all attributes and associations for all PhysicalTable objects returned by the GetMetadataObjects request.

```
 <GetMetadataObjects>
<!-- Reposid parameter specifies Test repository 1 -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
   <!-- Specify OMI_GET_METADATA(256) + OMI_ALL (1) flags -->
   <Flags>257</Flags>
   <Options/>
</GetMetadataObjects>
```

In the request, note the following:

- The <REPOSID> element specifies to issue the request in Test repository 1.
- The <Type> element specifies to get all objects of the PhysicalTable metadata type.
- The <FLAGS> element specifies a number representing the sum of the OMI_GET_METADATA and OMI_ALL flags.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices" DBMSType=""
  Desc="Sales offices in NW region" IsCompressed="0" IsEncrypted="0"
  LockedBy="" MemberType="" MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" NumRows="-1" SASTableName=""
TableName="">
<AccessControls/>
<Aggregations/>
<AnalyticTables/>
<Changes/>
<Columns>
<Column Id="A53TPPVI.A5000001" Name="City" Desc="City of Sales Office"/>
<Column Id="A53TPPVI.A5000002" Name="Address" Desc="Street Address of Sales
Office"/>
<Column Id="A53TPPVI.A5000003" Name="Manager" Desc="Name of Operations
Manager"/>
<Column Id="A53TPPVI.A5000004" Name="Employees" Desc="Number of employees"/>
</Columns>
<Documents/>
<Extensions/>
<ExternalIdentities/>
<ForeignKeys/>
<Groups/>
<Implementors/>
<Indexes/>
<Keywords/>
<ModelResults/>
<Notes/>
<PrimaryPropertyGroup/>
<Properties/>
<PropertySets/>
<ReachThruCubes/>
<ResponsibleParties/>
<Roles/>
<SASPasswords/>
<SourceClassifierMaps/>
<SourceTransformations/>
<SpecSourceTransformations/>
<SpecTargetTransformations/>
<TablePackage/>
<TargetClassifierMaps/>
<TargetTransformations/>
<Timestamps/>
<TrainedModelResults/>
<Trees/>
```

```
<UniqueKeys/>
<UsedByPrototypes/>
<UsingPrototype/>
</PhysicalTable>
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates" DBMSType=""
  Desc="Sales associates in NW region" IsCompressed="0" IsEncrypted="0"
  LockedBy="" MemberType="" MetadataCreated="05Feb2002:09:50:56"
  MetadataUpdated="05Feb2002:09:50:56" NumRows="-1" SASTableName=""
TableName="">
<AccessControls/>
<Aggregations/>
<AnalyticTables/>
<Changes/>
<Columns>
<Column Id="A53TPPVI.A5000005" Name="Name" Desc="Name of employee"/>
<Column Id="A53TPPVI.A5000006" Name="Address" Desc="Home Address"/>
<Column Id="A53TPPVI.A5000007" Name="Title" Desc="Job grade"/>
</Columns>
<Documents/>
<Extensions/>
<ExternalIdentities/>
<ForeignKeys/>
<Groups/>
<Implementors/>
<Indexes/>
<Keywords/>
<ModelResults/>
<Notes/>
<PrimaryPropertyGroup/>
<Properties/>
<PropertySets/>
<ReachThruCubes/>
<ResponsibleParties/>
<Roles/>
<SASPasswords/>
<SourceClassifierMaps/>
<SourceTransformations/>
<SpecSourceTransformations/>
<SpecTargetTransformations/>
<TablePackage/>
<TargetClassifierMaps/>
<TargetTransformations/>
<Timestamps/>
<TrainedModelResults/>
<Trees/>
<UniqueKeys/>
<UsedByPrototypes/>
<UsingPrototype/>
</PhysicalTable>
</Objects>
```

The OMI_ALL flag gets all of the attributes and associations for each object, including attributes and associations for which no value has been defined. This is useful when you want to get both actual and potential properties for all of the objects.

Suppressing Properties That Do Not Store Values from GetMetadataObjects Output

To limit a GetMetadataObjects request to get only properties that have values defined, set the OMI_SUCCINCT (2048) flag. Here is an example of the output of the previous GetMetadataObjects request when OMI_SUCCINCT is set:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"
  Desc="Sales offices in NW region" IsCompressed="0" IsEncrypted="0"
  MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" NumRows="-1">
<Columns>
<Column Id="A53TPPVI.A5000001" Name="City" Desc="City of Sales Office"/>
<Column Id="A53TPPVI.A5000002" Name="Address" Desc="Street Address of Sales
Office"/>
<Column Id="A53TPPVI.A5000003" Name="Manager" Desc="Name of Operations
Manager"/>
<Column Id="A53TPPVI.A5000004" Name="Employees" Desc="Number of employees"/>
</Columns>
</PhysicalTable>
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates"
  Desc="Sales associates in NW region" IsCompressed="0" IsEncrypted="0"
  MetadataCreated="05Feb2002:09:50:56" MetadataUpdated="05Feb2002:09:50:56"
  NumRows="-1">
<Columns>
<Column Id="A53TPPVI.A5000005" Name="Name" Desc="Name of employee"/>
<Column Id="A53TPPVI.A5000006" Name="Address" Desc="Home Address"/>
<Column Id="A53TPPVI.A5000007" Name="Title" Desc="Job grade"/>
</Columns>
</PhysicalTable>
</Objects>
```

Example of Retrieving Only Attributes of Objects

The following is an example of a GetMetadataObjects request that sets the OMI_GET_METADATA (256), OMI_ALL_SIMPLE (8), and OMI_SUCCINCT (2048) flags. When OMI_ALL_SIMPLE is set in a GetMetadataObjects request, the flag instructs the method to get only the attribute values of the returned objects.

This request specifies to get all attributes of all Column objects in Test repository 1:

```
<GetMetadataObjects>
<!-- Reposid parameter specifies Test repository 1 -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>Column</Type>
   <Objects/>
   <NS>SAS</NS>
   <!-- Specify OMI_GET_METADATA (256) + OMI_ALL_SIMPLE (8)
```

```
          + OMI_SUCCINCT (2048) flags -->
    <Flags>2312</Flags>
    <Options/>
</GetMetadataObjects>
```

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<Column Id="A53TPPVI.A5000001" Name="City" BeginPosition="0"
ColumnLength="32"
  ColumnName="City" ColumnType="12" Desc="City of Sales Office" EndPosition="0"
  IsDiscrete="0" IsNullable="0" MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" SASColumnLength="32" SASColumnName="City"

  SASColumnType="C" SASExtendedLength="0" SASFormat="$Char32."
SASInformat="$32."
  SASPrecision="0" SASScale="0"/>
<Column Id="A53TPPVI.A5000002" Name="Address" BeginPosition="0"
ColumnLength="32"
  ColumnName="Address" ColumnType="12" Desc="Street Address of Sales Office"
  EndPosition="0" IsDiscrete="0" IsNullable="0"
MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" SASColumnLength="32"
  SASColumnName="Street_Address" SASColumnType="C" SASExtendedLength="0"
  SASFormat="$Char32." SASInformat="$32." SASPrecision="0" SASScale="0"/>
<Column Id="A53TPPVI.A5000003" Name="Manager" BeginPosition="0"
ColumnLength="32"
  ColumnName="Manager" ColumnType="12" Desc="Name of Operations Manager"
  EndPosition="0" IsDiscrete="0" IsNullable="0"
MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" SASColumnLength="32"
SASColumnName="Manager"
  SASColumnType="C" SASExtendedLength="0" SASFormat="$Char32."
SASInformat="$32."
  SASPrecision="0" SASScale="0"/>
<Column Id="A53TPPVI.A5000004" Name="Employees" BeginPosition="0"
ColumnLength="3"
  ColumnName="Employees" ColumnType="6" Desc="Number of employees"
EndPosition="0"
  IsDiscrete="0" IsNullable="0" MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" SASColumnLength="3"
SASColumnName="Employees"
  SASColumnType="N" SASExtendedLength="0" SASFormat="3.2" SASInformat="3.2"
  SASPrecision="0" SASScale="0"/>
<Column Id="A53TPPVI.A5000005" Name="Name" BeginPosition="0"
ColumnLength="32"
  ColumnName="Employee_Name" ColumnType="12" Desc="Name of employee"
EndPosition="0"
  IsDiscrete="0" IsNullable="0" MetadataCreated="05Feb2002:09:50:56"
  MetadataUpdated="05Feb2002:09:50:56" SASColumnLength="32"
SASColumnName="Employee"
  SASColumnType="C" SASExtendedLength="0" SASFormat="$Char32."
SASInformat="$32."
  SASPrecision="0" SASScale="0"/>
<Column Id="A53TPPVI.A5000006" Name="Address" BeginPosition="0"
```

```
ColumnLength="32"
  ColumnName="Employee_Address" ColumnType="12" Desc="Home Address"
EndPosition="0"
  IsDiscrete="0" IsNullable="0" MetadataCreated="05Feb2002:09:50:56"
  MetadataUpdated="05Feb2002:09:50:56" SASColumnLength="32"
SASColumnName="Home_Address"
  SASColumnType="C" SASExtendedLength="0" SASFormat="$Char32."
SASInformat="$32."
  SASPrecision="0" SASScale="0"/>
<Column Id="A53TPPVI.A5000007" Name="Title" BeginPosition="0"
ColumnLength="32"
  ColumnName="Title" ColumnType="12" Desc="Job grade" EndPosition="0"
IsDiscrete="0"
  IsNullable="0" MetadataCreated="05Feb2002:09:50:56"
  MetadataUpdated="05Feb2002:09:50:56" SASColumnLength="32"
SASColumnName="Title"
  SASColumnType="C" SASExtendedLength="0" SASFormat="$Char32."
SASInformat="$32."
  SASPrecision="0" SASScale="0"/>
</Objects>
```

Example of Retrieving Specified Attributes of All Objects

The following is an example of a GetMetadataObjects request that gets specified attributes of all objects of the specified metadata type. The GetMetadataObjects request sets the OMI_GET_METADATA (256) and OMI_TEMPLATE (4) flags and submits a template that specifies which attributes to get in a <TEMPLATES> element within the <OPTIONS> element.

