THE POWER TO KNOW.

SAS® 9.2 Open Metadata Interface

Reference and Usage

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2010. *SAS® 9.2 Open Metadata Interface: Reference and Usage*. Cary, NC: SAS Institute Inc.

SAS® 9.2 Open Metadata Interface: Reference and Usage

Copyright © 2010, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

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

1st electronic book, February 2010

2nd electronic book, September 2010

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

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Contents

What's New

Overview

The SAS Open Metadata Interface documentation has been reorganized. You need information about this reorganization to successfully use this book.

The SAS 9.2 Open Metadata Interface software has been enhanced to provide improved metadata access, authorization, and server control functionality. It also offers a new security administration interface.

See the following topics for information about specific enhancements in each area:

Documentation Changes

- The content of *SAS 9.1.3 Open Metadata Interface: Reference*, the *SAS 9.1.3 Open Metadata Interface: User's Guide*, and the *SAS 9.1.3 Java Metadata Interface: User's Guide* has been merged into one document. The new document is the *SAS 9.2 Open Metadata Interface: Reference and Usage*. For information about the reorganization, see "About This Book" on page 3.
- Documentation about SAS Metadata Model metadata types is not in the *SAS 9.2 Open Metadata Interface: Reference and Usage*. This information is provided in a separate document: the *SAS 9.2 Metadata Model: Reference*.
- Documentation about PROC METADATA and SAS metadata DATA step functions is not included in the *SAS 9.2 Open Metadata Interface: Reference and Usage*. It is provided in *SAS 9.2 Language Interfaces to Metadata*.
- *SAS 9.2 Open Metadata Interface: Reference and Usage* and *SAS 9.2 Metadata Model: Reference* are available only online. See Product Documentation in the Knowledge Base, available at http://support.sas.com.

This *What's New* section describes enhancements to the SAS Open Metadata Interface. For information about enhancements to the SAS 9.2 Java Metadata Interface, see "What's New in the SAS 9.2 Java Metadata Interface" on page 19. For information about enhancements to the SAS 9.2 Metadata Model and SAS language interfaces to metadata, see their documentation.

Metadata Access Enhancements

The SAS Open Metadata Interface provides the IOMI server interface for creating and accessing metadata in a SAS Metadata Repository. The IOMI server interface has been enhanced to improve functionality in several areas. The areas and specific enhancements are as follows:

- Simplify creating and querying cross-repository references between objects in the foundation and custom repositories.
  - Dependency associations are no longer required to be defined between the foundation and custom repositories before cross-repository references can be created between objects in the repositories. Cross-repository references can now be created between objects in the foundation repository and custom repositories without preparation or restriction.
  - The foundation repository and custom repositories, which contain metadata for general use, are now referred to as public repositories. Project repositories, which serve as development playpens, are now referred to as private repositories.
  - Clients no longer need to set directionality flags in a GetMetadata request that is issued in a public repository to get information about cross-repository references in other public repositories. A GetMetadata request returns cross-repository references from all public repositories by default.
- Better separate metadata in public and private repositories.
  - Because dependencies are no longer required, a project repository can serve as a development playpen for any public repository. However, a user needs CheckinMetadata permission in the public repository's default ACT to update the repository with information from a project repository.
  - The OMI_DEPENDENCY_USED_BY (16384) flag, which used to specify directionality for queries in SAS 9.1, has been repurposed. In SAS 9.2, specify OMI_DEPENDENCY_USED_BY in the GetMetadata method only if you want to include associated objects from private repositories in a request that is issued in a public repository. For more information, see "GetMetadata" on page 89 and "GetMetadata and Cross-Repository References in SAS 9.2" on page 244.

    Specify OMI_DEPENDENCY_USED_BY in the GetMetadataObjects method to include objects of the specified metadata type from all private repositories in the method results.
  - The OMI_DEPENDENCY_USES (8192) flag has been repurposed. In SAS 9.2, set OMI_DEPENDENCY_USES in the GetMetadataObjects method to include objects of the specified metadata type from all public repositories in the method results. For more information, see "GetMetadataObjects" on page 93 and "Expanding a GetMetadataObjects Request to Include Additional Repositories" on page 273.
- Improve repository management and reporting.
  - A repository's persisted availability is now controlled by using the value in its Access= attribute. The Access= attribute accepts new values to support

online, read-only, administrative, and offline states for a repository. For more information about supported values, see the RepositoryBase metadata type in *SAS 9.2 Metadata Model: Reference*.The Access= attribute is set with the AddMetadata method and modified with the UpdateMetadata method.

- The AddMetadata method creates the file system directory for a new repository when the <CREATEREPOSCONTAINER/> option is used. For more information, see "AddMetadata" on page 77.
- The GetRepositories method now returns repository format, current access, name, path, type, and pause state values when the OMI_ALL (1) flag is set. For more information, see "GetRepositories" on page 97.

- Improve metadata searching capabilities and selection criteria. The GetMetadataObjects XMLSELECT search syntax has been enhanced by the following features:
  - GE (greater than or equal to), NE (not equal to), and LE (less than or equal to) operators have been added for specifying selection criteria.
  - GT (greater than) and LT (less than) operators have been extended to operate on character string values and numeric values.
  - Concatenated association paths are supported as selection criteria.
  - An object qualifier is supported on search strings that are specified on association names to filter the associated objects that are selected.

  For more information, see Chapter 15, "Filtering a GetMetadataObjects Request," on page 277.
- Support a consistent user interface to metadata in client user interfaces through the following new methods:

  AddUserFolders
  : Creates a home folder and subfolders for a Person. For more information, see "AddUserFolders" on page 81.

  GetUserFolders
  : Gets the home folder or specified subfolder for a Person. For more information, see "GetUserFolders" on page 107.

- Support metadata ownership through the following new methods:

  AddResponsibleParty
  : Creates a ResponsibleParty object for a Person or IdentityGroup in the repository that contains the Person or IdentityGroup object, even if the caller does not have WriteMetadata permission to the repository. For more information, see "AddResponsibleParty" on page 79.

  GetResponsibleParty
  : Gets the ResponsibleParty object associated with a Person or IdentityGroup and responsibility, even if the caller does not have ReadMetadata permission to the repository. For more information, see "GetResponsibleParty" on page 100.

- Improve metadata delete functionality. The DeleteMetadata method has been enhanced to delete the specified object and associated objects that are defined in a user-defined template. For more information, see "DeleteMetadata" on page 84 and Chapter 17, "Deleting Metadata Objects," on page 297.
- Return information about supported values for metadata type properties from the SAS Metadata Model. For more information, see "GetTypeProperties" on page 104.

Other changes in the IOMI server interface include:

- The CopyMetadata method is deprecated.

- The CheckinMetadata, CheckoutMetadata, FetchMetadata, and UndoCheckoutMetadata methods are retired.
- SAS Metadata Repository auditing is no longer supported. Client requests to store values for AuditPath=, AuditType=, AuditEngine=, and AuditOptions= attributes while adding a repository with the AddMetadata method or updating a repository definition with the UpdateMetadata method are ignored.
- The SAS Metadata Server class factory number has been changed to prevent SAS 9.1.3 clients from accessing a SAS 9.2 Metadata Server without first being updated to SAS 9.2. For more information, see "Connecting to the SAS Metadata Server" on page 12.

Authorization Enhancements

The SAS Open Metadata Interface provides the ISecurity server interface for requesting authorizations on metadata. The ISecurity server interface has been enhanced as follows:

- The IsAuthorized method now supports a Uniform Resource Name (URN) in the form REPOSID:*_reposID* in the RESOURCE parameter. For more information, see "IsAuthorized" on page 145.
- Several new methods have been added that support authorization based on roles, multiple authorizations, and internal user authentication.
  - The GetApplicationActionsAuthorizations method returns authorizations for the ApplicationActions in a SoftwareComponent object. For more information, see "GetApplicationActionsAuthorizations" on page 121.
  - The IsInRole method returns the TRUE value when the user indicated in the CREDHANDLE parameter is in a role. For more information, see "IsInRole" on page 148.
  - The GetAuthorizationsOnObject method returns the permissions that apply to a resource for all identities or specified identities. For more information, see "GetAuthorizationsOnObj" on page 173.
  - The GetInfo method gets 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" on page 132.
  - The GetLoginsforAuthDomain method gets the logins for the connected user for the specified authentication domain in order of identity precedence. For more information, see "GetLoginsforAuthDomain" on page 143.
  - New internal user authentication methods include:

    GetInternalLoginSitePolicies
    : Returns the active server-level internal authentication policies.

    SetInternalPassword
    : Creates an InternalLogin object for the specified user.

    SetInternalLoginUserOptions
    : Customizes internal authentication policies for the specified user.

    GetInternalLoginUserInfo
    : Gets availability information and internal authentication settings for the specified user.

    DeleteInternalLogin
    : Deletes the InternalLogin object that is associated with the specified user.

- New functionality is available in the ISecurity 1.1 interface. Clients that do not want the new SAS 9.2 methods and functionality should call the ISecurity server interface as they did in SAS 9.1.3. For more information, see "Using the ISecurity Server Interface" on page 116.

Server Control Enhancements

The SAS Open Metadata Interface provides the IServer server interface for controlling the SAS Metadata Server and getting server status information. The IServer server interface has been enhanced as follows:

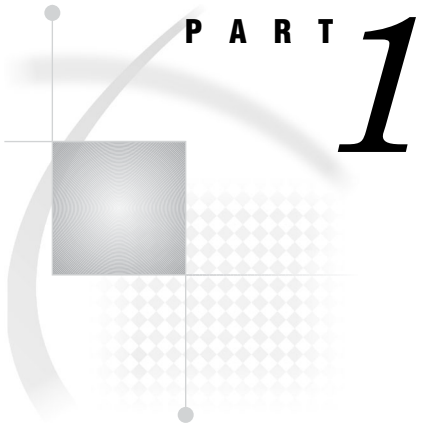
- The Pause method can no longer be used to downgrade the availability of a specified SAS Metadata Repository.
- The Pause method operates exclusively on the SAS Metadata Server and can downgrade the SAS Metadata Server to an ADMIN state (only users who have administrative user status on the SAS Metadata Server can access repositories) or the OFFLINE state (the SAS Metadata Server is not available to any users).
- The Pause method now supports a <PAUSECOMMENT> element to enable callers to include a user-defined text message that specifies the reason for a server pause for clients. The element is passed to the SAS Metadata Server in the OPTIONS parameter. For more information, see "Pause" on page 189.
- The Status method has been enhanced to poll the SAS Metadata Server for the content of the <PAUSECOMMENT> option.
- The Status method has been enhanced to poll the SAS Metadata Server for the SAS platform version, which includes the repository level, and the server locale.
- The Status method has been enhanced to optionally poll the SAS Metadata Server for the values of omaconfig.xml options and to provide journaling statistics. For more information, see "Status" on page 194.
- The Refresh method has been enhanced to support an <OMA JOURNALPATH="*file-name*"/> option. This option can change the location of the journal file on a running SAS Metadata Server. For more information, see "Refresh" on page 191.

New Security Administration Server Interface

A new server interface, ISecurityAdmin, provides three categories of methods:

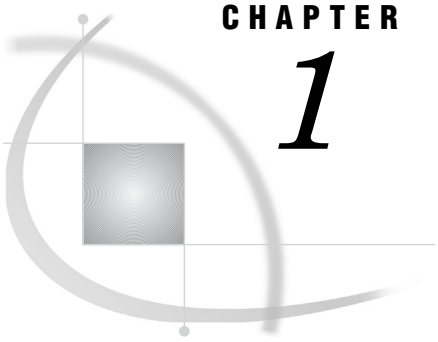
- Transaction context methods enable programmers of interactive clients to record user interactions and return correct effective permissions for authorization changes, factoring in group memberships, before applying the changes to authorization metadata on the SAS Metadata Server. The BeginTransactionContext 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 provide a programmatic way to assign and get permissions for identities on resources, to list authorized identities, and to apply and remove access control templates (ACTs) from resources.
- ACT administration methods create ACTs, modify the attributes of ACTs, list ACTs, and destroy ACTs.

For more information, see Chapter 8, "Security Administration (ISecurityAdmin Interface)," on page 155.



PART 1

Concepts



CHAPTER

1

Introduction

About This Book

This book provides reference and usage information about the SAS 9.2 Open Metadata Interface. It also provides usage information about the SAS 9.2 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 a Java 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 know XML.

Using these APIs with the SAS Metadata Model, which is described in *SAS 9.2 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.

In SAS 9.2, we encourage the use of the APIs 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 class documentation. You can view a Web-enabled version of SAS Java Metadata Interface documentation at support.sas.com/92api.

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 SAS 9.2 Phase 2 software. The SAS Open Metadata Interface uses the Integrated Object Model (IOM) to communicate with the SAS Metadata Server. Currently, this interface supports Java, Windows, and SAS clients.

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

- SAS 9.2 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 22 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 SAS 9.2 VJR. The VJR is installed when the SAS 9.2 Intelligence Platform Object Framework or SAS Management Console is installed. For easy access to the SAS 9.2 VJR, we recommend the SAS AppDev Studio development environment.

SAS language interfaces to metadata such as PROC METADATA and SAS metadata DATA step functions simply need access to Base SAS 9.2 software.

Prerequisites

- You must have access to a properly configured SAS 9.2 Metadata Server to create and access metadata. A properly configured metadata server has a foundation repository that contains standard SAS 9.2 metadata. The SAS 9.2 Deployment

Wizard installs and configures a proper SAS 9.2 Metadata Server for you. An existing SAS 9.1.3 Metadata Server can be migrated to the SAS 9.2 environment by using the SAS Migration Utility with the SAS 9.2 Deployment Wizard. For more information, see *SAS Intelligence Platform: 9.1.3 to 9.2 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 open clients.

It is a secondary source of information for users of the SAS language interfaces to metadata. In SAS 9.2, 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 as background information and 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.