```
<GetMetadataObjects>
<!-- Reposid parameter specifies Test repository 1 -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
   <!-- Specify OMI_GET_METADATA(256) + OMI_TEMPLATE (4) flags -->
   <Flags>260</Flags>
   <Options>
    <Templates>
      <PhysicalTable DBMSType="" IsCompressed="" IsEncrypted=""
        MemberType=""/>
       </Templates>
   </Options>
</GetMetadataObjects>
```

In the request, the template specifies to get the DBMSType, IsCompressed, IsEncrypted, and MemberType attributes for each of the PhysicalTable objects in repository A53TPPVI. Here is an example of the output from the request:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices" DBMSType=""
  IsCompressed="0"  IsEncrypted="0" MemberType=""/>
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates" DBMSType=""
```

```
     IsCompressed="0"  IsEncrypted="0" MemberType=""/>
   </Objects>
```

The SAS Metadata Server gets the requested properties and the Id and Name attributes that are returned by default.

For information about how to create a template, see "Understanding Templates" on page 337.

Example of Retrieving Associated Objects for All Objects

The following is an example of a GetMetadataObjects request that uses a template to retrieve associated objects of the specified metadata type. The GetMetadataObjects request sets the OMI_GET_METADATA (256) and OMI_TEMPLATE (4) flags and submits a template that specifies which associations to get in a <TEMPLATES> element in the <OPTIONS> element. The template specifies the metadata type and the association name for which associated objects should be returned.

```
<GetMetadataObjects>
<!-- Reposid parameter specifies Test repository 1 -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
   <!-- Specify OMI_GET_METADATA(256) + OMI_TEMPLATE (4) flags -->
   <Flags>260</Flags>
   <Options>
    <Templates>
      <PhysicalTable>
          <Columns/>
          <Extensions/>
          <Indexes/>
      </PhysicalTable>
    </Templates>
   </Options>
</GetMetadataObjects>
```

In the request, the template specifies to get objects that are associated with the requested PhysicalTable objects through the Columns, Extensions, and Indexes association names.

Here is an example of the output from the request:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices">
     <Columns>
         <Column Id="A53TPPVI.B7000001"/>
         <Column Id="A53TPPVI.B7000002"/>
         <Column Id="A53TPPVI.B7000003"/>
         <Column Id="A53TPPVI.B7000004"/>
      </Columns>
      <Extensions/>
      <Indexes/>
```

```
</PhysicalTable>
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates">
     <Columns>
         <Column Id="A53TPPVI.B7000005"/>
         <Column Id="A53TPPVI.B7000006"/>
         <Column Id="A53TPPVI.B7000007"/>
         <Column Id="A53TPPVI.B7000008"/>
      </Columns>
      <Extensions/>
      <Indexes/>
</Objects>
```

In this example, the returned PhysicalTable objects have associated Column objects. But, they do not have associated objects through the Extensions and Indexes association names.

In the request, note the following:

- By default, the GetMetadataObjects method returns only the Id value of associated objects. To get additional attributes, you must set a flag, such as OMI_ALL_SIMPLE (8), to get all attributes for the specified object and the associated objects. Or, you can include additional templates that request specific attributes of the associated objects.
- When an association name is specified in a template (<Columns/>, <Extensions/>, and <Indexes/> in the previous example), the GetMetadataObjects method gets associated objects of all metadata types that are valid for the specified association name. This example does not show objects of these associated metadata types because no objects of the additional metadata types were found. However, the Columns association name supports associations to two metadata types: Column and ColumnRange. The Extensions association name supports associations to two metadata types: Extension and NumericExtension. The Indexes association name has one valid metadata type: Index. This GetMetadataObjects request could have retrieved associated objects of all of these metadata types.

The GetMetadataObjects method also supports search criteria that enable you to filter the associated objects that are retrieved. For more information, see “Filtering the Associated Objects That Are Returned by a GetMetadataObjects Request” on page 389.

Expanding a GetMetadataObjects Request to Include Subtypes

The GetMetadataObjects method supports the OMI_INCLUDE_SUBTYPES (16) flag to enable you to list subtypes of the metadata type specified in the <TYPE> element. A subtype is a metadata type that inherits properties from a supertype. A supertype can have many subtypes. You can view the supertype and subtype relationships defined in the SAS Metadata Model in the “Hierarchical Listing of SAS Namespace Metadata Types” in the SAS Metadata Model: Reference.

When OMI_INCLUDE_SUBTYPES is set, the GetMetadataObjects method gets all objects of all subtypes of the specified metadata type, in addition to all objects of the specified metadata type. This enables you to avoid querying for each subtype. If you

want to get information about some subtypes, but not others, use the hierarchical listing to assess the hierarchical level at which to target your request.

The following is an example of a GetMetadataObjects request that sets OMI_INCLUDE_SUBTYPES and specifies to get all subtypes of supertype DataTable:

```
<GetMetadataObjects>
<! -- Reposid parameter specifies Test repository 1 -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>DataTable</Type>
   <Objects/>
   <NS>SAS</NS>
   <! -- Specify OMI_INCLUDE_SUBTYPES (16) flag -->
   <Flags>16</Flags>
   <Options/>
</GetMetadataObjects>
```

The DataTable supertype has the following subtypes defined for it in the SAS Metadata Model: ExternalTable, PhysicalTable, QueryTable, RelationalTable, TableCollection, and WorkTable. OMI_INCLUDE_SUBTYPES gets all objects of these subtypes that are defined in Test repository 1.

Here is an example of the output returned by the SAS Metadata Server:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"/>
<PhysicalTable Id="A53TPPVI.A4000002" Name="Sales Associates"/>
</Objects>
```

Test repository 1 has two objects of subtype PhysicalTable and no objects of the other subtypes.

The default behavior of the GetMetadataObjects method is to get the Id and Name values for all objects that are found. When OMI_INCLUDE_SUBTYPES is set with OMI_GET_METADATA (256) and GetMetadata flags, the GetMetadataObjects method gets the requested properties for all subtype objects. For more information, see “Expanding a GetMetadataObjects Request to Return Additional Properties” on page 357.

Expanding a GetMetadataObjects Request to Include Additional Repositories

Flag Behavior

The GetMetadataObjects method supports the OMI_DEPENDENCY_USES (8192) and OMI_DEPENDENCY_USED_BY (16384) flags to enable you to get objects from other repositories.

- Set OMI_DEPENDENCY_USES to include objects from all public repositories in the method results. The foundation repository and all custom repositories are public repositories.
- Set OMI_DEPENDENCY_USED_BY to include objects from all private repositories in the method results. Project repositories are considered to be private repositories.
- To get objects from all repositories on the SAS Metadata Server, set both flags.

Example of a GetMetadataObjects Request That Includes All Public Repositories

The following is an example of a GetMetadataObjects request that sets the OMI_DEPENDENCY_USES (8192) flag to get all public repositories:

```
<GetMetadataObjects>
<!-- Reposid parameter specifies host repository -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
 <!-- Specify OMI_DEPENDENCY_USES (8192) flag -->
   <Flags>8192</Flags>
   <Options/>
</GetMetadataObjects>
```

This request returns all objects of the PhysicalTable metadata type from the specified repository and all other public repositories.

In the request, note the following:

- The <REPOSID> element specifies a repository to search. When the OMI_DEPENDENCY_USES flag is set, specifying a value for the <REPOSID> element is optional. When a <REPOSID> value is omitted, the method gets objects of the specified metadata type first from the foundation repository. Then, it returns objects from custom repositories in the order in which they were registered.

  When a repository identifier is specified in the <REPOSID> element, the SAS Metadata Server gets objects from the specified repository first, before it gets objects from the foundation repository and custom repositories. The specified repository can be the foundation repository, a custom repository, or a project repository.
- The <TYPE> element specifies the metadata type of the objects to list.
- The OMI_DEPENDENCY_USES flag is expressed as a numeric value in the <FLAGS> element.
- Output is returned in the <OBJECTS> element.

Example of a GetMetadataObjects Request That Includes All Project Repositories

A GetMetadataObjects request can be expanded to include objects from all project repositories by setting the OMI_DEPENDENCY_USED_BY (16384) flag. The following is an example of a GetMetadataObjects request that sets the OMI_DEPENDENCY_USED_BY (16384) flag:

```
<GetMetadataObjects>
<!-- Reposid parameter specifies host repository -->
   <Reposid>A0000001.A53TPPVI</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <NS>SAS</NS>
 <!-- Specify OMI_DEPENDENCY_USED_BY (16384) flag -->
   <Flags>16384</Flags>
   <Options/>
</GetMetadataObjects>
```

In the request, note the following:

- The <REPOSID> element specifies a repository to search. When the OMI_DEPENDENCY_USED_BY flag is set, specifying a value for the <REPOSID> element is optional. When is <REPOSID> value is omitted, the method gets objects of the specified metadata type from all project repositories in the order in which the repositories were registered. When a repository identifier is specified in the <REPOSID> element, the SAS Metadata Server gets objects from the specified repository first, and then gets objects from the project repositories. The specified repository can be the foundation repository, a custom repository, or a project repository.
- The <TYPE> element specifies the metadata type of the objects to list.
- The OMI_DEPENDENCY_USED_BY flag is expressed as a numeric value in the <FLAGS> element.
- Output is returned in the <OBJECTS> element.

Example of a GetMetadataObjects Request That Includes All Repositories

To get objects from all repositories that are registered on the SAS Metadata Server (foundation, custom, and project), set both the OMI_DEPENDENCY_USES (8192) and OMI_DEPENDENCY_USED_BY (16384) flags.

The following is an example of a GetMetadataObjects request that gets objects of metadata type PhysicalTable from all repositories:

```
   <GetMetadataObjects>
   <!-- Reposid parameter specifies host repository -->
      <Reposid></Reposid>
      <Type>PhysicalTable</Type>
      <Objects/>
```

```
   <NS>SAS</NS>
  <!-- Specify OMI_DEPENDENCY_USES (8192) and
OMI_DEPENDENCY_USED_BY (16384) flags -->
   <Flags>24576</Flags>
   <Options/>
</GetMetadataObjects>
```

In the request, note the following:

- It is not necessary to specify a target repository in the <REPOSID> element. When the <REPOSID> element is blank, the method gets objects from all repositories, beginning with the foundation repository, then custom repositories, and then project repositories. Within each category, the repositories are listed in the order in which they were registered. When a repository identifier is specified in the <REPOSID> parameter, the SAS Metadata Server gets objects from that repository before it gets objects from other repositories. The specified repository can be the foundation repository, a custom repository, or a project repository.
- The <TYPE> element specifies the metadata type of the objects to list.
- The <FLAGS> element specifies the sum of the numeric values representing the OMI_DEPENDENCY_USES and OMI_DEPENDENCY_USED_BY flags (8192 + 16384 = 24576).
- Output is returned in the <OBJECTS> element.

Using GetMetadataObjects to List Repositories

The GetMetadataObjects method is typically issued in the SAS namespace to get all instances of a specified application metadata type. However, the method can also be issued in the REPOS namespace to get repositories.

The following is an example of a GetMetadataObjects request that gets repositories:

```
<GetMetadataObjects>
  <Reposid></Reposid>
  <Type>RepositoryBase</Type>
  <Objects/>
  <NS>REPOS</NS>
  <Flags>0</Flags>
  <Options/>
</GetMetadataObjects>
```

In the request, note the following:

- Specifying a value in the <REPOSID> element is optional. A GetMetadataObjects request that is issued in the REPOS namespace queries the SAS Repository Manager. Any other value entered in the <REPOSID> element is ignored and does not return an error.
- The <TYPE> element specifies the RepositoryBase metadata type. RepositoryBase is the valid value for listing repositories. Specifying a metadata type that describes an application metadata object in the REPOS namespace returns an error.
- The <NS> element specifies the REPOS namespace.

- The <FLAGS> and <OPTIONS> elements are blank. However, with the exception of the OMI_DEPENDENCY_USED_BY and OMI_DEPENDENCY_USES flags, flags and options that are supported in the SAS namespace can be specified in the REPOS namespace as well.
- Output is returned in the <OBJECTS> element.

17

Filtering a GetMetadataObjects Request

Overview of Filtering a GetMetadataObjects Request

The GetMetadataObjects method enables you to filter both the initial set of objects and the associated objects that are selected in the GetMetadataObjects request. This topic describes how to filter the initial set of objects selected by GetMetadataObjects. For information to filter the associated objects, see “Filtering the Associated Objects That Are Returned by a GetMetadataObjects Request” on page 389.

The GetMetadataObjects method supports an OMI_XMLSELECT (128) flag to enable you to filter the initial set of objects that are retrieved by the SAS Metadata Server. The OMI_XMLSELECT flag instructs the server to check the OPTIONS parameter for search criteria specified in an <XMLSELECT search="*criteria*"/> element. The search syntax supported in the <XMLSELECT search="*criteria*"/> element enables you to filter objects based on attribute criteria, association path criteria, and a combination of attribute and association path criteria.

The criteria can be concatenated with the logical operators AND, OR and NOT.

Attribute criteria enable you to select only objects that contain specified attribute values. For example, you can specify the following:

- select only Person objects that have a Name attribute value of John Doe.

By concatenating attribute criteria with the logical operators AND or OR, you can perform exclusive or inclusive filtering based on the attribute criteria. For example, you can specify the following:

- select objects that have a Name attribute value of John Doe or Jane Doe (exclusive search).
- select objects that have the attribute=value pairs Name="John Doe" and Title="Manager" (inclusive search).

Association path criteria enables you to select objects that have a specific association and whose associated objects meet association and attribute criteria. For example, you can specify the following:

- select Document objects that have a Reports association to a Report object.
- select Document objects that have a ResponsibleParties association to a ResponsibleParty object that has a Persons association to a Person object that has the Name attribute value of John Doe.

In both examples, the Document objects that are retrieved are filtered by one association path. In the first example, the filtering association path starts with the association Reports. Documents that do not have a Reports association are ignored. In the second example, the filtering association path starts with the association ResponsibleParties. Documents that do not have a ResponsibleParties association are ignored. In addition, the ResponsibleParty objects found through the ResponsibleParties association are filtered to include only objects that have a Persons association to a Person object that has the attribute value Name="John Doe".

When you concatenate association path criteria, filtering is based on two or more associations that are directly defined for the specified metadata type. The XMLSELECT search syntax supports explicit AND and OR logical operators in concatenated association path criteria. Concatenated association path criteria can be used to select objects that meet the following requirements:

- all of the criteria specified in all of the association paths
- the criteria in one association path or the criteria in another association path

For example, you can specify to select the following:

- Document objects that have a Reports association to a Report object that has a Name attribute value of Sales AND a ResponsibleParties association to a ResponsibleParty object that has a Persons association to a Person object that has a Name value of John Doe.
- Document objects that have a Reports association to a Report object that has a Name attribute value of Sales OR a ResponsibleParties association to a ResponsibleParty object that has a Persons association to a Person object that has a Name value of John Doe.

SAS also supports a NOT logical operator in attribute criteria and a NOT function in association path syntax. Instead of returning objects that match the specified attribute or association path criteria, the new operator and function return objects that do not match the specified criteria.

<XMLSELECT search="*criteria*"/> Syntax

General Syntax

The <XMLSELECT> element is specified in the OPTIONS parameter in the following form:

```
<XMLSELECT search="criteria"/>
```

The syntax of *criteria* varies depending on whether you are specifying attribute criteria, association path criteria, or both.

A statement that specifies only attribute criteria on the metadata type defined in the GetMetadataObjects TYPE parameter can be specified as one of the following:

```
AttributeCriteria

(AttributeCriteria)

Object[AttributeCriteria]
```

A statement that concatenates attribute criteria is specified as one of the following:

```
AttributeCriteria and|or  AttributeCriteria

(AttributeCriteria and|or
AttributeCriteria)

Object[AttributeCriteria  and|or
AttributeCriteria]
```

In a statement that specifies only attribute criteria, the brackets and parentheses around the criteria are optional. For all other syntax combinations, the brackets and parentheses must be specified as shown.

A statement that specifies both attribute criteria and an association path as criteria is specified as follows:

```
Object[AttributeCriteria][AssociationPath]
```

A statement that specifies only an association path as criteria is specified as follows:

```
Object[AssociationPath]
```

A statement that specifies an association path that has multiple association levels defined is specified as follows:

```
Object[AssociationPathLevel1/AssociationPathLevel2/AssociationPathLeveln]
```

A statement that concatenates association path criteria can be specified as follows:

```
Object[AssociationPath1][AssociationPath2][AssociationPath3]

Object[AssociationPath1] or [AssociationPath2] or [AssociationPath3]

Object[AssociationPath1] and [AssociationPath2] and [AssociationPath3]

Object[AssociationPath1] and [AssociationPath2] or [AssociationPath3]
```

The first example has an implied AND logical operator between association paths.

A statement that specifies the NOT logical operator in attribute criteria is specified as follows:

```
not(criteria)

Object[not(criteria)]
```

A statement that specifies the NOT function is specified as follows:

```
object[not(AssociationPath)]

object[not(AssociationPathLevel1/AssociationPathLevel2)]
```

Object Component

The *Object* component is required for all searches except simple attribute criteria searches. It specifies the object class type. Valid values are a metadata type name or an asterisk (*).

- The metadata type can be the same metadata type that is specified in the GetMetadataObjects TYPE parameter or a subtype of the metadata type. To determine the subtypes of a metadata type, see the metadata type descriptions in the SAS Metadata Model: Reference.
- An * (asterisk) is a shorthand method of referring to the metadata type specified in the TYPE parameter.

AttributeCriteria Component

The *AttributeCriteria* component is optional. It enables you to filter the objects that are selected to objects matching a specified attribute=value pair. The syntax of *AttributeCriteria* is as follows:

```
[@attrname cop 'value' lop AttributeCriteria]
```

- *@attrname* specifies an attribute name (for example, @Name or @Desc).
- *cop* is a comparison operator. The supported comparison operators include:

  =:
  Begins with the specified character string. Numeric and datetime values are not supported. MISSING character value is supported.

  ?, contains, or CONTAINS
  Contains the specified character string. Numeric and datetime values are not supported. MISSING character value is supported.

  =, eq, or EQ
  Equal to the specified character string, numeric, datetime, or MISSING value.

  ne, NE
  Not equal to the specified character string, numeric, datetime, or MISSING value.

  ge, GE
  Greater than or equal to the character string, numeric, or datetime value. MISSING value not supported.

  gt, GT
  Greater than the character string, numeric, or datetime value. MISSING value not supported.

  le, LE
  Less than or equal to the character string, numeric, or datetime value. MISSING value not supported.

  lt, LT
  Less than the character string, numeric, or datetime value. MISSING value not supported.