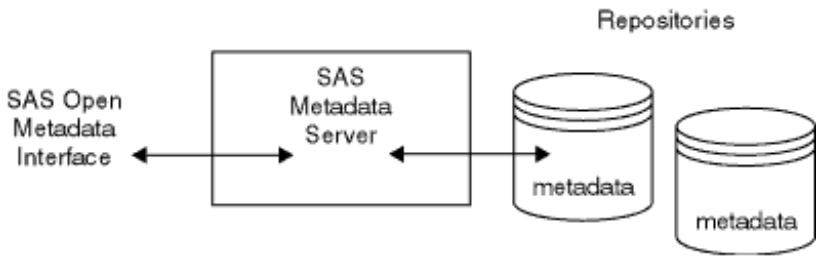
- The metadata model, called the SAS Metadata Model, provides classes and objects that define repositories, the SAS Repository Manager, and different types of application metadata.

  The SAS Metadata Server uses information that clients store 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 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. Its purpose is to provide a central, shared location for accessing metadata.

Display 1.1 SAS Open Metadata Architecture



What Can I Do with the SAS Open Metadata Interface?

The SAS Open Metadata Interface enables 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. For the most part, 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 information that is needed to access data stores so that the information is available centrally and can be maintained independently of the client.
- Return a list of data stores that contain a metadata item that you specify, such as the column name Salary.
- Return a list of available SAS 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 7, "Authorization (ISecurity Interface)," on page 113.

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*.

SAS 9.2 introduces a security administration interface to facilitate the creation and query of access controls. Security administration methods are available in the ISecurityAdmin server interface. For more information, see "Overview of the ISecurityAdmin Server Interface" on page 157.

CHAPTER

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.
- Manage access controls.
- Define and manage internal user accounts.

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 external maintenance tasks can be performed, such as running a backup, recovering memory, or changing certain server configuration and invocation options while the server is online.

A client that defines access controls does so to control access to data by defining controls on the metadata that describes the data. 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 requests authorizations queries the SAS Open Metadata Architecture authorization facility to determine if the specified user has appropriate permission to a requested resource based on active access controls. Then, depending on the decision,

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 authorization facility and how the authorization facility works, see the *SAS Intelligence Platform: Security Administration Guide*.

SAS 9.2 supports authorization based on role membership. Clients can define roles that identify application actions that will be controlled as metadata. Administrators can assign identities to the roles. The GetApplicationActionsAuthorizations method is provided to enable clients to request decisions based on role membership.

A client that manages access controls lists identities that have permissions on a resource, lists permissions that are defined directly on a resource, lists ACTs that are associated with a resource, and applies and removes ACTs from a resource. It can also create an ACT, modify the attributes of an ACT, and destroy an ACT.

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 on the SAS Metadata Server for authorizations to be meaningful. See the *SAS Intelligence Platform: Security Administration Guide* for detailed information about the security features that are available through the SAS Open Metadata Architecture authorization facility.

Important Terms

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

metadata type	specifies a template 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 160 metadata types.
namespace	specifies a group of related metadata types. Namespaces are used to partition metadata into different contexts. The SAS Open Metadata Interface defines two namespaces: SAS and REPOS. The SAS namespace contains metadata types that describe application elements. 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 in SAS 9.2. Instead, if you perform a SAS 9.2 planned installation to

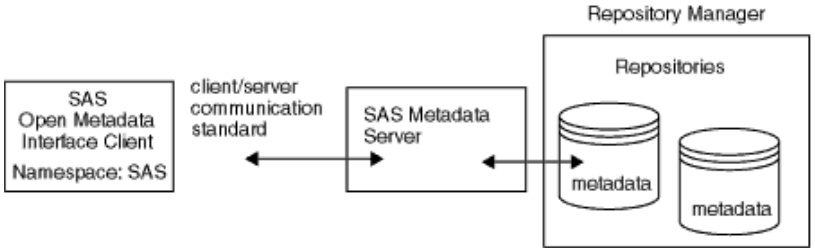
set up your SAS 9.2 Metadata Server, the SAS Deployment Wizard creates the first repository — a foundation repository — for you. We recommend that you create additional repositories with SAS Management Console because it creates default metadata in the repositories for you.

Creating and Accessing Application Metadata

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 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.

Display 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 Metadata Types” in *SAS Metadata Model: Reference*.

The SAS Open Metadata Interface provides the IOMI server interface for reading and writing metadata objects. For information about IOMI, see Chapter 6, “Metadata Access (IOMI Interface),” on page 61.

We recommend that clients use the SAS Java Metadata Interface to read and write metadata in SAS 9.2 instead of IOMI methods directly. For reference information about the SAS 9.2 Java Metadata Interface, see the documentation at support.sas.com/92api. For usage information, see Chapter 4, “Using the SAS Java Metadata Interface,” on page 25.

Methods that create metadata require you to identify the metadata repository in which the object is to be created and to submit an XML metadata property that defines the objects that you want to create. For more information, see Chapter 10, “Adding Metadata Objects,” on page 205. In addition, see “Understanding Associations” in *SAS Metadata Model: Reference*.

For information about the options that are available for accessing existing metadata objects, see “Querying Metadata Objects” on page 236.

Connecting to the SAS Metadata Server

The SAS Metadata Server is an object server. It uses the IOM provided by SAS Integration Technologies to communicate with clients.

SAS Integration Technologies provides interfaces that enable you 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 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 so that clients can use getter or 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, it provides methods for connecting to the SAS Metadata Server with the ISecurity, ISecurityAdmin, and IServer server interfaces, so that clients don't 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.

The SAS Integration Technologies interfaces continue to be supported for metadata clients that already use them. A metadata client that was created before SAS 9.2 should use the latest SAS Integration Technologies interfaces. Because of architectural changes in the SAS 9.2 Metadata Server, a SAS Open Metadata Interface client created using SAS 9.1 or earlier technology cannot connect to a SAS 9.2 Metadata Server without modification. To prevent access, the SAS 9.2 Metadata Server has a new class factory number, which all clients must specify to connect with a SAS 9.2 Metadata Server. The latest SAS Integration Technologies interfaces use the new class number.

SAS Integration Technologies provides the SAS Object Manager for SAS 9.2 Windows client development, and the Java Connection Factory for SAS 9.2 Java client development. For more information about the interfaces, see the *SAS 9.2 Integration Technologies: Windows Client Developer's Guide* and the *SAS 9.2 Integration Technologies: Java Client Developer's Guide*.

Connecting to the SAS Metadata Server with the SAS Java Metadata Interface

The SAS Java Metadata Interface and SAS Open Metadata Interface are contained in JAR files in the SAS 9.2 VJR. For information about the Java Runtime Environment and JAR files required by the SAS Java Metadata Interface, see "JRE and JAR Requirements" on page 22. 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 the MdOMRConnection interface for connecting to the SAS Metadata Server. The MdOMRConnection interface includes the following methods to enable you to connect to the SAS Metadata Server:

- makeOMIConnection — connects to the SAS Metadata Server with the SAS Java Metadata Interface. A client uses this interface to read and write metadata.
- makeISecurityConnection — connects to the SAS Metadata Server with 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 an example 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 37. The sample program is a metadata access client.

For more information about the SAS Java Metadata Interface, see Chapter 3, “Understanding the SAS Java Metadata Interface,” on page 19, and Chapter 4, “Using the SAS Java Metadata Interface,” on page 25.

For more information about connecting with the non-metadata server interfaces, see Chapter 7, “Authorization (ISecurity Interface),” on page 113, Chapter 8, “Security Administration (ISecurityAdmin Interface),” on page 155, and Chapter 9, “Server Control (IServer Interface),” on page 187.

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, and that clients will use 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.

username
: 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

A client must connect to the SAS Metadata Server before sending any requests. After connecting, using an interface simply involves using its methods.

Standard Interface

A method is typically issued by declaring object variables that represent its parameters in the client, and then referencing the object variables in the method request. In this documentation, we refer to this process 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 issue a method 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 a different name, the name in the object variable declaration must also be used to represent the parameter in the method request. In addition, the parameters need to be specified in the order given in the method documentation. As an example, consider the GetApplicationActionsAuthorizations method, whose documented syntax is:

```
GetApplicationActionsAuthorizations(credHandle, applicationContext, options, output);
```

When declaring object variables for the method's parameters, you do not have to use the parameter names CREDHANDLE, APPLICATIONCONTEXT, OPTIONS, and OUTPUT to represent the parameters. However, whatever names you do use, the object variable declarations must be specified in the method in the order given in the syntax statement.

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 an alternate interface, called 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. Clients declare object variables for the DoRequest method's parameters in the client. Then, clients submit another method in the DoRequest method's INMETADATA parameter. This other method's parameters are formatted in an XML string.

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 method string is described in "DoRequest" on page 87. The IOMI reference documentation includes examples of how to format methods for the DoRequest interface.

Because we encourage the use of the SAS Java Metadata Interface to read and write metadata over using the IOMI server interface directly in SAS 9.2, 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 the 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 representing the method parameters, with one exception. The submitted method's INMETADATA parameter

should be represented by a <METADATA> element. For an example of how this element is used, see the DoRequest interface example in "AddMetadata" on page 77.

If a published parameter name is missing from the XML string, the SAS Metadata Server will return an error. The method parameters do not need to be specified in the order given in the syntax.

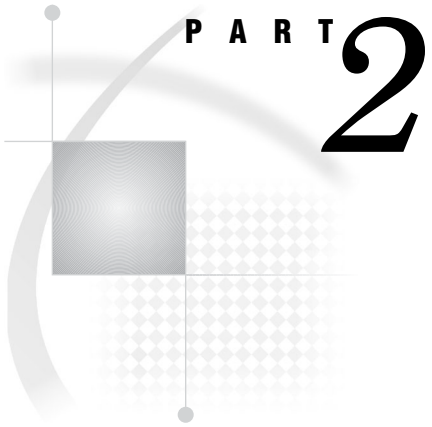
Controlling the SAS Metadata Server

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 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 9, "Server Control (IServer Interface)," on page 187.

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*.

In SAS 9.2, 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 Management Console documentation.



PART 2

SAS Java Metadata Interface

CHAPTER

3

Understanding the SAS Java Metadata Interface

What's New in the SAS 9.2 Java Metadata Interface

Overview

The SAS 9.2 Java Metadata Interface was enhanced to be more efficient in a multi-user environment. In addition, it surfaces new metadata access functionality that was added to the SAS Open Metadata Interface in a Java environment.

General Enhancements

Specific changes and enhancements are as follows:

- The static and remote versions of the SAS Java Metadata Interface have been reconciled to use the same methods to perform the same actions. Use of the remote version is recommended over the use of the static version. This document addresses the use of the remote version. For more information about the remote and static versions, see "SAS Java Metadata Interface Overview" on page 21.
- The MdServerStore class is deprecated. Clients should no longer interact directly with the ServerStore.
- Two new utility classes are available in the MdFactory interface. The MdSecurityUtil class contains methods that return quick authorizations on the caller's ability to read a metadata object, write a metadata object, and add a metadata object to a folder. The MdRepositoryUtil class contains methods for getting the repository type, name, and metadata object identifier using the inverse information.
- The MdObjectFactoryListener is deprecated and replaced by MdFactoryListener. MdFactoryListener can be used to notify all users of a factory when objects are added, updated, or deleted on the SAS Metadata Server.
- A new getServerModelVersion() method replaces the getServerVersion() method, which was used to return the SAS Metadata Model version number in use by the SAS session.

- A new getPlatformVersion() method returns the version number of the active SAS Metadata Server.
- Several new methods were added to the MdOMIUtil interface:
  - The getFoundationRepository() and getFoundationReposID() methods get the foundation repository.
  - The getMetadataNoCache() and getMetadataObjectsNoCache() methods provide quick retrieval of metadata object attributes and associations in a map. These two methods do not cache any data within an object store.
  - The getObjectPath() method returns the path of an object that resides in the SAS folder tree.
  - The getResponsibleParty() method gets the ResponsibleParty object associated with the specified Identity and Responsibility.
  - The getUserHomeFolders() method retrieves the home folder or specific subfolder for the specified user, or creates the folder if it cannot be found.

- The MdOMRConnection interface supports a new method, makeISecurityAdminConnection, to connect to the SAS Metadata Server with the new SAS 9.2 Open Metadata Interface ISecurityAdmin interface. The MdOMRConnection interface has methods to connect with the ISecurity and IServer server interfaces. For information about the makeISecurityAdminConnection method, see Chapter 8, "Security Administration (ISecurityAdmin Interface)," on page 155.

To use the SAS 9.2 Java Metadata Interface effectively, you should be familiar with enhancements to the SAS Open Metadata Interface and SAS 9.2 Metadata Model. See and "What's New in the SAS 9.2 Metadata Model" in the *SAS 9.2 Metadata Model: Reference* for this information.

About This Section

This section describes how to create, read, and update metadata in metadata repositories with the SAS 9.2 Java Metadata Interface. Reference information about the SAS Java Metadata Interface is provided as class documentation. You can view a Web-enabled version of the documentation at support.sas.com/92api.

The SAS Java Metadata Interface is a Java API that provides a Java 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.

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 enabless users to perform metadata access tasks without having to know XML.

The API has interfaces and classes for the following:

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

There are two implementations of the SAS Java Metadata Interface:

- A remote version for applications that support single or multiple users. These applications have objects that need to be sent to a remote environment, such as the SAS middle tier.
- A static version for single-user applications that do not need to support objects in the SAS middle tier.

In SAS 9.2, the static and remote versions have been reconciled to use the same methods to perform the same actions. However, for new applications, the use of the remote version is recommended over the use of the static version for both single-user and multi-user applications. The static version is supported for backward compatibility.

The SAS Java Metadata Interface includes the following Java packages:

com.sas.metadata.remote
: provides the remote Java object interface to the SAS Metadata Server.

com.sas.metadata
: provides the static Java object interface to the SAS Metadata Server.

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, including SAS metadata repositories, Lightweight Directory Access Protocol (LDAP) repositories, and WebDAV repositories, from client applications. Using information service methods, a client can submit a single query that searches all available repositories and returns the results in a smart object that provides a generic interface to common data elements. The com.sas.services.information package is described in the SAS Foundation Services class documentation. SAS Foundation Services is a component of SAS Integration Technologies.

JRE and JAR Requirements

The current release of the Java client software requires Java 2 SDK, Standard Edition, Version 1.5 (JDK 1.5.0).

The SAS Java Metadata Interface depends on several JAR files. These JAR files are included in the SAS VJR, which is installed when the SAS Intelligence Platform Object Framework or SAS Management Console is installed.

The JAR files are:

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

How the SAS Java Metadata Interface Works

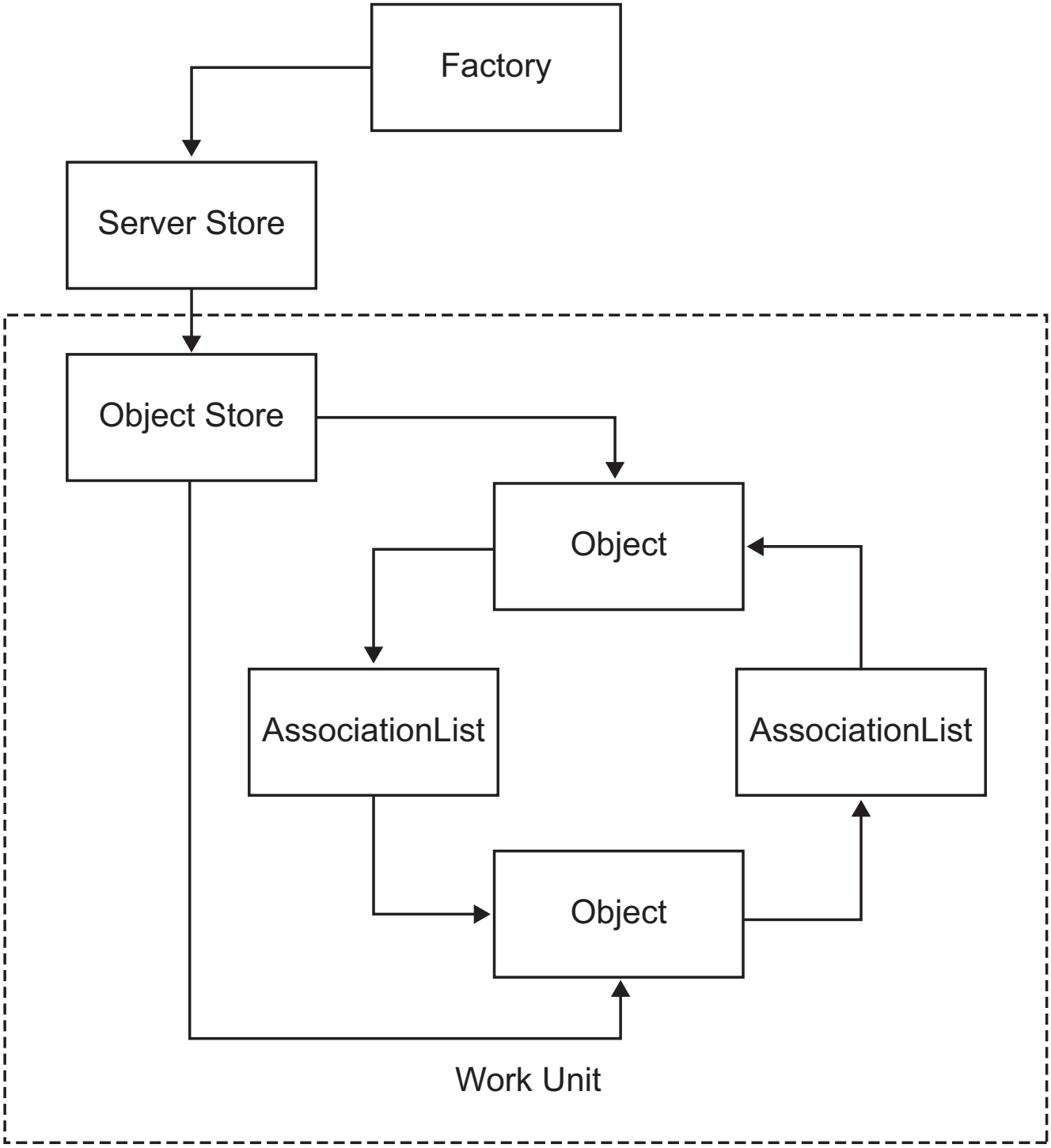
The SAS Java Metadata Interface consists of the following:

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

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 3.1 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 *SAS Metadata Model: Reference*.

In Figure 3.1 on page 23, 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 51.

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 25.

For documentation about specific classes and methods, see the SAS Java Metadata Interface at support.sas.com/92api.

CHAPTER

4

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" and "Summary Table of IOMI Options" in Chapter 6, "Metadata Access (IOMI Interface)," on page 61.

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 metadata property strings. 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 workunit. 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 9.2 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.

The sections that follow provide example code fragments to illustrate each step. To see how the fragments are submitted in an actual program, see “Sample Program” on page 37.

The examples use the remote version of the SAS Java Metadata Interface. If you recall, the remote version supports single or multiple users and enables clients to persist objects in a remote environment, such as the SAS middle tier. The static version of the SAS Java Metadata Interface supports a single user and does not support objects in the SAS middle tier.

An object factory is needed for each user who will use an application; therefore, typically, 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.

The remote version of the SAS Java Metadata Interface provides the MdFactory interface for instantiating an object factory. The static version provides an MdObjectFactory class for that purpose.

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 remote version of the 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. The makeOMRConnection method connects to the SAS Metadata Server and provides access to com.sas.metadata.remote methods for reading, writing, and updating metadata objects that represent application elements. The com.sas.metadata.remote interface is a Java implementation of the SAS Open Metadata Interface IOMI server interface.

Example of Connecting to the SAS Metadata Server with the makeOMRConnection Method

An object factory is instantiated once per use. Each user has his own object factory instance. The following code instantiates an object factory and creates a connection to the SAS Metadata Server using the makeOMRConnection method:

```
/**
 * The following statements instantiate the object factory.
 */
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, we have the following:

- An object factory in which to create Java 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.

We can now get information about repositories defined on the SAS Metadata Server and 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. The getRepositories method exists in the MdOMIUtil interface.

The following code issues a getRepositories method:

```
/**
 * This example retrieves a list of the repositories that are registered
```

```
     * on the SAS Metadata Server.
     * @return the list of available repositories (list of CMetadata objects)
     */
    public List getRepositories()
    {
       try
       {
          // Repositories are listed with the getRepositories method.
          System.out.println("\nThe repositories contained on this
SAS Metadata Server are: ");

          MdOMIUtil omiUtil = _factory.getOMIUtil();
          List reposList = omiUtil.getRepositories();
          Iterator iter = reposList.iterator();
          while (iter.hasNext())
          {
             // Print the name and id of each repository.
             CMetadata repository = (CMetadata) iter.next();
             System.out.println("Repository: " +
                                repository.getName()
                                + " (" + repository.getFQID() +")");
          }
          return reposList;
       }
       catch (MdException e)
       {
          e.printStackTrace();
       }
       catch (RemoteException e)
       {
          e.printStackTrace();
       }
       return new ArrayList();
    }
```

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. Method parameters take in a CMetadata object to allow the methods to be used with any SAS Open Metadata Interface metadata object.

Creating Objects

You can create 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 update or delete in a SAS Metadata Repository. You must also create an object store in which to hold the metadata objects. The object store maintains a list of the 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 TextStore object. The code then creates associations between these objects. After the metadata objects are created, they are persisted to the SAS Metadata Server.

Notes:

- To persist a 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. Or, you can use methods from the CMetadata interface. The CMetadata interface enables you to determine the identifier of a target repository, and then reference it in the createComplexMetadataObject method as a variable.
- Because these are new metadata objects, they are assigned metadata object identifiers when they are persisted to the SAS Metadata Server. A request that creates Java objects to represent existing metadata objects needs to determine their metadata object instance identifiers before persisting changes to the SAS Metadata Server. For more information, see "Getting and Updating Existing Objects" on page 33.

```
/**
 * This example creates a table, column, and a note on the column using
 * the store methods.
 * @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 reposFQID 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 all our objects.
         // This will be used to maintain a list of objects to persist
         // to the SAS Metadata Server.
         MdObjectStore store = _factory.createObjectStore();

         // Create a PhysicalTable object named "TableTest".
         PhysicalTable table=(PhysicalTable) _factory.createComplexMetadataObject(
```

```
                store,
                null,
                "TableTest",
                MetadataObjects.PHYSICALTABLE,
                shortReposID
               );

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

      // Set the attributes of the column.
      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 note for the column named "NoteTest".
      TextStore note = (TextStore) _factory.createComplexMetadataObject(
              store,
              null,
              "NoteTest",
              MetadataObjects.TEXTSTORE,
              shortReposID
             );

      // Set the StoredText= attribute for the note
      note.setStoredText("Information about the note");

      // Associate the note with the column.
      column.getNotes().add(note);

      // Now, persist all of these changes to the
      // SAS Metadata 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();
        }
     }
  }
```

For more information about object stores and AssociationList objects, see "SAS Java Metadata Interface Overview" on page 21.

Getting and Updating Existing Objects

To update an existing metadata object, you must know its metadata object instance identifier. The SAS Java Metadata Interface provides several ways for getting information about existing metadata objects. This section provides an example of one way you can get information about the metadata objects created in "Creating Objects" on page 31. 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 objects that are returned by the request. In the example that follows, the <TEMPLATES> and <XMLSELECT> elements and their corresponding flags are used to get all PhysicalTable objects named "TableTest," their associated Column and Note metadata objects, and specific attributes of all of the objects.

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 used to denote XML attribute values, to indicate that they should be treated as characters. In this example, additional quotation marks are escaped by using a backslash (\) character. For example, ""is specified as \"\". △

The objects are returned in an object store and can be edited.

```
/**
 * This example reads the newly created objects from the 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 GetMetadataObjectsSubset options
         // strings. These XML strings are used in conjunction with SAS Open
         // Metadata Interface flags. The <XMLSELECT> element specifies filter
         // criteria. The <TEMPLATES> element specifies the metadata properties
```

```
// to be returned for each object from the server.
String xmlSelect = "<XMLSELECT Search=\"@Name='TableTest'\"/>";
String template =
   "<Templates>" +
      "<PhysicalTable Id=\"\" Name=\"\" Desc=\"\">" +
        "<Columns/>" +
      "</PhysicalTable>" +
      "<Column Id=\"\" Name=\"\" Desc=\"\" ColumnName=\"\">" +
         "<Notes/>" +
      "</Column>" +
      "<TextStore Id=\"\" Name=\"\" Desc=\"\" StoredText=\"\"/>" +
    "</Templates>";

// Add the xmlSelect  and  template 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
// (MdObjectFactory.PHYSICALTABLE) using the OMI_XMLSELECT,
// OMI_TEMPLATE, and OMI_GET_METADATA flags. OMI_GET_METADATA
// tells the server to get  all of the attributes specified in the
// template for each object that is returned. The table, column, and
// note will be read from the server and created within the specified
// object store.
int flags = MdOMIUtil.OMI_XMLSELECT | MdOMIUtil.OMI_TEMPLATE |
              MdOMIUtil.OMI_GET_METADATA;
List tableList = _factory.getOMIUtil().getMetadataObjectsSubset(
        store,
        repository.getFQID(),
        MetadataObjects.PHYSICALTABLE,
        flags,
        sOption
        );            Iterator iter = tableList.iterator();
while (iter.hasNext())
{
   // Print the Id= and Name= of the table returned from the server
   PhysicalTable table = (PhysicalTable) iter.next();
   System.out.println("Found table: " + table.getName() + " (" +
                       table.getId() + ")");

   // Get the columns for this table.
   AssociationList columns = table.getColumns();
   for (int i = 0; i < columns.size(); i++)
   {
      // Print the Id= and Name= of associated columns
      Column column = (Column) columns.get(i);
      System.out.println("Found column: " + column.getName() + "
                          (" + column.getId() + ")");
      System.out.println("\tDescription: " + column.getDesc());
      System.out.println("\tColumnName: " + column.getColumnName());

      // Get notes associated with the columns.
      AssociationList notes = column.getNotes();
      for (int j = 0; j < notes.size(); j++)
```

```
                    {
                       // Print the Id=, Name=, and StoredText= values of associated
                      //  notes
                       TextStore note = (TextStore) notes.get(j);
                       System.out.println("Found textstore: " + note.getName() + "
                                          (" +note.getId() + ")");
                           System.out.println("\tStoredText: " + note.getStoredText());
                    }
                 }
              }

              // 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.B8001K8S)
Found column: ColumnTest (A5PCE796.B5005N66)
            Description: This is a description of a column
            ColumnName: MyTestColumnName
Found textstore: NoteTest (A5PCE796.A3002BIS)
            StoredText: Information about the note
```

The output prints the metadata type, name, and metadata object identifier of the PhysicalTable, Column, and TextStore objects. In addition, it prints the values of the Column object's Description= and ColumnName= attributes, and it prints the value of the TextStore object's StoredText= attribute.

For more information about the search criteria supported in the <XMLSELECT> element, see Chapter 15, "Filtering a GetMetadataObjects Request," on page 277. For more information about the templates supported in the <TEMPLATES> element, see "Using Templates" on page 260.

Deleting Objects

This section provides an example of how to delete metadata objects. Similar to updating objects, you must create Java objects that represent the server metadata objects on the client before you can delete them. When you delete an object, all of its dependent objects are automatically deleted as well. A dependent object is an object that has a 1:1 cardinality with the specified object and cannot exist independently of the object. An example of a dependent object is a Column object. A Column object cannot exist on the SAS Metadata Server independently of some type of table object.

See the DataTable metadata type in *SAS 9.2 Metadata Model: Reference* for a listing of the supported table subtypes.

In this example, we use the getMetadataObjectsSubset method to get the objects that we created and updated in "Creating Objects" on page 31, and in "Getting and Updating Existing Objects" on page 33. We use the deleteMetadataObjects method to delete them. 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 deleteList = new ArrayList();

         // Query for the table again
         String xmlSelect = "<XMLSELECT Search=\"@Name='TableTest'\"/>";
         String template = "<Templates>" +
                           "<PhysicalTable>" +
                             "<Columns/>" +
                           "</PhysicalTable>" +
                           "<Column>" +
                              "<Notes/>" +
                           "</Column>" +
                         "</Templates>";

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

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

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

```
                // Get the columns
                AssociationList columns = table.getColumns();
                for (int i = 0; i < columns.size(); i++)
                {
                   Column column = (Column) columns.get(i);
                   deleteList.add(column);

                   // Get the notes
                   AssociationList notes = column.getNotes();
                   for (int j = 0; j < notes.size(); j++)
                   {
                      TextStore note = (TextStore) notes.get(j);
                      deleteList.add(note);
                   }
                }
             }

             // 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 they
             // are 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."