- '*value*' is a character or numeric string enclosed within single quotation marks.

  Character Strings:

  Searches of character strings compare the <XMLSELECT search="*criteria*"/> search pattern value with the attribute data value and determine which string appears first in a sorted list. The sort order is based on the collating sequence for the specified locale. If either the data or pattern values contain any characters that are not in the locale's collation list, the locale determines how to order the unknown characters.

  Missing Values:

  - To search for a MISSING numeric or datetime attribute value, specify a period enclosed within single quotation marks.
  - To search for a MISSING character attribute value, specify two adjacent single quotation marks.

"V" in an Attribute's Length Limit:

When evaluating attributes that have a "V**n**" in the Length limit, for example "V64", the search process evaluates only the **n** number of characters to make the selection. A "V**n**" in the Length limit indicates the property is arbitrarily large. The documented length (**n**) is the maximum length 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. The search process does not search these "overflow" objects.

Datetime Values:

In the current release, searches by date or by time are not supported. However, the SAS Metadata Server supports datetime queries on the MetadataCreated and MetadataUpdated attributes. The supported DATETIME formats are the following:

- ddmmmyyyy:hh:mm:ss.s
- ddmmmyyyy:hh:mm:ss
- a SAS date value that represents a ddmmmyyyy:hh:mm:ss value
- a MISSING datetime '.' value

The DATE format ddmmmyyyy is not supported.

Datetime queries are supported only in the standard interface. For more information about how to issue SAS Open Metadata Interface methods, see "Communicating with the SAS Metadata Server" on page 14.

Note: Objects are persisted to disk with a GMT datetime value. Therefore, 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 SAS Metadata Server converts values that you specify in a search string to GMT values for you. However, the datetime values returned by the server look different from the values that you submitted in the search string.

The following are examples of queries in the supported formats:

```
<XMLSELECT search="*[@MetadataCreated GT '27May2003:09:20:17.2']"/>
<XMLSELECT search="*[@MetadataCreated LT '27May2010:09:20:17']"/>
<XMLSELECT search="*[@MetadataCreated GT '1309907400']"/>
<XMLSELECT search="*[@MetadataUpdated EQ '.']"/>
```

In the third example, '1309907400' is the SAS date value for '30May2003:19:03:11' GMT.

- *lop* is the logical operator AND or OR. It enables you to specify an additional *AttributeCriteria* string that is appended to and concatenated with the first *AttributeCriteria* string. AND specifies that both conditions must be met in order for an object to be selected for retrieval. OR specifies that either condition can be met for an object to be selected. An example of an OR comparison is the following:

```
[@Name = 'John Doe' or @Name = 'Jane Doe']
```

Compound attribute criteria are also supported. Use parentheses to control evaluation order. For example:

```
search="*[@ProductName='SAS/CONNECT' and
```

```
(@Name contains 'test - SAS/CONNECT Server' or @Name='test')]"
```

In this example, the expression enclosed within the parenthesis is evaluated first.

Note: Whether single, concatenated, or compound attribute criteria are used, the attribute test is applied only if all of the specified attribute names are valid for the object. That is, if one of the attribute names in the attribute string is misspelled, then no objects are selected. If the OMI_INCLUDE_SUBTYPES flag is set with OMI_XMLSELECT, the metadata type and subtype objects to be tested might support a different set of attribute names. Only objects that contain all of the specified attribute names are tested for a match.

NOT Logical Operator in AttributeCriteria Component

The NOT logical operator returns object instances of the requested metadata type that do not have the specified value in the specified attribute. For example, the following search string returns Person objects with values other than 'Senior' in the Title attribute:

```
<XMLSelect search="Person[not(@Title ? 'Senior')]"/>
```

The NOT logical operator must be specified before the attribute criteria that it qualifies in the search string. And, the attribute criteria must be enclosed within parentheses. For example:

```
not(criteria)
```

```
object[not(criteria)]
```

NOT is supported in concatenated attribute criteria as follows:

```
not (criteria) and  not (criteria)
```

The preceding NOT (criteria) request specifies to omit object instances that meet the first criteria, and to omit object instances that meet the second criteria.

Valid use of the NOT logical operator for attribute criteria in an association path specification is as follows:

```
*[AssociationName/AssociatedObject[not(criteria)]]
```

The string specifies to return object instances that have the specified association, and that do not meet the specified attribute criteria.

You might want to use the NOT logical operator for attribute criteria in association paths to find object instances that do not meet the specified attribute criteria, and have the specified association and an associated object that meets the specified attribute criteria:

```
*[not(criteria)][AssociationName/AssociatedObject [criteria]]
```

For an example of how the NOT logical operator can be used in a GetMetadataObjects request, see “Examples of How the New Template Form Can Be Used” on page 347. The example uses the NOT logical operator in the Search attribute that is specified on an association name in a template to filter the associated objects that are retrieved.

For information about how to get objects that do not have a specified association, see “NOT Function in the AssociationPath Component” on page 383.

AssociationPath Component

The *AssociationPath* component enables you to specify one or more associations as search criteria. To be selected, the objects specified in the *Object* component must have an association that meets the criteria in *AssociationPath*.

The syntax of *AssociationPath* is as follows:

```
Object[AssociationPath] AND | OR [AssociationPathn]
```

Each *AssociationPath* is the following:

```
[AssociationPathLevel1/AssociationPathLevel2/AssociationPathLeveln]
```

And, *AssociationPathLevel* is the following:

```
AssociationName/AssociatedObject[AttributeCriteria]
```

In the syntax, *Object* can be a metadata type name or an asterisk. For more information, see "Object Component" on page 374.

Each *AssociationPath* specifies one association of *Object* to evaluate.

An *AssociationPath* specification can include one or many association path levels. The first *AssociationPathLevel* identifies the association of *Object* that will be evaluated. Each subsequent level filters the objects that are returned for this first association by specifying additional associations, associated metadata types, and attribute criteria that the first set of objects must match in order to be selected. For more information, see "Understanding How Association Paths Are Evaluated" on page 378. Each *AssociationPathLevel* within the *AssociationPath* is separated from the other levels by a slash (/).*AttributeCriteria* is optional in an *AssociationPathLevel*.

Understanding How Association Paths Are Evaluated

AssociationPathLevel

To understand how an association path is evaluated, we must consider each association path level that is specified. The *AssociationPathLevel* specifies an association name and an associated object that is evaluated, as well as optional attribute criteria that the associated objects must meet to be selected.

The first *AssociationPathLevel* in *AssociationPath* sets the context for the request. It specifies the association that an object must have defined to be evaluated. When considered in the context of *Object*, the syntax of the first *AssociationPathLevel* looks like the following:

```
Object[AssociationName/AssociatedObject[AttributeCriteria]]
```

Object can be the same metadata type name that is specified in the GetMetadataObjects TYPE parameter, a subtype of the metadata type in TYPE, or an asterisk, which defaults to the value in the TYPE parameter. The value that you specify in *Object* indicates what association names are valid. In the first *AssociationPathLevel*, the following is true:

- If *Object* is a metadata type, then *AssociationName* must be an association name that is valid for that metadata type as defined in the SAS Metadata Model. For example, if *Object* is Report, then *AssociationName* must be an association name that is defined for the Report metadata type in the SAS Metadata Model.
- If *Object* is an *, then *AssociationName* must be an association name that is valid for the metadata type specified in the TYPE parameter.

The *AssociatedObject* in the *AssociationPathLevel* specification can also be a metadata type name or an asterisk. However, in this position, the specified value stipulates whether associated objects of one metadata type should be evaluated, or, that associated objects of all of the potential associated metadata types defined for the association name should be evaluated. For example:

- When *AssociatedObject* is a metadata type, this says, “Give me only object instances of this metadata type that are related under the specified association name.”
- When *AssociatedObject* is an asterisk, this says “Give me object instances of all potential metadata types that are defined for the specified association name.”

Consider the following *AssociationPathLevel* specifications to understand how the asterisk and metadata type names are evaluated in the *Object* and *AssociatedObject* positions. For these examples, assume that Report is the metadata type specified in the TYPE parameter.

```
*[ReportLocation/*]
```

```
Report[ReportLocation/Email]
```

The first specification selects objects of the metadata type specified in the TYPE parameter (Report) that have a ReportLocation association and associated objects of any of the metadata types that are valid for the ReportLocation association name. The ReportLocation association name supports associations to objects of 19 metadata types.

The second specification selects Report objects that have a ReportLocation association to an Email object. (Email is one of the 19 supported associated metadata types.)

If a subtype were specified in either the *Object* or *AssociatedObject* positions, then the SAS Metadata Server would select only objects and associated objects of the specified subtype. Report is a subtype of the Classifier metadata type. If Classifier were the metadata type specified in the TYPE parameter, the first specification above would apply to the Classifier metadata type. The second specification would still only apply to Report objects.

The *AttributeCriteria* component in *AssociationPathLevel* further limits the *Objects* that are selected to objects whose associated objects meet the specified attribute criteria. For example, consider the following request:

```
Report[ReportLocation/Document[@TextType='XML']]
```

The attribute criteria limit the Report objects that are selected to objects that have associated Document objects that have the attribute TextType="XML". When attribute criteria are specified in a query that has an * in the *AssociatedObject* component, the attribute criteria are applied to all associated objects.

Subsequent *AssociationPathLevels* in an *AssociationPath* get their context from the *AssociatedObject* in the preceding level.

- When the preceding associated object is a metadata type, *AssociationName* must be an association name that is valid for that metadata type.
- When the preceding associated object is an *, *AssociationName* can be any association name defined for one of the metadata types supported by the preceding association name.

Consider the following *AssociationPath*. The *AssociationPathLevels* are separated by a / (slash):

```
Report[ReportLocation/Document[@TextType='XML']/AssociationName/AssociatedObject]
```

AssociationName must be an association that is valid for the Document metadata type. *AssociatedObject* must be a metadata type that is supported by *AssociationName* or an *.

Consider the following *AssociationPath*:

```
Report[ReportLocation/*/AssociationName/AssociatedObject]
```

AssociationName can be an association name that is valid for any of the 19 metadata types supported by the ReportLocation association. *AssociatedObject* must be a metadata type that is supported by *AssociationName* or an *.

The following is an example of an *AssociationPath* that specifies multiple *AssociationPathLevels* and specifies metadata types in the *AssociatedObject* positions:

```
Report[ResponsibleParties/ResponsibleParty/Persons/Person
/Locations/Location[@Area='New York']]
```

The request selects Report objects that have a ResponsibleParties association to a ResponsibleParty object that has a Persons association to a Person who has a Locations association to a Location object that has the attribute value Area="New York". It has three *AssociationPathLevels*:

1. ResponsibleParties/ResponsibleParty
2. Persons/Person
3. Locations/Location

The following is an example of an *AssociationPath* that specifies multiple *AssociationPathLevels* and specifies asterisks in the *AssociatedObject* positions:

```
Report[ResponsibleParties/*[@Role='OWNER']/Persons/*[@Name='John Doe']]
```

The request selects Report objects that have a ResponsibleParties association to any object that has a Role attribute value of Owner and a Persons association to any object that has a Name attribute value of John Doe. It has two *AssociationPathLevels*:

1. ResponsibleParties/*
2. Persons/*

Effect of OMI_INCLUDE_SUBTYPES Flag on an Association Path

Setting the OMI_INCLUDE_SUBTYPES (16) flag with OMI_XMLSELECT in the GetMetadataObjects method can drastically alter the results. When OMI_INCLUDE_SUBTYPES is set, the SAS Metadata Server applies the specified selection criteria to objects of the metadata types specified in *Object* and *AssociatedObject*, and to all of their subtypes.

If the metadata types have no subtypes defined for them in the SAS Metadata Model (neither Report nor Person have subtypes defined), the flag has no effect. However, some metadata types, such as Classifier, have many subtypes. The ability to get subtypes is useful when you want to get objects of similar metadata types that have common properties. Consider the following request:

```
Classifier[ResponsibleParties/*[@Role='OWNER']/Persons/*[@Name='John Doe']]
```

When OMI_INCLUDE_SUBTYPES is set, this request returns all Classifier objects, and objects of its subtypes Cube, Dimension, DataTable, ExternalTable, JoinTable, PhysicalTable, QueryTable, RelationalTable, Report, SharedDimension, TableCollection, and WorkTable objects that are owned by John Doe.

To determine what metadata types have subtypes, see the SAS Metadata Model: Reference.

Understanding How Concatenated Association Paths Are Evaluated

An <XMLSELECT search="*criteria*"/> search string that includes concatenated *AssociationPath* criteria must specify an association name that is valid for the primary *Object* component in each *AssociationPath*.

- If *Object* is a metadata type, then all association paths must begin with an association name that is valid for that metadata type.
- If *Object* is an *, then each association path must begin with an association name that is valid for the metadata type specified in the TYPE parameter.
- If *Object* is an * and the OMI_INCLUDE_SUBTYPES flag is set, then each association path must begin with an association name that is valid for the metadata type specified in the TYPE parameter or one of its subtypes.

The search syntax supports explicit AND and OR operators between *AssociationPath* components. If an operator is omitted, the *AssociationPath* components are joined by an implied AND operator. When the AND operator is specified or implied, only objects that meet the criteria in the combined *AssociationPath* components are selected.

The following table summarizes the evaluation algorithm used when multiple association paths are concatenated with the AND logical operator. If any association path criteria evaluates to False, then the concatenated association path criteria is False.

Search= "*	[path]	lop	[path]	lop	[path]	"=result
	[True]	and	[True]			=True
	[True]	and	[False]			=False
	[False]	and	[False]			=False
	[True]	and	[True]	and	[True]	=True
	[True]	and	[True]	and	[False]	=False
	[True]	and	[False]	and	[False]	=False
	[False]	and	[False]	and	[False]	=False

The following table summarizes the evaluation algorithm used when multiple association paths are concatenated with the OR logical operator. Only one association path criteria must be True in order for the concatenated association path criteria to be True.

search= "*	[path]	lop	[path]	lop	[path]	"=result
	[True]	or	[True]			=True
	[True]	or	[False]			=True
	[False]	or	[False]			=False
	[True]	or	[True]	or	[True]	=True
	[True]	or	[True]	or	[False]	=True
	[True]	or	[False]	or	[False]	=True
	[False]	or	[False]	or	[False]	=False

The following table summarizes the evaluation algorithm used when multiple association paths are concatenated with both the AND and OR logical operators. Paths that are separated by an AND logical operator are treated as if they have parenthesis around them. That is, they are read as a set, and they must both be true to evaluate as True. Otherwise, they evaluate to False. When OR is used between association paths, only one path must be True to return a True.

search= "*	[path1]	lop	[path2]	lop	[path3]	"=result
	[True]	and	[True]	or	[True]	True
	[True]	and	[True]	or	[False]	True

search="*	[path1]	lop	[path2]	lop	[path3]	"=result
	[True]	and	[False]	or	[True]	True
	[True]	or	[True]	and	[False]	True
	[False]	or	[True]	and	[False]	False
	[False]	or	[False]	and	[False]	False

When the OMI_INCLUDE_SUBTYPES flag is set, only objects that have all of the specified association names are tested for a match.

Example 1:

- Object 1 and Object 2 are subtypes of *. Object 1 has valid associations to associationname1 and associationname2. Object 2 has valid associations to associationname3 and associationname4.
- Query: `search="*[associationname1/object][associationname2/object]"`

  Result: Only Object 1 is returned.
- Query: `search="*[associationname3/object][associationname4/object]"`

  Result: Only Object 2 is returned.
- Query: `search="*[associationname1/object][associationname4/object]"`

  Result: Neither object is returned.

Example 2:

- Object 1 and Object 2 are subtypes of *. Object 1 has valid associations to associationname1, associationname2, and associationname3. Object 2 has valid associations to associationname2, associationname3, and associationname4.
- Query: `search="*[associationname2/object][associationname3/object]"`

  Result: Selects Object 1 and Object 2.

When the OR logical operator is specified, objects must meet the criteria in one of the specified *AssociationPath* components to be selected.

Example 3:

- Object 1 has a valid association to associationname1 and associationname2. Object 2 has a valid association to associationname2 and associationname3.
- Query: `search="*[associationname2/object] or [associationname3/object]"`

  Result: Selects Object 1 and Object 2.

NOT Function in the AssociationPath Component

The NOT function can be specified in the syntax to select all objects of the specified metadata type that do not have a specified *AssociationPath*.

A Search expression that specifies the NOT function takes the following form:

```
object[not(AssociationName/AssociatedObject)]
object[not(AssociationName/AssociatedObject[AttributeCriteria])]
object[not(AssociationPathLevel1/AssociationPathLeveln)]
```

The NOT syntax can be specified only for a complete *AssociationPath*. A syntax error is returned when the NOT function parameter does not contain the complete *AssociationPath*, as shown in these examples:

```
object[not(AssociationPathLevel)/AssociationPathLevel]
object[AssociationPathLevel/not(AssociationPathLevel)]
```

When more than one *AssociationPathLevel* is specified in an *AssociationPath*, it is better to use specific object types between each *AssociationPathLevel* instead of an * (asterisk). Using an asterisk between each *AssociationPathLevel* can cause misleading results. When * is specified between association path levels, some objects might be returned because the next association level is not valid for the associated object, rather than because an associated object is not found for the next level.

The NOT function can find incomplete or obsolete metadata. For example, the following specification returns PhysicalTable objects that do not have any Column objects defined:

```
search="PhysicalTable[not(Columns/Column)]"
```

The following request finds SASLibrary objects that are missing a UsingPackages association to a Directory or DatabaseSchema object:

```
search="SASLibrary[not(UsingPackages/Directory)][not(UsingPackages/DatabaseSchema)]"
```

The following request finds Property objects that do not have an association to an object. The use of the * in the next request is valid because the Property metadata type supports an AssociatedObject association to all metadata types.

```
search="Property[not(AssociatedObject/*)]"
```

However, a Property object that is returned by this request does not necessarily represent an orphaned Property object. A Property object can have a PropertySets association to a PropertySet, or a PropertyGroups association to a PropertyGroup, instead of being associated directly with an object.

The following is an example of a search string that specifies two *AssociationPathLevel* values:

```
search="Property[not(AssociatedObject/*/AssociatedFile/File)]"
```

This request is problematic. A Property object can be associated through the AssociatedObject association to all object types. But, not all object types can be associated to the next level AssociatedFile association. For example, a Cube object has a valid association to the AssociatedFile association, and a PhysicalTable object does not. When an object from the previous *AssociationPathLevel* is filtered by an incompatible association at the next level, no associated objects can be found.

For more meaningful output, here is a better way to code this request:

```
search="Property[not(AssociatedObject/Cube/AssociatedFile/File)]
   [not(AssociatedObject/PhysicalTable)]"
```

In this example, the first criteria specifies to return Property objects that are not associated to a Cube object through the AssociatedObject association and not associated to a File object through the AssociatedFile association. The second criteria specifies to return Property objects that are not associated to a PhysicalTable object through the AssociatedObject association. Because the criteria

are concatenated with an implied AND operator, an object must meet both criteria to be returned.