Sample Program

The following is an example of an executable file that contains the code examples from this section.

```
/**
 * Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513
 */
```

```
package com.sas.metadata.remote.test;

import java.rmi.RemoteException;
import java.util.ArrayList;
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.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.TextStore;


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

   /**
    * The following statements instantiate the object factory.
    */
   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;
}

/**
 * This example displays the status of the SAS Metadata Server.
 */
public void displayServerInformation()
{
   try
   {
      MdOMRConnection connection = _factory.getConnection();

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

/**
 * This example retrieves a list of the repositories that are registered
 * on the SAS Metadata Server.
 * @return the list of available repositories (list of CMetadata objects)
 */
public List getRepositories()
{
   try
   {
      // Repositories are listed with the getRepositories method.
      System.out.println("\nThe repositories contained on this server are: ");
```

```
      MdOMIUtil omiUtil = _factory.getOMIUtil();
      List reposList = omiUtil.getRepositories();
      Iterator iter = reposList.iterator();
      while (iter.hasNext())
      {
         // Print the Name= and Id= of each repository.
         CMetadata repository = (CMetadata) iter.next();
         System.out.println("Repository: " + repository.getName() + "
                            (" + repository.getFQID() +")");
      }         return reposList;
   }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
   {
      e.printStackTrace();
   }
   return new ArrayList();
}

/**
 * This example lists the metadata types available on the SAS Metadata Server
 * and their descriptions.
 */
public void displayMetadataTypes()
{
   try
   {
      // Metadata types are listed with the getTypes method.
      System.out.println("\nThe object types contained on this server are: ");
      List nameList = new ArrayList(100);
      List descList = new ArrayList(100);
      _factory.getOMIUtil().getTypes(nameList, descList);
      Iterator iter1 = nameList.iterator();
      Iterator iter2 = descList.iterator();
      while (iter1.hasNext() && iter2.hasNext())
      {
         // Print the name and description of each metadata object type
         String name = (String) iter1.next();
         String desc = (String) iter2.next();
         System.out.println("Type: " + name + ", desc: " + desc);
      }      }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
   {
      e.printStackTrace();
   }
}
```

```
/**
 * This example creates a table, column, and a note on the column.
 * @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 reposFQID 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 all our objects.
         // This will be used to maintain a list of objects to persist
         // to the server.
         MdObjectStore store = _factory.createObjectStore();

         // Create a PhysicalTable object named "TableTest".
         PhysicalTable table=(PhysicalTable)_factory.createComplexMetadataObject(
                  store,
                  null,
                  "TableTest",
                  MetadataObjects.PHYSICALTABLE,
                  shortReposID
                  );

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

         // Set the attributes of the column.
         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 note for the column named "NoteTest".
            TextStore note = (TextStore) _factory.createComplexMetadataObject(
                     store,
                     null,
                     "NoteTest",
                     MetadataObjects.TEXTSTORE,
                     shortReposID
                     );

            // Set the StoredText= attribute for the note
            note.setStoredText("Information about the note");

            // Associate the note with the column.
            column.getNotes().add(note);

            // 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();
         }
      }
   }

   /**
    * This example reads the newly created objects from the 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 GetMetadataObjectsSubset options
            // strings. These XML strings are used in conjunction with SAS Open
            // Metadata Interface flags. The <XMLSELECT> element specifies
            // filter criteria. The <TEMPLATES> element specifies the metadata
```

```
// properties to be returned for each object from the server.
String xmlSelect = "<XMLSELECT Search=\"@Name='TableTest'\"/>";
String template =
   "<Templates>" +
      "<PhysicalTable Id=\"\" Name=\"\" Desc=\"\">" +
        "<Columns/>" +
      "</PhysicalTable>" +
      "<Column Id=\"\" Name=\"\" Desc=\"\" ColumnName=\"\">" +
         "<Notes/>" +
      "</Column>" +
      "<TextStore Id=\"\" Name=\"\" Desc=\"\" StoredText=\"\"/>" +
    "</Templates>";

// Add the xmlSelect and template 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
// (MdObjectFactory.PHYSICALTABLE) using the OMI_XMLSELECT,
// OMI_TEMPLATE, and OMI_GET_METADATA flags. OMI_GET_METADATA
// tells the server to get all of the attributes specified in the template
// for each object that  is returned. The table, column, and note will be
// read from the server and created within the specified object store.
int flags = MdOMIUtil.OMI_XMLSELECT | MdOMIUtil.OMI_TEMPLATE |
               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 Id=  and Name= of the table returned from the server
   PhysicalTable table = (PhysicalTable) iter.next();
   System.out.println("Found table: " + table.getName() + "
                        (" + table.getId() + ")");

   // Get the columns for this table.
   AssociationList columns = table.getColumns();
   for (int i = 0; i < columns.size(); i++)
   {
      // Print the Id= and Name= of associated columns
      Column column = (Column) columns.get(i);
      System.out.println("Found column: " + column.getName() + "
                             (" + column.getId() + ")");
      System.out.println("\tColumnName: " + column.getColumnName());

      // Get notes associated with the columns.
      AssociationList notes = column.getNotes();
      for (int j = 0; j < notes.size(); j++)
      {
```

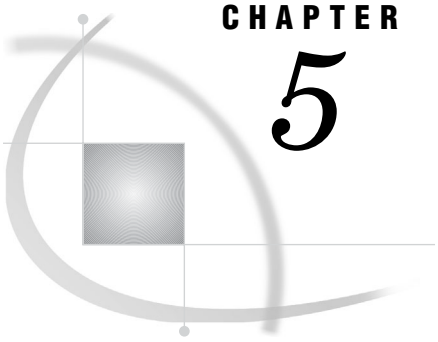
```
                    // Print the Id=, Name=, and StoredText= values of associated
                    //  notes
                    TextStore note = (TextStore) notes.get(j);
                    System.out.println("Found textstore: " + note.getName() + "
                            ("+ note.getId() + ")");
                          System.out.println("\tStoredText: " + note.getStoredText());
                }
             }
          }

          // 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();
       }
    }
}

/**
 * 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 deleteList = new ArrayList();

         // Query for the table again
         String xmlSelect = "<XMLSELECT Search=\"@Name='TableTest'\"/>";
         String template = "<Templates>" +
                              "<PhysicalTable>" +
                                "<Columns/>" +
                              "</PhysicalTable>" +
                              "<Column>" +
                                 "<Notes/>" +
                              "</Column>" +
                            "</Templates>";

         // Add the xmlSelect and template strings together
```

```
      String sOptions = xmlSelect + template;

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



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

         // Get the columns
         AssociationList columns = table.getColumns();
         for (int i = 0; i < columns.size(); i++)
         {
            Column column = (Column) columns.get(i);
            deleteList.add(column);

            // Get the notes
            AssociationList notes = column.getNotes();
            for (int j = 0; j < notes.size(); j++)
            {
               TextStore note = (TextStore) notes.get(j);
               deleteList.add(note);
            }
         }
      }

      // 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 they
      // are no longer being used.
      store.dispose();
   }
   catch (MdException e)
   {
      e.printStackTrace();
   }
   catch (RemoteException e)
```

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

/**
 * This example displays the PhysicalTable objects in a 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 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();
   }
}

/**
 * This example gets 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("\nGetting information for a specific PhysicalTable");
      // Create a template to get detailed information for this table
      String template = "<Templates>" +
                          "<PhysicalTable>" +
                            "<Columns/>" +
                            "<Notes/>" +
                            "<Keywords/>" +
                          "</PhysicalTable>" +
                        "</Templates>";

      // Use the OMI_ALL_SIMPLE flag to get all of the table's attributes.
      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("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
      List repositories = tester.getRepositories();
      CMetadata repos = (CMetadata) repositories.get(0);

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

      // Create a new PhysicalTable object and add it to the server
      tester.createTable(repos);

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

      System.exit(1);
   }

}
```



CHAPTER

5

Understanding com.sas.metadata.remote 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 interface	The starting point for all Java clients. This interface enables you to create the other objects needed to connect to the SAS Metadata Server and interact with its metadata objects. In the static version, this interface is called MdObjectFactory.
MdOMRConnection interface	Contains methods for connecting to the SAS Metadata Server. In the static version, this interface is called MetadataWorkspace.
CMetadata interface	Specifies the base interface that is used to describe SAS Open Metadata Interface objects.

Class or Interface Name	Description
MdOMIUtil interface	Contains methods for getting and setting SAS Metadata Server objects and their attributes. In the static version, this interface is called MetadataUtil.
AssociationList class	Contains methods for defining and maintaining associations.
MdObjectStore interface	Specifies the container for created or modified metadata objects.
MdUtil interface	Contains generic methods, such as debugging methods. In the static version, this interface is called Util.

Working with the MdFactory Interface

The MdFactory interface instantiates the Java object factory and provides methods for creating and deleting Java objects, for invoking the SAS Java Metadata Interface event-handling interface and messaging mechanisms, and for deleting the object factory.

Instantiating the Object Factory

The object factory is instantiated one time only for a SAS Java Metadata Interface client before any other tasks are performed with the getInstance method:

```
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. In this way, the factory will behave as if it is running in a local, single JVM environment.

Creating Java 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 objects on the client. MdFactory provides the createComplexMetadataObject method for creating objects. The createComplexMetadataObject method creates a complex 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 that will be persisted to the SAS Metadata Server.

The following are examples of the createComplexMetadataObject method. A method call that creates a complex object describing a new metadata object has the following form:

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

A method call that creates a complex object representing an existing object on the SAS Metadata Server has the following form:

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

You can get the identifiers of all repositories registered 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 66.

Invoking the Event Handling Interface

The MdFactoryListener interface includes the MdFactoryEvent class and the addMdFactoryListener method. MdFactoryListener is a newer version of the SAS 9.1 MdObjectFactoryListener interface, which is deprecated in SAS 9.2.

You can have multiple listeners in a factory. The addMdFactoryListener method can be instantiated either directly before or after the server connection is made. 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).

If a listener is added to a factory, it must be removed at the end of the factory 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 object from the server and client, as well as its object store.

A new object that was created on the client and persisted to the SAS Metadata Server can also be deleted from its object store by calling the CMetadata delete method. The CMetadata delete method marks the client object as deleted. The client object is deleted the next time the updateMetadataAll method is called on the object store.

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.

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.
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 9.2 returns the value 9200, which represents version 9.2.0.0.
getServerModelVersion	Gets the SAS Metadata Model version number in the form XX.XX. (For example, 11.03.)
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 intermediate 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, FQID, 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 in the SAS Metadata Server.

Working with the MdOMIUtil Interface

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

The MdOMIUtil interface includes AddMetadata, UpdateMetadata, and DoRequest methods. The AddMetadata and UpdateMetadata methods enable you to pass XML-formatted metadata property strings 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 method calls. 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.

Method Name	Description
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 a subset of the 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/92api.

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 two getMetadataObjects methods support the OMI_XMLSELECT flag and the <XMLSELECT> element to pass a search string. All four getMetadata* methods support the OMI_TEMPLATE flag and the <TEMPLATES> element to enable you to specify the attributes and associations to retrieve in a property string.

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 6, "Metadata Access (IOMI Interface)," on page 61. For usage information, see Chapter 12, "Overview of Querying Metadata," on page 235, Chapter 13, "Using GetMetadata to Get the Properties of a Specified Metadata Object," on page 243, Chapter 14, "Using GetMetadataObjects to Get All Metadata of a Specified Metadata Type," on page 263, and Chapter 15, "Filtering a GetMetadataObjects Request," on page 277.

To make the flags easier to use, the MdOMIUtil interface defines constant values for each of the flags. These constants are in the documentation at support.sas.com/92api. 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 AssociationList object is created on the client, but it is empty.
- 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 Java objects representing both sides of the bidirectional association.

Working with the MdObjectStore Interface

All Java 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 and by the event handling interface to track changes in object stores.

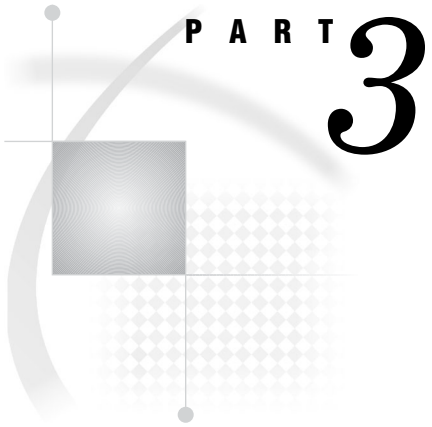
An object store is created with the statement:

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

An object store is deleted with the MdObjectStore dispose method. 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.



PART 3

Server Interfaces

CHAPTER

6

Metadata Access (IOMI Interface)

Overview of the IOMI Server Interface

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.

In SAS 9.2, 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.

Using the IOMI Server Interface

Using the IOMI server interface is a matter of using its methods. To access these 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*.

Introduction to IOMI Methods

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

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 64.

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

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 metadata type that you want to read or write, enclosed in angle brackets. 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>
```

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

The second nested element, Column, is the *association subelement*. The association 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 only interested in associated objects of this metadata type.

All in all, the nested elements in the example specify that the main metadata object, PhysicalTable, has a Columns association to an object of metadata type Column.

CAUTION:
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 white space 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.

Category	Method	Description
Read Methods	GetMetadata	Gets specified properties for a specified metadata object
	GetMetadataObjects	Gets all metadata objects of the specified metadata type from the specified repository
Repository Methods	GetRepositories	Gets the metadata repositories on the SAS Metadata Server
Write Methods	AddMetadata	Adds metadata objects to a repository
	DeleteMetadata	Deletes metadata objects from a repository
	UpdateMetadata	Updates metadata objects in a repository
Messaging Method	DoRequest	Executes XML-formatted method calls
Management Methods	GetNamespaces	Gets the namespaces defined on the SAS Metadata Server
	GetSubtypes	Gets all possible subtypes for a specified metadata type
	GetTypes	Gets all of the metadata types in a namespace
	GetTypeProperties	Gets all possible properties for a specified metadata type
	IsSubtypeOf	Determines whether one metadata type is a subtype of another metadata type
User Interface Helper Methods	AddResponsibleParty	Creates a ResponsibleParty object for the specified identity in the repository that contains the identity's metadata definition
	AddUserFolders	Creates a user's home folder and subfolders
	GetResponsibleParty	Gets the ResponsibleParty object associated with the specified Person or IdentityGroup and responsibility
	GetUserFolders	Gets a user's home folder or subfolders

Using 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. See "Summary Table of

IOMI Flags" on page 68 for a list of the available flags and the methods for which they are supported.

Specifying a Flag

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

Corresponding XML Elements

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> element in the OPTIONS parameter. 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 in the OPTIONS parameter. See "Summary Table of IOMI Options" on page 74 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:

Flag name	Numeric Indicator	Method	Description
OMI_ALL	1	GetMetadata GetRepositories GetTypeProperties	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	Gets the descendants of the returned subtypes and the subtypes.
OMI_ALL_SIMPLE	8	GetMetadata	Gets all of the attributes of the requested object.
OMI_CI_DELETE_ALL	33		Obsolete in SAS 9.2.
OMI_CI_NODELETE	524288		Obsolete in SAS 9.2.

Flag name	Numeric Indicator	Method	Description
OMI_DELETE	32	DeleteMetadata	Deletes the contents of a repository and the repository's registration.
OMI_DEPENDENCY_USED_BY	16384	GetMetadata GetMetadataObjects	New SAS 9.2 behavior. 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	New SAS 9.2 behavior: Specifies to include associations to objects from all production repositories (the foundation repository and all custom repositories) in the method results.
OMI_GET_METADATA	256	GetMetadataObjects	Executes a GetMetadata call for each object that is returned by the GetMetadataObjects method.
OMI_IGNORE_NOTFOUND	134217728	DeleteMetadata UpdateMetadata	Prevents a delete or update operation from being aborted when a request specifies to delete or update an object that does not exist.

Flag name	Numeric Indicator	Method	Description
OMI_INCLUDE_SUBTYPES	16	GetMetadata GetMetadataObjects	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	Locks the specified object and any associated objects selected by GetMetadata flags and options from update by everyone except the caller.
OMI_LOCK_TEMPLATE	65536		Obsolete in SAS 9.2.
OMI_MATCH_CASE	512	GetMetadataObjects	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_NO_DEPENDENCY_CHAIN	16777216		Obsolete in SAS 9.2.
OMI_NOFORMAT	67108864	GetMetadata	Causes date, time, and datetime values in the output XML stream to be returned as raw SAS date, SAS time, and SAS datetime floating-point values. Without the OMI_NOFORMAT flag, the default US-English locale is used to format the values into recognizable character strings.
OMI_PURGE	1048576		Obsolete in SAS 9.2.

Flag name	Numeric Indicator	Method	Description
OMI_REINIT	2097152	DeleteMetadata	Deletes the contents of a repository, but does not remove the repository's registration from the SAS Repository Manager.
OMI_RETURN_LIST	1024	DeleteMetadata UpdateMetadata	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.
OMI_SUCCINCT	2048	GetMetadata GetTypes	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 241.

Flag name	Numeric Indicator	Method	Description
OMI_TEMPLATE	4	DeleteMetadata GetMetadata	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 metadata property string in a <TEMPLATES> element. For more information about GetMetadata usage, see "Using Templates" on page 260. For more information about DeleteMetadata usage, see Chapter 17, "Deleting Metadata Objects," on page 297.
OMI_TRUNCATE	4194304	DeleteMetadata	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.
OMI_TRUSTED_CLIENT	268435456	AddMetadata DeleteMetadata UpdateMetadata	Determines whether the client can call this method.
OMI_UNLOCK	131072	UpdateMetadata	Unlocks an object lock that is held by the caller.

Flag name	Numeric Indicator	Method	Description
OMI_UNLOCK_FORCE	262144	UpdateMetadata	Unlocks an object lock that is held by another user.
OMI_XMLSELECT	128	GetMetadataObjects	Checks the OPTIONS parameter for search criteria that filters the objects that are returned. The search criteria are passed as a search string in an <XMLSELECT> element. For more information, see Chapter 15, "Filtering a GetMetadataObjects Request," on page 277.

Summary Table of IOMI Options

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

Option	Flag	Method	Description
<DOAS>	None	AddMetadata DeleteMetadata GetMetadata GetMetadataObjects GetSubtypes GetTypeProperties IsSubtypeOf UpdateMetadata	Enables a client to make a request on behalf of another user. For more information, see "<DOAS> Option" on page 75.
<REPOSID>	OMI_SUCCINCT (2048)	GetTypes	Specifies the repository ID of the repository whose metadata you want to evaluate. See "Using GetTypes to Get Actual Metadata Types in a Repository" on page 241.

Option	Flag	Method	Description
<TEMPLATES>	OMI_TEMPLATE (4)	DeleteMetadata GetMetadata	In DeleteMetadata, submits templates that specify associated objects to delete with the specified metadata object. For more information, see Chapter 17, "Deleting Metadata Objects," on page 297. In GetMetadata, specifies additional properties to retrieve for the specified metadata object beyond those specified in the INMETADATA parameter and by GetMetadata flags. See "Using Templates" on page 260.
<XMLSELECT>	OMI_XMLSELECT (128) and OMI_MATCH_CASE (512)	GetMetadataObjects	Specifies a search string to filter the objects that are retrieved. See Chapter 15, "Filtering a GetMetadataObjects Request," on page 277.

<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*"/>, where *CredHandle* is a handle that is returned by the ISecurity GetCredentials method that represents the other user's credentials. For more information, see "GetCredentials" on page 129.

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

Adds metadata objects to a repository.
Category: Write methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 64.
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 75.

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 Path= in advance, or pass the <CREATEREPOSCONTAINER/> element in the AddMetadata request to create the directory. The Path= attribute accepts an absolute or relative pathname; use backslashes or forward slashes (\ and /) to indicate the directory levels. The <CREATEREPOSCONTAINER/> element is new in SAS 9.2.

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 64.

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 10, “Adding Metadata Objects,” on page 205.

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= properties.

```
<!-- 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" on page 110
- "GetRepositories" on page 97

AddResponsibleParty

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

Category: User interface helper methods

Syntax

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

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

In SAS 9.2, 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 RESPONSIBILITY= 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 IDENTITYNAME= or RESPONSIBILITY=, 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" on page 87
- "GetResponsibleParty" on page 100

AddUserFolders

Creates a user's home folder and subfolders.
Category: User interface helper methods

Syntax

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

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 9.2 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 forwardslash (/) 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 or 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" will not additionally create subfolders. However, a request to create any of the subfolders ("My Folder" or "Application Data") will additionally 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 also 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" on page 87.

The AddUserFolders method uses the security policy defined for the foundation repository to determine where to create the folders. Beginning in SAS 9.2, 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 /Users 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 will not create a repository or /Users folder for you.

A successful AddUserFolders request creates a subfolder in the /Users folder that has the name specified in the PERSONNAME= parameter. The request also 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 display 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 display 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 administrators only 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 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 the /Users folder will remain in this folder, and home folders for new users will be 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 is 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" on page 87
- "GetUserFolders" on page 107

DeleteMetadata

Deletes metadata objects from a repository.
Category: Write methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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 aborted 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.
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 75. <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. However, it can 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 300. 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 SAS 9.2, a DeleteMetadata method issued in the SAS namespace deletes the specified object and associated objects that are specified in a template when the OMI_TEMPLATE flag is set. A template is submitted in a <TEMPLATE> element within the <TEMPLATES> element. The use of a <TEMPLATE> element within the <TEMPLATES> element is unique to DeleteMetadata. It is supported to enable multiple objects and their associated objects to be deleted by the DeleteMetadata method.

For usage information about deleting SAS namespace metadata objects, see Chapter 17, "Deleting Metadata Objects," on page 297.

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" on page 77
- "UpdateMetadata" on page 110

DoRequest

Executes XML-formatted method calls.
Category: Messaging method

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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>
   <Parameter n>Value</Parameter n>
</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. In SAS 9.2, methods that call both 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

Gets specified properties for the specified metadata object.
Category: Read methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
inMetadata	C	in	Metadata property string that identifies the object and properties 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 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 in its 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, 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. 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_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. 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_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 all properties that do not contain a value or that contain a null value. OMI_TEMPLATE=4 Checks the OPTIONS parameter for user-defined templates that define which metadata properties to return. The user-defined templates are submitted in a <TEMPLATES> element in the OPTIONS parameter.

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 75. <TEMPLATES> Specifies properties to retrieve for the specified metadata type beyond properties already specified in the INMETADATA parameter and by GetMetadata flags. The <TEMPLATES> element must be specified with the OMI_TEMPLATE flag.

Details

The GetMetadata method gets properties for the specified metadata object.

The method provides several ways to identify the properties that you want to retrieve. For usage information, see Chapter 13, "Using GetMetadata to Get the Properties of a Specified Metadata Object," on page 243.

In previous releases of SAS, when information was requested about associated objects, the method returned information about associated objects that were stored in the same repository as the requested object by default, and the user had to set a flag to get information about associated objects in other repositories. Beginning in SAS 9.2, a GetMetadata method that requests associated objects that is issued in a public repository (the foundation or a custom repository) returns associated objects from all public repositories by default. You set the OMI_DEPENDENCY_USED_BY flag only if you want to include associated objects that are in project repositories in the results.

When GetMetadata is issued in a project repository, it always returns associated objects that are in the same repository.

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.

The OMI_LOCK (32768) flag is one of several multi-user concurrency controls supported by the SAS Open Metadata Interface. The flag enables you to lock the specified object and any associated objects selected by GetMetadata flags and options from use by other users. Metadata objects that are locked with OMI_LOCK are unlocked by issuing an UpdateMetadata method call that sets the OMI_UNLOCK or OMI_UNLOCK_FORCE flag. For an overview of the concurrency controls supported by the SAS Open Metadata Interface, see Chapter 16, "Metadata Locking Options," on page 295.

The OMI_INCLUDE_SUBTYPES flag extends template processing 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 objects. For more information, see "Using GetMetadata to Get Common Properties for Sets of Objects" on page 254.

Some GetMetadata flags have interdependencies that can affect the metadata that is returned when more than one flag is set. For more information, see "Using IOMI Flags" on page 67.

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 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” on page 93

GetMetadataObjects

Gets all metadata objects of the specified metadata type in the specified repository.
Category: Read methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see “Return Code” on page 64.
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 repository and custom repositories) in the method results. OMI_NO_DEPENDENCY_CHAIN=16777216 Obsolete in SAS 9.2. 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 75. <XMLSELECT> Specifies a search string to filter the objects that are retrieved. See Chapter 15, “Filtering a GetMetadataObjects Request,” on page 277 for more information.

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 information 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 Chapter 14, "Using GetMetadataObjects to Get All Metadata of a Specified Metadata Type," on page 263, and Chapter 15, "Filtering a GetMetadataObjects Request," on page 277.

The behavior of the OMI_DEPENDENCY_USED_BY and OMI_DEPENDENCY_USES flags is different in SAS 9.2 than in previous versions of SAS. In SAS 9.2, set OMI_DEPENDENCY_USES to get metadata objects from all public repositories (the foundation repository and all custom repositories) in the method results, and to get metadata objects from the specified repository. Set OMI_DEPENDENCY_USED_BY to get metadata objects of the specified metadata type from all project repositories in the method results, and to get 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 or 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 gets metadata objects from the repositories specified by the flags. A request that specifies to get objects from all registered repositories returns a specified repository first, followed by the foundation repository, followed by custom repositories in the order they were registered, and then followed by project repositories in the order they were registered.

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

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, and 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" on page 89

GetNamespaces

Gets the namespaces defined on the SAS Metadata Server.
Category: Management methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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.
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" on page 105
- "GetSubtypes" on page 102

GetRepositories

Gets the metadata repositories on the SAS Metadata Server.
Category: Repository methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
repositories	C	out	Returned list of all repositories that are registered on the SAS Metadata Server.
flags	L	in	OMI_ALL=1 Gets information about repository location, format, type and availability in addition to listing 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 information, the description, and the default namespace for the SAS Repository Manager and each repository that is registered in the SAS Repository Manager.

When issued with the OMI_ALL (1) flag set, the GetRepositories method also gets the following attributes for each repository:

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

RepositoryFormat= "*number*"
a numeric double value indicating the format level of the repository. (For example, 11.0.)

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

Access= "OMS_FULL | OMS_READONLY | OMS_ADMIN | OMS_OFFLINE"
a descriptor that indicates the access mode that the administrator set for the repository.

OMS_FULL
Specifies the repository is available to all users for read and write access.

OMS_READONLY

Specifies the repository is only to be read.

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

OMS_OFFLINE
Specifies the repository is unavailable to all users.

PauseState= "*empty-string* | ADMIN | ADMIN(READONLY) | OFFLINE"
Reports a repository state change as the result of a server pause. This attribute is set by the Pause method and cleared by the Resume method. The value is usually the server Pause value (ADMIN or OFFLINE), unless the repository is registered with a less restrictive Access= value.

empty string
Indicates the SAS Metadata Server is online. It has not been paused by the Pause method. The repository can be accessed in its intended access mode.