For an example of a GetMetadataObjects request that specifies the NOT function, see “Example of Getting Objects That Do Not Have a Specified Association” on page 392.

Sample Search Strings for Common Filters

Single Attribute Search on the Metadata Type in the TYPE Parameter

Both of the following <XMLSELECT search="*criteria*"/> elements select all objects that have a Name attribute value of John Doe:

```
<XMLSELECT search="*[@Name='John Doe']"/>
<XMLSELECT search="Person[@Name='John Doe']"/>
```

Single Attribute Search on a Subtype of the TYPE Parameter

The following <XMLSELECT search="*criteria*"/> element selects PhysicalTable objects that have a DBMSType attribute value of Oracle:

```
<Type>RelationalTable</Type>
...
<XMLSELECT search="PhysicalTable[@DBMSType='Oracle']"/>
```

Selecting Objects Whose Attributes Begin with a Value

The following <XMLSELECT search="*criteria*"/> element selects Person objects that have a Name value that begins with John:

```
<XMLSELECT search="Person[@Name =:'John']"/>
```

Selecting Objects Whose Attributes Have a Missing Value or Blank String

The following <XMLSELECT search="*criteria*"/> element selects WorkTable objects that have a missing numeric value in the NumRows attribute:

```
<XMLSELECT search="WorkTable[@NumRows='.']"/>
```

The following <XMLSELECT search="*criteria*"/> element selects WorkTable objects that have a blank string in the MemberType attribute:

```
<XMLSELECT search="WorkTable[@MemberType='']"/>
```

Specifying Concatenated Attributes

The following <XMLSELECT search="*criteria*"/> element selects objects that have either the Name attribute value of John Doe or Jane Doe:

```
<XMLSELECT search="*[@Name='John Doe' OR @Name='Jane Doe']"/>
```

The logical operator can be specified in uppercase or lowercase letters.

Searching by Association Name

The following <XMLSELECT search="*criteria*"/> element selects objects that have any objects associated with them through the ResponsibleParties association:

```
<XMLSELECT search="*[ResponsibleParties/*]"/>
```

Searching by Association Name and Attribute Criteria

The following <XMLSELECT search="*criteria*"/> element selects any objects that have a Role attribute value of OWNER associated with them through the ResponsibleParties association:

```
<XMLSELECT search="*[ResponsibleParties/*[@Role='OWNER']]"/>
```

Specifying Multiple Association Levels in an Association Path

The following <XMLSELECT search="*criteria*"/> element selects objects that have a Role attribute value of OWNER associated with them through the ResponsibleParties association. The ResponsibleParties association has a Persons association to a Person object that has a Name attribute value of John Doe.

```
<XMLSELECT search="*[ResponsibleParties/*[@Role='OWNER']/Persons/Person
[@Name='John Doe']]"/>
```

The following <XMLSELECT search="*criteria*"/> element selects objects owned by any type of object with a Name attribute value of John Doe:

```
<XMLSELECT search="*[ResponsibleParties/*[@Role='OWNER']/Persons/*[@Name='John
Doe']]"/>
```

This request is identical to the preceding request, except that an asterisk is substituted for the Person object to specify any object in the second path level. The Persons association supports associations to Person and IdentityGroup objects.

Specifying an asterisk in this position causes the SAS Metadata Server to evaluate IdentityGroup objects and Person objects in its search.

Specifying Concatenated Association Paths

The following <XMLSELECT search="*criteria*"/> element selects objects that have objects associated to them through the ResponsibleParties and Reports associations that meet the criteria specified for those association names.

```
<XMLSELECT search="*[ResponsibleParties/*[@Role='Owner']/Persons/
*[@Name='Joe Accountant']][Reports/Report[@Name='Sales']/
Groups/Group[@Name='Accountant']]"/>
```

Each association path is enclosed within a pair of left and right square bracket delimiters. An object is returned when all path levels of each association path are successfully searched using the object as the starting point for each search.

This request returns objects that are owned by a Person or IdentityGroup named Joe Accountant and also have a Reports association to a Report with a name of Sales that belongs to a Group named Accountant.

Using OMI_XMLSELECT with Other Flags

By default, XML searches are not case sensitive. A case-sensitive search can be performed by specifying the OMI_MATCH_CASE (512) flag with the OMI_XMLSELECT flag.

Controlling XMLSELECT Query Size

If the XMLSELECT query size becomes an issue, the OMI_MATCH_CASE (512) flag can be specified to reduce the size of the query. This flag prevents attribute values from being uppercased before the match comparison. To obtain the benefits, the attribute values in the XMLSELECT string must match the values in the metadata repository.

Examples of Search Strings That Filter Objects Based on UsageVersion

The SAS Metadata Model defines a UsageVersion attribute for all metadata types to enable version management of metadata definitions. A UsageVersion value consists of a major version number (0<=major<=999), a minor version number

(0<=minor<=99), and a build number (0<=build<=9999). The build number is reserved for future use. UsageVersion values are persisted in metadata as a double value in the form *MMMmmbbbb.0*.

Major version zero is reserved to indicate that an object was created prior to SAS 9.2, or that the object is not versioned. Most SAS objects are versioned as 1.0, unless there is a reason (such as an existing versioning scheme) to start at a higher version number.

The following examples show how the comparison operators described in "AttributeCriteria Component" on page 375 can be used to specify version criteria for the UsageVersion attribute.

In the <XMLSELECT search="*criteria*"/> search string, the UsageVersion value can be expressed without the leading zeros in the *MMM* part of the *MMMmmbbbb.0* format. These three examples all refer to version 1.1:

```
@UsageVersion LE   '1010000.0'
@UsageVersion LE  '01010000.0'
@UsageVersion LE '001010000.0'
```

The following are examples of search strings that execute common queries:

- Find version 0 or non-versioned objects: `<XMLSELECT search="*[@UsageVersion EQ '0.0']"/>`
- Find version 1.0 objects: `<XMLSELECT search="*[@UsageVersion EQ '1000000.0']"/>`
- Find version 1.1 objects: `<XMLSELECT search="*[@UsageVersion EQ '1010000.0']"/>`
- Find version 1.10 objects: `<XMLSELECT search="*[@UsageVersion EQ '1100000.0']"/>`
- Find all major version 1 objects less than or equal to version 1.1: `<XMLSELECT search="*[@UsageVersion GE '1000000.0' and @UsageVersion LE '1010000.0' ]"/>`
- Find all major version 1 objects: `<XMLSELECT search="*[@UsageVersion GE '1000000.0' and @UsageVersion LT '2000000.0']"/>`

Example of a GetMetadataObjects Request That Specifies the <XMLSELECT search="*criteria*"/> Element

The following example shows how the <XMLSELECT search="*criteria*"/> element is specified in a GetMetadataObjects method call. The search string specifies concatenated AssociationPath criteria.

```
<GetMetadataObjects>
   <Reposid>A0000001.A52WE4LI</Reposid>
   <Type>Document</Type>
   <Objects/>
   <NS>SAS</NS>
```

```
   <!-- Specify the OMI_XMLSELECT (128) flag  -->
   <Flags>128</Flags>
<!-- Include the <XMLSELECT search="criteria"/> element -->
   <Options>
      <XMLSELECT search="*[ResponsibleParties/*[@Role='Owner']/Persons/*
     [@Name='Joe Accountant']][Reports/Report[@Name='Sales']/Groups/Group
     [@Name='Accountant']]"/>
   </Options>
</GetMetadataObjects>
```

Output is returned in the <OBJECTS> element.

Filtering the Associated Objects That Are Returned by a GetMetadataObjects Request

Search criteria that are specified in the <XMLSELECT search="*criteria*"/> element of a GetMetadataObjects method call filters the initial set of metadata objects that are retrieved. You can filter the associated objects that are retrieved by GetMetadataObjects by setting the OMI_GET_METADATA (256) and OMI_TEMPLATE (4) flags and specifying a Search attribute in the association name subelement of a template that requests associated objects.

The Search attribute supports full search syntax as described in “<XMLSELECT search="criteria"/> Syntax” on page 373.

The Search attribute is specified as follows:

```
<AssociationName search="Object"/>

<AssociationName search="Object[Criteria]"/>
```

- *Object* can be an * or a SAS Metadata Model metadata type. The metadata type must be a valid associated object for the specified <ASSOCIATIONNAME>.

  When *Object* is an *, the GetMetadataObjects method selects for retrieval all metadata types that are valid for <ASSOCIATIONNAME>, similar to specifying <ASSOCIATIONNAME/> without search criteria.

  When *Object* is a metadata type, the GetMetadataObjects method gets only associated objects of the specified metadata type.

- [*Criteria*] is an attribute criteria specification or an association path specification that conforms to the syntax documented in “<XMLSELECT search="criteria"/> Syntax” on page 373. When criteria are specified, GetMetadataObjects gets only associated objects specified by *Object* that also meet the specified criteria.