ADMIN
Indicates this repository has been downgraded to an ADMIN state by a server pause. Only users who have administrative status on the server can read and write to this repository.

ADMIN(READONLY)
Indicates this repository has been downgraded to an ADMIN state by a server pause. Its intended state is READONLY. It is available for reading only to users who have administrative status on the server.

OFFLINE
Indicates the repository is not available to any users because the SAS Metadata Server has been paused to an OFFLINE state or the repository is registered with Access="OMS_OFFLINE".

CurrentAccess= "READONLY | OFFLINE"
The SAS Metadata Server manages two copies 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. When a problem is encountered, valid values are READONLY and OFFLINE. When the SAS Metadata Server can access a repository as intended, GetRepositories returns a CurrentAccess= value that matches the repository's Access= attribute.

The additional attributes that are retrieved by OMI_ALL are available in the standard interface and 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.A0000001" Name="REPOSMGR"
      Desc="The Repository Manager"  DefaultNS="REPOS"/>
   <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 Repository Access and Status Information” on page 238.

GetResponsibleParty

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

Category: User interface helper methods

Syntax

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

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 RESPONSIBILITY=. 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" on page 87
- "AddResponsibleParty" on page 79

GetSubtypes

Gets all possible subtypes for a specified metadata type.
Category: Management methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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.

Parameter	Type	Direction	Description
flags	L	in	OMI_ALL_DESCENDANTS=64 Specifies to get the descendants of the returned subtypes and the subtypes.
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 75.

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" on page 105
- "IsSubtypeOf" on page 108

GetTypeProperties

Gets all possible properties for a specified metadata type.
Category: Management methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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 New in SAS 9.2. 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 75.

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" on page 105
- "GetSubtypes" on page 102

GetTypes

Gets all of the metadata types in a namespace.

Category: Management methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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 on 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 241.

```
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" on page 96
- "GetSubtypes" on page 102

GetUserFolders

Gets a user's home folder or subfolders.
Category: User interface helper methods

Syntax

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

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" on page 87
- "AddUserFolders" on page 81

IsSubtypeOf

Determines whether one metadata type is a subtype of another metadata type.
Category: Management methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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.
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 75.

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" on page 102

UpdateMetadata

Updates specified metadata objects in a repository.
Category: Write methods

Syntax

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

Parameters

Parameter	Type	Direction	Description
rc	N	out	Return code for the method. For more information, see "Return Code" on page 64.
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.
ns	C	in	Namespace to use as the context for the request.

Parameter	Type	Direction	Description
flags	L	in	OMI_IGNORE_NOTFOUND = 134217728 Prevents an update operation from being aborted when a request specifies to update an object that does not exist. OMI_RETURN_LIST = 1024 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 75.

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, unless the OMI_IGNORE_NOTFOUND (134217728) flag is set.

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 information about these directives and general UpdateMetadata usage, see Chapter 11, “Updating Metadata Objects,” on page 221.

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 16, “Metadata Locking Options,” on page 295. 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);
```

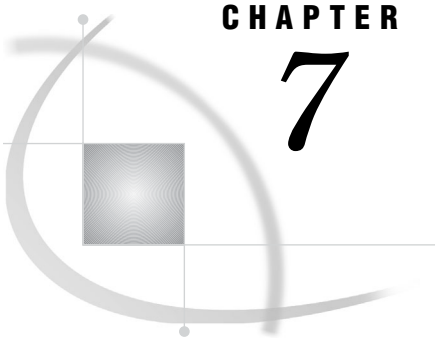
Example 2: DoRequest Method

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” on page 84
- “GetMetadata” on page 89



CHAPTER 7

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.

In SAS 9.2, 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 a SAS 9.2 environment. It also offers several new methods.

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, which are new in SAS 9.2, see Chapter 8, "Security Administration (ISecurityAdmin Interface)," on page 155.

  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, and that the calling user is a trusted user of the SAS Metadata Server.
- 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 9.2 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, import com.sas.meta.SASOMI.ISecurity and com.sas.meta.SASOMI.ISecurityPackage.
- To use SAS 9.2 methods, import com.sas.meta.SASOMI.ISecurity_1_1 and com.sas.meta.SASOMI.ISecurity_1_1Package.

The SAS 9.2 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 in SAS 9.2, 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" on page 129.

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" on page 121.
- The GetIdentity method supports submission of the string LOGINID: *userid* to identity a Person or IdentityGroup. For more information, see "GetIdentity" on page 131.
- 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" on page 132.

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 "SetInternalPassword" on page 153, "SetInternalLoginUserOptions" on page 150, "GetInternalLoginUserInfo" on page 140, and "DeleteInternalLogin" on page 119.

Understanding the ISecurity 1.0 Interface

The ISecurity 1.0 interface includes the following authorization methods:

GetCredentials
: Returns a handle to a provider-specific credential.

FreeCredentials
: Frees the handle returned by GetCredentials.

GetAuthorizations
: Gets authorization information for a resource, depending on the type of authorization requested.

GetIdentity
: Gets identity metadata for the specified user.

IsAuthorized
: Determines whether an authenticated user is authorized to access a resource with a specific permission.

Understanding the ISecurity 1.1 Interface

The ISecurity 1.1 interface contains three categories of methods:

- ISecurity 1.0 authorization methods that were updated for the SAS 9.2 environment
- Internal authentication methods
- Generalized authorization methods

The SAS 9.2 methods support internal SAS Metadata Server authentication when new internal user accounts are used, in addition to the traditional external authentication.

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

GetInternalLoginSitePolicies
: Returns the active server-level internal authentication policies.

SetInternalPassword
: Creates an InternalLogin object for the specified user.

SetInternalLoginUserOptions
: Customizes internal authentication policies for the specified user.

GetInternalLoginUserInfo
: Gets availability information and internal authentication settings for the specified user.

DeleteInternalLogin
: Deletes the InternalLogin object that is associated with the specified user.

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

GetApplicationActionsAuthorizations
: Returns authorizations for ApplicationActions in a SoftwareComponent object.

GetAuthorizationsforObjects
: Gets authorizations for a specified set of objects and permissions.

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.

GetLoginsforAuthDomain
: Retrieves the logins for the connected user for the specified authentication domain in order of identity precedence.

IsInRole
: Returns the TRUE value when the user specified in CREDHANDLE is in a role.

For more information, see the documentation for the individual methods.

DeleteInternalLogin

Deletes the InternalLogin object that is associated with the specified user.
Category: Internal authentication methods
Interface version: ISecurity 1.1

Syntax

```
DeleteInternalLogin(personName);
```

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” on page 150

FreeCredentials

Frees the handle returned by GetCredentials.
Category: Authorization methods
Interface version: ISecurity 1.0

Syntax

```
FreeCredentials(credHandle);
```

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" on page 129

GetApplicationActionsAuthorizations

Returns authorizations for ApplicationActions in a SoftwareComponent object.
Category: Generalized authorization methods
Interface version: ISecurity 1.1

Syntax

```
GetApplicationActionsAuthorizations(credHandle,applicationContext,options,output);
```

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.

Parameter	Type	Direction	Description
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.
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" on page 148

GetAuthorizations

Gets authorization information for a resource, depending on the type of authorization requested.

Category: Authorization methods

Interface version: ISecurity 1.0

Syntax

```
GetAuthorizations(authType,credHandle,resource,permission,authorizations);
```

Parameters

Parameter	Type	Direction	Description
authType	string	in	The type of authorization to perform.
credHandle	string	in	Credential handle identifying a user identity, or an empty string.
resource	string	in	Passed resource identifier.

Parameter	Type	Direction	Description
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. Currently, the only supported AUTHTYPE value is Cube.

Cube returns an array of strings[*][4]. The number of rows depends on the structure of the cube. Each row has the following four columns:

Type
: Indicates the metadata type in the row. This is either Hierarchy, Dimension, 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” on page 145.

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. The value ''Cube'' is
 // the only supported authType. ''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” on page 145

GetAuthorizationsforObjects

Gets authorizations for a specified set of objects and permissions.
Category: Generalized authorization methods
Interface version: ISecurity 1.1

Syntax

```
GetAuthorizationsforObjects(credHandle,permissions,resources,permMask,GRANT,
conditionNDXs,conditionPermMasks,conditions);
```

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 116.

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" on page 123
- "GetApplicationActionsAuthorizations" on page 121

GetCredentials

Returns a handle to a provider-specific credential.
Category: Authorization methods
Interface version: ISecurity 1.0

Syntax

```
GetCredentials(userid,credHandle);
```

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" on page 120
- "GetIdentity" on page 131

GetIdentity

Gets identity metadata for the specified user.
Category: Authorization methods
Interface version: ISecurity 1.0

Syntax

```
GetIdentity(credHandle,identity);
```

Parameters

Parameter	Type	Direction	Description
credHandle	string	in	Credential handle identifying a user identity, an empty string, or a user ID in the form LOGINID:*userid*.
identity	string	out	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:
//   Here the returned Identity corresponds to the credential handle
//   obtained in the previous call to GetCredentials().

 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" on page 129

GetInfo

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: Generalized authorization methods

Interface version: ISecurity 1.1

Syntax

```
GetInfo("infoType",identity,options,output);
```

Parameters

Table 7.1

Parameters	Type	Direction	Description
infoType	string	in	Specifies the identity information to get. Valid values are: □ GetIdentityInfo □ EnterprisePolicies □ SASPW_Alias □ ALL
identity	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= value of the identity.

Parameters	Type	Direction	Description
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

If IDENTITY is an empty string, then "INFOTYPE" requests information for the connected user. If IDENTITY is a credential handle or URN-like identifier, and the connected user is a trusted user, then information is returned for the specified identity. For information about the format of a URN, see "Identifying Resources to ISecurity Methods" on page 116.

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. A connected user must have ReadMetadata permission to the requested Identity object in order to obtain information about it. The following are examples of how the *IdentityType*:*Name* form is used:

```
'Person:Jane'
'IdentityGroup:AccountingDept'
'Role:AccountsPayableClerks'
```

A description of each "INFOTYPE" value and its options follows.

"GetIdentityInfo"

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 *userID 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 *userID 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= 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'.

"EnterprisePolicies"

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.

"SASPW_Alias"

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 startup 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” on page 140

GetInternalLoginSitePolicies

Returns the active server-level internal authentication policies.
Category: Internal authentication methods
Interface version: ISecurity 1.1

Syntax

```
GetInternalLoginSitePolicies(siteMinPasswordLength,siteIsDigitRequired,
siteIsMixedCaseRequired,siteSizeHistoryList,sitePasswordChangeDelayInMinutes,
siteExpirationDays,siteNumFailuresForLockout,siteLockoutInMinutes,
siteDaysToSuspension);
```

Parameters

Parameter	Type	Direction	Description
siteMinPasswordLength	int	out	Specifies the minimum length for passwords in characters.
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.

Parameter	Type	Direction	Description
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" on page 140
- □ "SetInternalLoginUserOptions" on page 150

GetInternalLoginUserInfo

Gets availability information and internal authentication settings for the specified user.

Category: 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

Parameter	Type	Direction	Description
personName	string	in	Specifies the Name= 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.
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.

Parameter	Type	Direction	Description
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" on page 150
- "GetInternalLoginSitePolicies" on page 138

GetLoginsforAuthDomain

Retrieves the logins for the connected user for the specified authentication domain in order of identity precedence.

Category: Generalized authorization methods

Interface version: ISecurity 1.1

Syntax

```
GetLoginsforAuthDomain(credHandle,authDomain,options,output);
```

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.

Parameter	Type	Direction	Description
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.
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" on page 129
- "GetInfo" on page 132

IsAuthorized

Determines whether an authenticated user is authorized to access a resource with a specific permission.

Category: Authorization methods

Interface version: ISecurity 1.0

Syntax

```
IsAuthorized(credHandle,resource,permission,permissionCondition,authorized);
```

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.
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`

- Beginning in SAS 9.2, 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 a 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" on page 123

IsInRole

Returns the TRUE value when the user specified in CREDHANDLE is in a role.
Category: Generalized authorization methods
Interface version: ISecurity 1.1

Syntax

```
IsInRole(credHandle,roleSpec,options,inRole);
```

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*

Parameter	Type	Direction	Description
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 if 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" on page 121

SetInternalLoginUserOptions

Customizes internal authentication policies for the specified user.
Category: Internal authentication methods
Interface version: ISecurity 1.1

Syntax

```
SetInternalLoginUserOptions(personName,isDisabled,bypassStrength,bypassHistory,
useStdPasswordExpirationDate,passwordExpirationDays,bypassLockout,
bypassInactivitySuspension,expireAccount,accountExpirationDate);
```

Parameters

Parameters	Type	Direction	Description
personName	string	in	Specifies the Name= 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.
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" on page 138
- "GetInternalLoginUserInfo" on page 140
- "DeleteInternalLogin" on page 119

SetInternalPassword

Creates an InternalLogin object for the specified user.
Category: Internal authentication methods
Interface version: ISecurity 1.1

Syntax

```
SetInternalPassword(personName,passwordValue);
```

Parameters

Parameter	Type	Direction	Description
personName	string	in	Specifies the Name= 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.

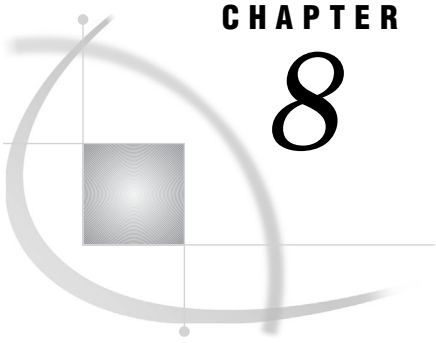
Examples

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" on page 138
- "GetInternalLoginUserInfo" on page 140
- "SetInternalLoginUserOptions" on page 150



CHAPTER 8

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 9.2 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 9.2 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, as described in "Identifying Resources to ISecurity Methods" on page 116.