The following is an example of a GetMetadataObjects request that specifies search criteria on an association name. The request specifies to get Document objects and ExternalTable objects that are associated with the Document objects through the Objects association and have the words Human Resources in their Desc attribute.

```
   <GetMetadataObjects>
      <Reposid>A0000001.A5DQTZY5</Reposid>
      <Type>Document</Type>
      <Objects/>
      <NS>SAS</NS>
      <!-- OMI_GET_METADATA(256) + OMI_TEMPLATE (4) + OMI_ALL_SIMPLE (8) -->
```

```
      <Flags>268</Flags>
      <Options>
         <Templates>
            <Document>
              <Objects search="ExternalTable[@Desc ? 'Human
   Resources']"/>
            </Document>
         </Templates>
      </Options>
   </GetMetadataObjects>
```

In the request, note the following:

- The <REPOSID> element identifies the repository from which to get the objects.
- The <TYPE> element specifies to get objects of metadata type Document.
- The <FLAGS> element specifies the sum of the OMI_GET_METADATA, OMI_TEMPLATE, and OMI_ALL_SIMPLE flags (256 + 4 + 8 = 268). The OMI_GET_METADATA and OMI_TEMPLATE flags are required to process the request. OMI_ALL_SIMPLE is optional and is used here to show the filtering that occurs. When the required flags are used alone, the GetMetadataObjects method gets only the Id attribute of selected associated objects.
- The <OPTIONS> element includes a <TEMPLATES> element that contains a template. The template specifies to get ExternalTable objects that are associated with the Document objects through the Objects association and have the words Human Resources in their Desc attribute.

Here is an example of the output from the request:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
   <Document Id="A5DQTZY5.B9000001" Name="MyDocument" ChangeState=""
Desc="Document object created to do search string tests" LockedBy="" MetadataCreated=
"07Aug2008:14:04:35" MetadataUpdated="07Aug2008: 18:40:11" PublicType="Document"
TextRole="" TextType="" URI="text file" URIType="" UsageVersion="1000000">
   <Objects SEARCH="ExternalTable[@Desc ? 'Human Resources']">
          <ExternalTable  Id="A5DQTZY5.BA000002" ChangeState="" Desc="Human Resources
information from Oracle database" LockedBy="" MetadataCreated="07Aug2008:14:04:35"
MetadataUpdated="07Aug2008: 18:40:36" Name="Oracle HR" NumRows="-1" PublicType=
"ExternalFile"  TableName="" UsageVersion="1000000"/>
         <ExternalTable  Id="A5DQTZY5.BA000004" ChangeState="" Desc="Human Resources
information from Sybase database" LockedBy="" MetadataCreated="07Aug2008:14:04:35"
MetadataUpdated="07Aug2008: 18:40:36" Name="Sybase HR" NumRows="-1" PublicType=
"ExternalFile"  TableName="" UsageVersion="1000000"/>
   </Objects>
</Document>
</Objects>
```

Two ExternalTable objects that met the selection criteria were found .

Example of Using <XMLSELECT search="*criteria*"/> and a Template

The following is an example of a GetMetadataObjects request that uses an <XMLSELECT search="*criteria*"/> element to filter the initial set of objects that are retrieved, and a template to filter the associated objects that are retrieved:

```
<GetMetadataObjects>
   <Reposid>A00000001.A5DQTZY5</Reposid>
   <Type>Document</Type>
   <Objects/>
   <NS>SAS</NS>
<!-- OMI_XMLSELECT(128) + OMI_GET_METADATA(256) + OMI_TEMPLATE (4)
+ OMI_ALL_SIMPLE (8) -->
   <Flags>396</Flags>
   <Options>
      <XMLSELECT search="*[@Name ? 'Customer']"/>
      <Templates>
         <Document Id="" Name="" TextType="">
           <ResponsibleParties search="ResponsibleParty[@Role='OWNER']"/>
         </Document>
      </Templates>
   </Options>
</GetMetadataObjects>
```

In the request, note the following:

- The <REPOSID> element identifies the repository from which to get the objects.
- The <TYPE> element specifies to get objects of metadata type Document.
- The <FLAGS> element specifies the sum of the OMI_XMLSELECT, OMI_GET_METADATA, OMI_TEMPLATE, and OMI_ALL_SIMPLE flags (128 + 256 + 4 + 8 = 396).
- The <OPTIONS> element includes both an <XMLSELECT search="*criteria*"/> element and a <TEMPLATES> element. The <XMLSELECT search="*criteria*"/> search string specifies to get only Document objects that contain the word Customer in the Name attribute. The property string in the <TEMPLATES> element specifies to get only associated ResponsibleParty objects that have the Role attribute value of OWNER.

Here is an example of the output from the request:

```
<!-- Using the GETMETADATAOBJECTS method. -->

<Objects>
   <Document  Id="A5DQTZY5. BN000002" Name="2008 New Customers"
ChangeState=" " Desc="New customers in the Southwest Region in 2008" LockedBy=""
MetadataCreated="21Aug2008:21:11:32" MetadataUpdated="21Aug2008:21:11:32"
PublicType=""  TextRole= ""  TextType="HTML" URI=""  URIType=""
UsageVersion="0">
<ResponsibleParties SEARCH="ResponsibleParty[@Role='OWNER']">
<ResponsibleParty Id="A5DQTZY5. BE000004" ChangeState=" " Desc=" " LockedBy=""
```

```
MetadataCreated="21Aug2008:21:11:32" MetadataUpdated="21Aug2008:21:11:32"
Name="Manager" Role="Owner" UsageVersion="0"/></ResponsibleParties>
</Document>
</Objects>
```

One Document object that included the name Customer in the Name attribute was found. One ResponsibleParty object that had the Role attribute value of OWNER was found.

For more information, see “Understanding Templates” on page 337 and “Creating Templates for the Get Methods” on page 340.

Example of Getting Objects That Do Not Have a Specified Association

The NOT function is intended to be used primarily with the GetMetadataObjects method. The NOT function can be specified in the <XMLSELECT search="*criteria*"/> element to find all objects of a specified metadata type that do NOT have an association through the specified *AssociationPath*.

To be useful, a Property object must be associated with an object, or belong to a PropertySet or a PropertyGroup. The following request specifies to concatenate three association paths with the NOT function to return Property objects that do not have any of these associations:

```
<GetMetadataObjects>
  <Reposid>A0000000.A5TJRDIT</Reposid>
  <Type>Property</Type>
  <Objects/>
  <Ns>SAS</Ns>
  <Flags>404</Flags>
  <Options>
  <XMLSELECT search="Property[not(AssociatedObject/*)]
[not(AssociatedPropertySet/*)][not(AssociatedPropertyGroup/*)]"/>
  <Templates>
  <Property Id=" Name="" Desc="" PropertyName="" PropertyRole=""/>
  </Templates>
  </Options>
  </GetMetadataObjects>
```

When there are no orphaned Property objects, the request returns no objects.

18

Metadata Locking Options

Overview of Metadata Locking Options

The SAS Open Metadata Interface enables you to control concurrent access to metadata by multiple users in two ways:

- You can perform object-level locking within a repository by setting the SAS Open Metadata Interface OMI_LOCK and OMI_UNLOCK flags.
- You can impose a change management process in which objects are locked and checked from a primary repository to a project repository by using the SAS Open Metadata Interface change management facility.

To use these mechanisms, a user must have a registered identity on the SAS Metadata Server. For more information, see the *SAS Intelligence Platform: Security Administration Guide*.

The change management functionality is implemented in SAS Data Integration Studio. For more information, see the SAS Data Integration Studio documentation.

Using SAS Open Metadata Interface Flags to Lock Objects

The SAS Open Metadata Interface provides the OMI_LOCK, OMI_UNLOCK, and OMI_UNLOCK_FORCE flags to lock metadata objects. These flags represent the simplest concurrency control provided by the SAS Open Metadata Interface. Before making an update, issue a GetMetadata method call with the OMI_LOCK flag on the object that you want to modify. The specified object and any associated objects are locked. Each object's LockedBy attribute is updated with the metadata identifier representing the caller. The object remains locked from update by users other than the caller until a GetMetadata or UpdateMetadata method is issued with the OMI_UNLOCK or OMI_UNLOCK_FORCE flag.

OMI_UNLOCK unlocks a lock held by the caller. It is the recommended method of unlocking a lock. OMI_UNLOCK_FORCE unlocks a lock held by another user. It is to be used as an emergency override mechanism. While locked, objects can be read by other users. They cannot be updated by other users.

The LockedBy attribute is set and cleared automatically by the lock flags. It can be queried to determine whether an object is locked, and who holds the lock. The LockedBy attribute cannot be changed or cleared directly with the UpdateMetadata method.

For more information, see “GetMetadata Method” on page 109. Also see “UpdateMetadata Method” on page 139.

19

Deleting Metadata Objects

Overview of DeleteMetadata Functionality

The DeleteMetadata method is provided to remove metadata from a SAS Metadata Repository. The DeleteMetadata method can also be used to delete a SAS Metadata Repository.

Using DeleteMetadata to Delete Objects from a SAS Metadata Repository

To use the DeleteMetadata method to delete an object from a SAS Metadata Repository, submit a metadata property string identifying the object that you want to delete in the method's INMETADATA parameter. Specify the SAS namespace in the NS parameter, and set the OMI_TRUSTED_CLIENT (268435456) flag in the FLAGS parameter.

When a SAS namespace metadata type is specified for removal, the server responds as follows:

- If the specified metadata type is a PrimaryType subtype in the SAS Metadata Model, the server checks to see whether the object has a value in the PublicType attribute. If a value is found, and it matches the TypeName value in a type definition in the SAS type dictionary, the server deletes the specified object and any associated objects that are indicated for the object in the type definition, unless you specify a user-defined template.

The purpose of the SAS type dictionary is to hide the details of an object's logical metadata definition from clients. You can override the SAS type dictionary by setting the OMI_TEMPLATE (4) flag, and submitting a user-defined template.

- If the specified metadata type is a PrimaryType subtype and the PublicType attribute does not have a value, or is a SecondaryType subtype in the SAS Metadata Model, then only the specified object and any associated objects that have a 1..1 association to the object are deleted, unless you specify a user-defined template.

An object that is a SecondaryType subtype in the SAS Metadata Model is considered to be owned by a PrimaryType object, and is thus deleted when the PrimaryType object is deleted, although it does not have to be. SecondaryType subtypes and PrimaryType subtypes that do not store a value in the PublicType attribute are treated as independent objects by the DeleteMetadata method.

DeleteMetadata is typically used to delete one metadata entity at time (logical metadata definition or independent object). If you want to delete two or more entities, specify additional property strings in the INMETADATA parameter.

Note: When you specify multiple independent objects, take care not to specify associated objects that have a 1..1 cardinality in the same DeleteMetadata request. A 1..1 cardinality indicates a required relationship. When one object in the association is deleted, the metadata server automatically deletes the other object as well. If you specify the partner object for deletion, the metadata server attempts to locate objects that have already been deleted, and cancels the Delete operation when they are not found. You can prevent the operation from being canceled by setting the OMI_IGNORE_NOTFOUND (134217728) flag, but it is better to avoid specifying the associated objects instead.

DeleteMetadata does not list the identifiers of associated object instances that are deleted by default. To include them in the output listing, set the OMI_RETURN_LIST (1024) flag.

For an example of a DeleteMetadata request that deletes a specified object, see "DeleteMetadata Method" on page 102. For information about how to specify a user-defined template, see "Creating a Template for DeleteMetadata" on page 396.

Creating a Template for DeleteMetadata

Purpose

User-defined templates are supported in DeleteMetadata to enable clients to delete custom logical metadata definitions. A custom logical metadata definition is a logical metadata definition that meets the following criteria:

- does not use a PrimaryType subtype as its primary object
- uses a PrimaryType subtype as its primary object, but does not store a value in the PublicType attribute

- uses a PrimaryType subtype and specifies a PublicType value, but you want to delete a subset of its associations or associated objects

To delete a custom logical metadata definition:

1. Specify the primary or top-level object in the logical metadata definition in the INMETADATA parameter.
2. Specify the SAS namespace in the NS parameter.
3. Set the OMI_TEMPLATE (4) flag in the FLAGS parameter.
4. Specify a template in the <TEMPLATES> element in the OPTIONS parameter. The template should specify the association names to traverse to delete associated objects.

When the OMI_TEMPLATE (4) flag is set, and an appropriate user-defined template is submitted in the OPTIONS parameter, DeleteMetadata ignores the value in the metadata object's PublicType attribute if one is found, and deletes the specified associations instead.

Form of a DeleteMetadata Template

A DeleteMetadata template is a metadata property string that specifies the associations that should be deleted with the object instance that is specified in the INMETADATA parameter of the DeleteMetadata method. You can submit the metadata property string to the metadata server in two ways. Specify the metadata property string directly within the <TEMPLATES> element in the OPTIONS parameter, as follows:

```
<Templates>
    <MetadataType>
     <AssociationName1/>
     <AssociationName2/>
     <AssociationName3/>
   </MetadataType>
</Templates>
```

MetadataType is the same metadata type as in the INMETADATA parameter, and *AssociationNamen* are association names that are valid for *MetadataType*, as defined in the SAS Metadata Model.

Or, specify the metadata property string in a <TEMPLATE> subelement in the <TEMPLATES> element in the OPTIONS parameter, as follows:

```
<Templates>
 <Template TemplateName="name">
   <MetadataType>
     <AssociationName1/>
     <AssociationName2/>
     <AssociationName3/>
   </MetadataType>
  </Template>
</Templates>
```

When the second way is used, the INMETADATA property string must specify a TemplateName attribute with a matching value.

The new template form is useful when you are deleting multiple entities at once. It is also useful when you want to take advantage of the template attributes. For more information, see “Template Attributes” on page 339. Also, see “Examples” on page 398. Otherwise, the legacy template form is more efficient and sufficient for most requests.

How DeleteMetadata Processes a User-Defined Template

The DeleteMetadata method processes requests that contain a template as follows:

- The OMI_TEMPLATE flag instructs the DeleteMetadata method to look for a user-defined template in a <TEMPLATES> subelement in the OPTIONS parameter.
  - If a TemplateName attribute is specified in the INMETADATA parameter, the server looks for a <TEMPLATE> subelement that has a matching TemplateName value in the OPTIONS parameter and uses it if it is found. If a matching named template is not found in the OPTIONS parameter, the server returns an error.
  - If a TemplateName attribute is not specified in the INMETADATA parameter, the server looks for a metadata property string of the same metadata type that was specified in the INMETADATA parameter in the <TEMPLATES> element in the OPTIONS parameter and uses it if it is found. If a matching metadata type is not found, and the object is a PrimaryType subtype with a valid value in the PublicType attribute, then the SAS type dictionary is used to delete the object. Otherwise, the server deletes the specified object only.

Examples

The following example shows how to issue a DeleteMetadata request that submits a user-defined template using the legacy template form. The specified Group has four objects associated through the Members association: a PhysicalTable object named “My Table,” a PhysicalTable object named “Their Table,” a TextStore object named “My Notes,” and a TextStore object named “Their Notes.” The request specifies to delete the Group object and all objects associated to it through the Members association. The request is formatted for the INMETADATA parameter of the DoRequest method:

```
<DeleteMetadata>
<Metadata>
<Group Id="A5TJRDIT.A1000004" Name="My Group"/>
</Metadata>
<Ns>SAS</Ns>
<!-- OMI_TRUSTED_CLIENT, OMI_TEMPLATE, + OMI_RETURN_LIST -->
<Flags>268436484</Flags>
<Options>
<Templates>
 <Group>
   <Members/>
</Group>
</Templates>
</Options>
```

```
</DeleteMetadata>
```

Here is sample output from the request, reformatted for readability:

```
<DeleteMetadata>
<Metadata>
<Group Id="A5TJRDIT.A1000004"/>
<PhysicalTable Id="A5TJRDIT.B20000TF"/>
<PhysicalTable Id="A5TJRDIT.B20000TG"/>
<TextStore Id="A5TJRDIT.AE0001D9"/>
<TextStore Id="A5TJRDIT.AE0001DA"/>
</Metadata>
<Ns>SAS</Ns>
<Flags>268436484</Flags>
<Options>
<Templates>
<Group>
<Members/>
</Group>
</Templates>
</Options>
</DeleteMetadata>
```

The following example shows how to issue a DeleteMetadata request that submits a user-defined template using the new template form. This request specifies to delete the Group object described in the previous example. However, in this request, we specify to delete the Group object, and its associated My Table and My Notes objects, leaving the other two objects intact.

```
<DeleteMetadata>
 <Metadata>
 <Group Id="A5TJRDIT.A1000004" TemplateName="test"/>
 </Metadata>
<Ns>SAS</Ns>
<!-- OMI_TRUSTED_CLIENT, OMI_TEMPLATE, + OMI_RETURN_LIST -->
<Flags>268436484</Flags>
<Options>
<Templates>
<Template TemplateName="test">
<Group>
 <Members>
   <PhysicalTable match="@Name='My Table'" TemplateExpand="Yes"/>
   <PhysicalTable TemplateExpand="No"/>
   <TextStore match="@Name='My Notes'" TemplateExpand="Yes"/>
   <TextStore TemplateExpand="No"/>
  </Members>
 </Group>
</Template>
</Templates>
</Options>
</DeleteMetadata>
```

Here is sample output from the request, reformatted for readability:

```
<DeleteMetadata>
<Metadata>
<Group Id="A5TJRDIT.A1000004"/>
<PhysicalTable Id="A5TJRDIT.B20000TF"/>
<TextStore Id="A5TJRDIT.AE0001D9"/>
```

```
</Metadata>
<Ns>SAS</Ns>
<Flags>268436484</Flags>
<Options>
<Templates>
<Template TemplateName="test">
<Group>
<Members>
<PhysicalTable match="@Name='My Table'" TemplateExpand="Yes"/>
<PhysicalTable TemplateExpand="No"/>
<TextStore match="@Name='My Notes'" TemplateExpand="Yes"/>
<TextStore TemplateExpand="No"/>
</Members>
</Group>
</Template>
</Templates>
</Options>
</DeleteMetadata>
```

The Match and TemplateExpand attributes are used to filter the associated objects that are deleted. TextStore and PhysicalTable objects that do not meet the match criteria are ignored.

Deleting a Repository

Note: You must have administrative user status on the SAS Metadata Server to unregister, clear, or delete a repository. For more information about administrative user status, see the *SAS Intelligence Platform: Security Administration Guide*.

The DeleteMetadata method call for deleting a repository is similar to a DeleteMetadata method call for deleting an application metadata, with two exceptions. To delete a repository, specify the following:

- The REPOS namespace.
- One of several flags that indicate whether you want to delete the whole repository, simply clear the repository's contents, or you only want to unregister the repository.

Keep in mind the following things when using the REPOS namespace flags:

- You should not unregister or delete the foundation repository if you have other repositories defined.
- You should not combine REPOS namespace flags in a request.
- Do not attempt to clear the objects from a project repository or delete a project repository using the DeleteMetadata method. Use SAS Data Integration Studio or SAS Management Console to clear or delete project repositories. These products contain extra code to handle locks on objects in non-project repositories that are related to objects in project repositories.

A repository is unregistered by executing the DeleteMetadata method with the OMI_TRUSTED_CLIENT (2097152) flag on a repository object in the REPOS namespace.

Set the OMI_REINIT and the OMI_TRUSTED_CLIENT flags to clear a repository if you want to repopulate it completely with different metadata.

Set the OMI_DELETE (32) and the OMI_TRUSTED_CLIENT flags to delete a repository. OMI_DELETE deletes the contents of a repository and removes the whole registration from the SAS Repository Manager.