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 the SAS Management Console Authorization 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" on page 161 and "EndTransactionContext" on page 167.

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.

GetAccessControlTemplatesOnObj
: Lists the ACTs that are associated with a resource.

GetAuthorizationsOnObj
: Returns the authorizations that apply to a resource for specified identities and permissions.

GetIdentitiesOnObj
: Returns Person, IdentityGroup, and Role objects associated with a specified resource.

RemoveACTFromObj
: Removes the authorizations defined by an ACT from the specified resource.

SetAuthorizationsOnObj
: Sets permissions for identities on a resource.

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.

The ACT administration methods include the following:

CreateAccessControlTemplate
: Creates an ACT.

DestroyAccessControlTemplate
: Destroys an ACT and removes references to it from all associated objects.

GetAccessControlTemplateAttribs
: Retrieves the attributes of an ACT.

GetAccessControlTemplateList
: Lists all ACTs in the specified repository or in all public repositories.

SetAccessControlTemplateAttribs
: Changes the attributes of an ACT.

ApplyACTToObj

Applies the authorizations defined in an ACT to the specified resource.
Category: General authorization administration methods

Syntax

```
ApplyACTToObj(tCtxt,resource,flags,ACTresource);
```

Parameters

Parameter	Type	Direction	Description
tCtxt	string	in	Optional handle representing a server-side transaction context.
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
   {
    iSecurityAdmin.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 a value is specified in the
     // resource parameter, 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" on page 163
- "RemoveACTFromObj" on page 181
- "DestroyAccessControlTemplate" on page 165

BeginTransactionContext

Creates a transaction context for an authorization request.
Category: Transaction context methods

Syntax

```
BeginTransactionContext(resource,flags,tCtxt);
```

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" on page 167

CreateAccessControlTemplate

Creates an ACT.
Category: ACT administration methods

Syntax

```
CreateAccessControlTemplate(tCtxt,REPOSresource,ACT_attributes);
```

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=	specifies a description of the ACT. A string up to 200 characters is supported.
Use=	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" on page 183
- "SetAccessControlTemplateAttribs" on page 182
- "DestroyAccessControlTemplate" on page 165

DestroyAccessControlTemplate

Destroys an ACT and removes references to it from all associated objects.
Category: ACT administration methods

Syntax

```
DestroyAccessControlTemplate(tCtxt,ACTresource);
```

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 157.

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" on page 163
- "RemoveACTFromObj" on page 181

EndTransactionContext

Terminates the transaction context for the specified TCTXT handle.
Category: Transaction context methods

Syntax

```
EndTransactionContext(tCtxt,resource,flags);
```

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” on page 161

GetAccessControlTemplatesOnObj

Lists the ACTs that are associated with a resource.
Category: General authorization administration methods

Syntax

```
GetAccessControlTemplatesOnObj(tCtxt,resource,flags,ACT_list);
```

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.
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= value. Column 2: Contains the ACT Description= 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" on page 159
- "RemoveACTFromObj" on page 181
- "CreateAccessControlTemplate" on page 163

GetAccessControlTemplateAttribs

Retrieves the attributes of an ACT.
Category: ACT administration methods

Syntax

```
GetAccessControlTemplateAttribs(tCtxt,ACTresource,ACT_attributes);
```

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.
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 ACT attributes, see "CreateAccessControlTemplate" on page 163.

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" on page 182

GetAccessControlTemplateList

Lists all ACTs in the specified repository or in all public repositories.
Category: ACT administration methods

Syntax

```
GetAccessControlTemplateList(tCtxt,REPOSresource,flags,ACT_list);
```

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.

Parameter	Type	Direction	Description
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).
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= 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= value. Column 3: Contains the ACT's Description= value. Column 4: Contains the ACT's Use= 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" on page 170

GetAuthorizationsOnObj

Returns the authorizations that apply to a resource for specified identities and permissions.

Category: General authorization administration methods

Syntax

```
GetAuthorizationsOnObj(tCtxt,resource,flags,identities,permissions,authorizations);
```

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.
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.

Parameter	Type	Direction	Description
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= 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= 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= 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 8.1 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” on page 183

GetIdentitiesOnObj

Returns Person, IdentityGroup, and Role objects associated with a specified resource.
Category: General authorization administration methods

Syntax

```
GetIdentitiesOnObj(tCtxt,resource,flags,id_List);
```

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 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= value of the identity. Column 2: If both the SECAD_RETURN_IDENTITY_ORIGIN and SECAD_RETURN_DISPLAY_NAME flags are set, contains the DisplayName= 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" on page 173

RemoveACTFromObj

Removes the authorizations defined by an ACT from the specified resource.
Category: General authorization administration methods

Syntax

```
RemoveACTFromObj(tCtxt,resource,flags,ACTresource);
```

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" on page 165
- "ApplyACTToObj" on page 159

SetAccessControlTemplateAttribs

Changes the attributes of an ACT.
Category: ACT administration methods

Syntax

```
SetAccessControlTemplateAttribs(tCtxt,ACTresource,ACT_attributes);
```

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" on page 163.

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" on page 163
- "GetAccessControlTemplateAttribs" on page 170

SetAuthorizationsOnObj

Sets permissions for identities on a resource.
Category: General authorization administration methods

Syntax

```
SetAuthorizationsOnObj(tCtxt,resource,flags,authorizations);
```

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.
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= 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” on page 173
- “CreateAccessControlTemplate” on page 163

CHAPTER

9

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 Status, 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.
- A user must have administrative user status on the SAS Metadata Server to issue all methods except Status. For more information about administrative user status, see the *SAS Intelligence Platform: Security Administration Guide*.
- 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 9.2 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 9.2 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 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.

Refresh
: Changes certain SAS Metadata Server invocation and configuration options on a running server.

Resume
: Returns a paused SAS Metadata Server to an ONLINE state.

Status
: Polls the SAS Metadata Server for status, platform version, SAS Metadata Model version, server locale, server configuration information, and journaling statistics.

Stop
: Shuts down the SAS Metadata Server.

Pause

Temporarily limits the availability of the SAS Metadata Server.
Category: Server control methods

Syntax

```
rc=Pause(options);
```

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 it did, RC=1 means it did not.
options	C	in	<PAUSECOMMENT> Used with the <SERVER> element, this option enables administrators to set a free-form text comment that is retrieved with server status queries. <REPOSITORY> Deprecated in SAS 9.2. <SERVER State="*value*"/> Specifies to limit the availability of the SAS Metadata Server and the desired server state. See the "Details" section for more information.

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. A running SAS Metadata Server supports three states:

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	New in SAS 9.2. Indicates the SAS Metadata Server is available for reading and writing, but only to users who have administrative user status on the SAS Metadata Server. The server is unavailable to other users.
OFFLINE	Indicates the SAS Metadata Server is running, but is temporarily unavailable to all users.

The <SERVER> element is specified as follows:

```
<SERVER State="ADMIN|OFFLINE"/>
```

The State attribute specifies the state to apply to the SAS Metadata Server.

If the Pause method is issued without options, the SAS Metadata Server changes to an OFFLINE state. To change the server to an ADMIN state, you must specify <SERVER State="ADMIN"/> when you issue the method.

The <PAUSECOMMENT> element can be used with the <SERVER> element to enable administrators to set a free-form text comment that is retrieved with server status queries. The <PAUSECOMMENT> element is specified as follows:

```
<PauseComment>This is a test of the Pause method. </PauseComment>
```

The state of the metadata server is obtained by issuing the Status method. For more information, see “Status” on page 194. The <PAUSECOMMENT> element can be added to any exception returned by a method that rejects a user request because of a paused status.

A paused SAS Metadata Server is returned to the ONLINE state by issuing the Resume method. Resume clears the text in the <PAUSECOMMENT> element. When a paused SAS Metadata Server is stopped and restarted, it restarts in an ONLINE state. A server pause is not persisted between server sessions.

In SAS 9.2, the Pause method can no longer be used to limit the availability of specific metadata repositories. To limit a metadata repository’s availability, use the UpdateMetadata method to change the value in the repository’s Access= attribute. If a Pause method is issued that specifies the <REPOSITORY> option, the SAS Metadata Server returns an error.

Examples

The following example pauses the SAS Metadata Server to an OFFLINE state:

```
<! --- Pause the server to OFFLINE  --->
options=' ';

rc=serverObject.Pause(options);
```

OFFLINE is the default value when the Pause method is issued without options specified.

The following example pauses the SAS Metadata Server to an ADMIN state:

```
<! --- Pause the server to ADMIN   --->
options='<Server State="ADMIN"/>';

rc=serverObject.Pause(options);
```

The following example shows how the <PAUSECOMMENT> element is used:

```
<! --- Pause the server to OFFLINE with a comment  --->
options='<PauseComment>This is a test of the Pause method.  Client "+
"activity on the SAS Metadata Server will be resumed shortly.</PauseComment>';

rc=serverObject.Pause(options);
```

Refresh

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 server.

Category: Server control methods

Syntax

```
rc=Refresh(options);
```

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 it did, RC=1 means it did or not.
options	C	in	<ARM *system-option(s)*/> Invokes specified ARM logging system options. <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 file in the specified location. This option is ignored if journaling is not enabled on the SAS Metadata Server. <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 value. <SERVER/> Specifies to close and reopen all of the metadata repositories that are managed by the SAS Metadata Server to recover memory.

Details

A user must have administrative user status on the SAS Metadata Server to issue the Refresh method.

Issuing the Refresh method without options has no effect.

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 <SERVER/> element does not have any attributes. It is specified in the OPTIONS parameter as follows:

```
<SERVER/>
```

Issuing the Refresh method with the <ARM *system-option*/> or <OMA JOURNALPATH="*filename*"/> element reconfigures a feature on the running SAS Metadata Server.

- When issued with the <ARM *system-option*/> element, Refresh enables or disables ARM logging for all repositories on the metadata server. The option specifies ARM logging system options as follows:

  ```
  <ARM ARMSUBSYS="(ARM_NONE|ARM_OMA)" ARMLOC="fileref|filename"/>
  ```

  "(ARM_NONE)" disables ARM logging on the server. When specified together, "(ARM_OMA)" and ARMLOC="*filename*" start ARM logging in the specified location. If ARM logging is already enabled, specifying just ARMLOC="*filename*" writes the ARM log to the specified 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*.
- The <OMA JOURNALPATH= "*filename*"/> element redirects journaling entries to a new journal file in the specified location. The option cannot turn journaling on and off.

The <ARM *system-option*/> and <OMA JOURNALPATH=" *filename*"/> elements do not close and reopen repositories. They do not affect memory usage.

The <REPOSITORY/> element unloads the specified metadata repository (or metadata repositories) from memory, closes their containers on disk, and then reopens them. The <REPOSITORY/> element is specified as follows:

```
<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. In SAS releases before 9.2, ALL excluded the SAS Repository Manager. ALL is the default value if <REPOSITORY/> is specified without an Id= value and has the same effect as specifying the <SERVER/> element.

In previous SAS releases, the Refresh method supported a State= attribute in the <REPOSITORY/> element. The State= attribute is deprecated in SAS 9.2. Repositories are always paused to an OFFLINE state and resumed to their current Access= value.

The SAS Metadata Server must refresh repositories when there is no other activity on the server, so it automatically delays other client requests until the Refresh method completes. This might have a small effect on server performance.

Examples

The following example shows how ARM_OMA logging is enabled:

```
options='<ARM armbsubsys="(ARM_OMA)" armloc="myARM.log"/>';
rc=serverObject.Refresh(options);
```

The following example shows how ARM_OMA logging is disabled:

```
options='<ARM armbsubsys="(ARM_NONE)")/>';
rc=serverObject.Refresh(options);
```

The following example shows how to pause and resume a metadata repository to recover memory:

```
options='<Repository Id="A5H9YT45"/> ';
rc=serverObject.Refresh(options);
```

Resume

Returns a paused SAS Metadata Server to an ONLINE state.
Category: Server control methods

Syntax

```
rc=Resume(options);
```

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 it did, RC=1 means it did not.
options	C	in	<REPOSITORY/> Deprecated in SAS 9.2. <SERVER/> Specifies to return the SAS Metadata Server to an ONLINE state.

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 an ONLINE state. The ONLINE state makes the server available to process client requests.

Beginning in SAS 9.2, the Resume method cannot be used to change the availability of specific metadata repositories. If the Resume method is issued with the <REPOSITORY/> element, the SAS Metadata Server returns an error.

If the Resume method is issued without options, the method operates as if the <SERVER/> element was specified.

Example

The following example returns the SAS Metadata Server to an ONLINE state:

```
<! --- Resume the server to its normal access mode --->
options=' ';

rc=serverObject.Resume(options);
```

Status

Polls the SAS Metadata Server for status, platform version, SAS Metadata Model version, server locale, server configuration information, and journaling statistics.

Category: Server control methods

Syntax

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

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 it did, RC=1 means it did not.
inmeta	string	in	An XML element that requests additional information to be returned from the SAS Metadata Server. One or more elements can be specified. If no elements are specified, the Status method returns <MODELVERSION/>, <PLATFORMVERSION/>, <SERVERSTATE/>, <PAUSECOMMENT/>, and <SERVERLOCALE/>, by default. <MODELVERSION/> Requests the SAS Metadata Model version number. <OMA *AttributeName*=" "/> Requests the value that is active for the specified <OMA> attribute in the server's default configuration or in the omaconfig.xml file. <OMA *JournalStatistic*=" "/> Returns the specified journal statistic about the SAS Metadata Server journal file. See "Requesting Journaling Statistics" on page 197 for information about the available statistics. <PLATFORMVERSION/> Requests the SAS Metadata Server version number. <PAUSECOMMENT/> Used with <SERVERSTATE/>, this option returns a user-defined text comment describing why the server is unavailable if the server is in an ADMIN or OFFLINE state. <RPOSMGR *AttributeName*=" "/> Requests the value that is active for the specified <RPOSMGR> attribute in the server's default configuration or in the omaconfig.xml file. <SERVERLOCALE/> Requests the server locale that is active for the SAS Metadata Server session. <SERVERSTATE/> Requests information about the SAS Metadata Server's current state. <STATE/> Deprecated in SAS 9.2. Use <SERVERSTATE/> instead. <VERSION/> Deprecated in SAS 9.1.3, Service Pack 3. Use <MODELVERSION/> and <PLATFORMVERSION/> instead.
outmeta	string	out	Mirrors the INMETA parameter with the exception that return values are filled in.
options	C	in	No options are supported at this time.

Details

Any user who has access to the SAS Metadata Server can use the Status method.

The INMETA parameter is a string containing one or more XML elements that requests information. If a null string is passed, the Status method returns values for <MODELVERSION/>, <PLATFORMVERSION/>, <SERVERSTATE/>, <PAUSECOMMENT/>, and <SERVERLOCALE/>, by default. These options are considered the standard elements.

The other elements in the INMETA parameter are optional and can be used to return the following information:

- the values of specified server configuration options that are set in the omaconfig.xml file.
- journaling statistics for installations that have journaling enabled. In SAS 9.2, the default server configuration has journaling enabled.

For more information, see "Requesting omaconfig.xml Values" on page 197 and "Requesting Journaling Statistics" on page 197.

Standard Status Elements

The following is a more detailed description of the standard Status elements:

<MODELVERSION/>
Returns the SAS Metadata Model version number in the form X.XX. For example, 11.03. 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, as it was between SAS 9.1.3 and SAS 9.2, 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 for conversion of 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 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 online, 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 9.2, the platform version number is 9.2.0.0. For a SAS Metadata Server that is still running SAS 9.1.3, Service Pack 4, the platform version number is 9.1.3.4.

<SERVERSTATE/>
Returns the SAS Metadata Server's current state. Valid values are ONLINE, ADMIN, or OFFLINE. No response means that the server is down.

<STATE/>
Returns the same information as <SERVERSTATE/>.

<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.

<VERSION/>
Returns the same information as <MODELVERSION/>.

Note: The Status method reports the availability of the SAS Metadata Server to client requests. It cannot be used to check the state of specific metadata repositories. 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 plug-in, or issue a GetRepositories method that sets the OMI_ALL (1) flag. For more information, see "GetRepositories" on page 97. △

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 a definitive listing of all server configuration settings.

The omaconfig.xml file supports options in three categories.

- General server control, where each option is specified as an XML attribute of an <OMA> element.
- Repository manager control, where each option is specified as an XML attribute of an <RPOSMGR> element.
- Internal authentication control, where each option is specified as an XML attribute of an <InternalAuthenticationPolicy> element. This category is new in SAS 9.2.

The Status method obtains values for specified <OMA> and <RPOSMGR> elements, if attributes for these elements exist in the omaconfig.xml file. If the requested attribute is missing from the omaconfig.xml file, the Status method returns a blank value. A blank value indicates that the option is operating with a standard configuration setting.

To request an attribute value, submit an XML element of the appropriate category (<OMA> or <RPOSMGR>) that specifies the name of the attribute whose value you'd like to read in the INMETA parameter. To determine the attribute names available in each category and learn their standard configuration settings, see "Configuration Files: Reference for omaconfig.xml" in the *SAS Intelligence Platform: System Administration Guide*. Capitalize both the category name and the attribute name in the input XML element. The Status method is case sensitive.

To determine the settings of <InternalAuthenticationPolicy> options, use the GetInternalLoginSitePolicies method. For more information, see "GetInternalLoginSitePolicies" on page 138.

Requesting Server Invocation Options

The only server invocation option that is returned by the Status method is LOCALE, which specifies the server language encoding for the server session. To get the active server LOCALE value, specify <SERVERLOCALE/> in the Status method.

Requesting Journaling Statistics

Journaling is a SAS Metadata Server performance feature that makes updates available in memory before they are written to disk. For more information about SAS

Metadata Server journaling, see the *SAS Intelligence Platform: System Administration Guide*.

<OMA JOURNALDATAAVAILABLE=""/>
Returns the number of bytes of data that are in the journal file in memory and have yet to be applied to metadata repositories on disk.

<OMA JOURNALENTRYCOUNTER=""/>
Returns a sequence number that indicates the number of transactions that have been processed since the SAS Metadata Server was started with journaling enabled. The number includes transactions that are still in the journal file, and transactions that have been applied to metadata repositories on disk. This number is reset when the server is paused to an OFFLINE state and then resumed, and when the server is stopped and then restarted.

<OMA JOURNALHISTORICALDATA=""/>
Returns the number of transactions that have been processed to the disk data sets since the SAS Metadata Server was started.

<OMA JOURNALMAXDATAAVAILABLE=""/>
Returns the number of bytes of data that has been written to the journal file. This number grows through the life of the journal file. It is not reset when the server is stopped and then restarted.

<OMA JOURNALQUEUELENGTH=""/>
Returns the number of transactions in the journal file that are waiting to be applied to metadata repositories on disk.

<OMA JOURNALSPACEAVAILABLE=""/>
Returns either the number of bytes of space left in the journal file if the file is a fixed size, or 999999999 if the file is not a fixed size.

<OMA SERVERSTARTPATH=""/>
Returns the pathname of the directory where the SAS Metadata Server was started.

<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 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.

Although these journaling statistics are requested like omaconfig.xml options, they are not documented the same way because they do not configure the SAS Metadata Server— they return statistics only.

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 9.2  --->
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>11.03</ModelVersion>
<PlatformVersion>9.2.0.0</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:

```
<!--- Default values returned by the Status method in SAs 9.2 --->
<Status>
<Metadata/>
<Options/>
</Status>
```

The Status method call that follows requests omaconfig.xml values and <SERVERLOCALE/>:

```
<!-- Get omaconfig.xml values -->

<Status>
<Metadata>
<OMA MAXACTIVETHREADS=""/>
<OMA JOURNALPATH=""/>
<OMA ALERTEMAIL=""/>
<RPOSMGR
LIBREF=""
ENGINE=""
PATH=""
OPTIONS=""/>
<SERVERLOCALE/>
</Metadata>
<Options/>
</Status>
```

This Status method call requests the value of the omaconfig.xml <OMA JOURNALPATH="*filename*"/> server configuration option and journaling statistics:

```
<!-- Get journaling statistics-->

<Status>
<Metadata>
<OMA JOURNALPATH=""
     JOURNALSTATE=""
     JOURNALQUEUELENGTH=""
     JOURNALDATAAVAILABLE=""
     JOURNALSPACEAVAILABLE=""
     JOURNALENTRYCOUNTER=""/>
</Metadata>
<Options/>
</Status>
```

The value returned for the JOURNALPATH attribute, which can be an absolute or relative pathname or a blank string, indicates whether journaling is enabled on the SAS Metadata Server. The other attributes return statistics about journaling, if journaling is enabled.

Related Methods

- "GetInternalLoginSitePolicies" on page 138

Stop

Shuts down the SAS Metadata Server.
Category: Server control methods

Syntax

```
rc=Stop(options);
```

Parameter	Type	Direction	Description
options	C	in	No options are supported at this time.

Details

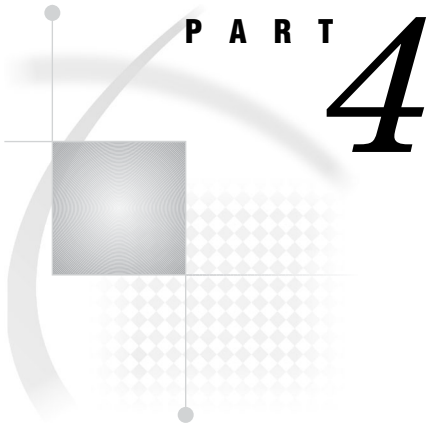
A return code of 0 indicates that the SAS Metadata Server was successfully stopped. A return code other than 0 indicates that the SAS Metadata Server failed to stop.

The Stop method is available only in the standard interface.

A user must have administrative user status on the SAS Metadata Server to stop the server.

Example

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

rc=serverObject.Stop(options);
```



PART 4

IOMI Server Interface Usage

CHAPTER

10

Adding Metadata Objects

Overview of Adding Metadata

The SAS Open Metadata Interface enables you to create metadata objects and their associations directly, or to create the metadata objects first, and then add associations to them later.

You create metadata objects with the AddMetadata method. With AddMetadata, you can do the following:

- create an object
- create an object and an association to an existing object
- create an object, an association, and the associated object

You add associations to an existing object by using the UpdateMetadata method. For information about using the UpdateMetadata method, see Chapter 11, "Updating Metadata Objects," on page 221.

Using the AddMetadata Method

The AddMetadata method enables you to create metadata objects that describe both metadata repositories and application elements. Adding a metadata object that

describes a metadata repository defines the repository in the SAS Metadata Server's SAS Repository Manager. A SAS Metadata Repository must be defined before you can define metadata objects describing application elements in the repository.

Note: The SAS 9.2 deployment process creates the initial SAS Metadata Repository for you, so direct use of the SAS Open Metadata Interface to define repositories is no longer necessary nor recommended. △

An application element is an item such as a table, a column, a key, or a SAS library. You create objects representing application elements to describe a data source or other entity used by your application.

Creating 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 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 a metadata repository, you specify the following information:

- a metadata property string that defines the repository's properties in the <METADATA> element
- the SAS Repository Manager identifier A0000001.A0000001 in the <REPOSID> element
- the REPOS namespace in the <NS> element
- the OMI_TRUSTED_CLIENT (268435456) flag in the <FLAGS> element

To create a metadata object describing an application component, you specify the following information:

- a metadata property string that defines the application element's properties in the <METADATA> element
- 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 metadata property string must be formatted in XML as described in "Constructing a Metadata Property String" on page 64. At a minimum, the metadata property string must specify the metadata type and any required properties of 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 application metadata types, see "Alphabetical Listing of SAS Namespace Metadata Types" in the *SAS 9.2 Metadata Model: Reference*.

Although it is not required, it is recommended that you provide values for all of an object's attributes, and for as many associations as possible in the metadata property string to make your metadata meaningful. Before creating objects, see "Selecting Metadata Types to Represent Application Elements" on page 210.

Creating Associations While Creating an Object

The metadata property string that defines a metadata object describing an application element can also define associations to metadata objects describing related application elements.

An association is defined by including XML subelements in the metadata property string that defines the application element:

- the first XML subelement specifies the association name
- the second element, nested within the first element, specifies a metadata property string that either references or defines the associated metadata 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="A53TPPVI"), in either the main element or in an association subelement.

Creating Cross-Repository References

The default behavior of the AddMetadata method is to create the primary 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 subelements defining the association. 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 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.

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 215.

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 association subelement.

When used to create multiple, unrelated metadata objects in the same repository, the form *$Alias* is used.

The 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 an application metadata object and an association to an existing 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 an object, an association, and a new associated 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 Get the Registered Repositories" on page 238.) 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, Unrelated Metadata Objects in an AddMetadata Request

To create multiple, unrelated 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>
```

Creating Multiple Associated Objects

To define multiple associations for an object, stack the associated object definitions within the primary object definition as follows:

```
<MetadataType Name="String" Desc="String">
   <AssociationName1>
     <AssociatedMetadataType Name="String" Desc="String"/>
   </AssociationName1>
   <AssociationName2>
     <AssociatedMetadataType Name="String" Desc="String"/>
  </AssociationName2>
</MetadataType>
```

Selecting Metadata Types to Represent Application Elements

The SAS 9.2 Metadata Model defines approximately 160 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 SASLibrary metadata type to represent data in a SAS library. 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 more closely describe the data source, and he might need to provide supporting information. For example, if the SAS library he is describing contains SAS tables that exist in a physical file system, he might select the PhysicalTable metadata type to describe the tables. If the tables are transient, he might use the WorkTable or QueryTable metadata types to describe the tables. 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.

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 application element, and to associate the objects with one another.

To assist metadata programmers in building consistent metadata definitions, the SAS 9.2 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 PrimaryType and SecondaryType in *SAS 9.2 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, see "PrimaryType and SecondaryType Abstract Types" in *SAS 9.2 Metadata Model: Reference.*

Example of an AddMetadata Request That Creates an Application Metadata Object

The following AddMetadata request creates an application 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 that 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 211, 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 an Application Metadata Object" on page 210. 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 an Application Metadata Object" on page 210. 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 an Application Metadata Object" on page 210. 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 an Application Metadata Object" on page 210. 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 an Application Metadata Object" on page 210. 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.

Additional Information

For more information about the metadata types used in these examples and other SAS Metadata Model metadata types, see the "Alphabetical Listing of SAS Namespace Metadata Types" in *SAS Metadata Model: Reference*. For reference information about the AddMetadata method, see "AddMetadata" on page 77.

CHAPTER

11

Updating Metadata Objects

Overview of Updating Metadata

The SAS Open Metadata Interface provides the UpdateMetadata method for updating existing metadata objects. 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 10, "Adding Metadata Objects," on page 205. 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 17, "Deleting Metadata Objects," on page 297. 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

The UpdateMetadata method enables you to 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 "Creating a Logical Metadata Definition" in the *SAS 9.2 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.

To modify an object's attributes, specify the metadata object, 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. The following is an example of 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.

To add, modify, or delete an association, or to modify an associated object's attributes, include an association name subelement and associated object definition in the metadata property string. In the association name subelement, include the Function= attribute. In the associated object definition, include the appropriate identifier and value. For example:

```
<Metadata>
  <MetadataType Id="reposid.objectid">
     <AssociationName Function="directive">
       <AssociatedMetadataType Id |ObjRef="value"/>
     </AssociationName>
  </MetadataType>
</Metadata>
```

In this metadata property string, note the following:

- The <METADATA> parameter identifies the metadata type, the object instance, the association name, and the associated object definition that you want to modify.

- The association name subelement specifies a Function= attribute and a directive that specifies how the SAS Metadata Server should process the associated object definition, depending on whether the association name supports a single or a multiple association.
- The associated object definition has a choice of object identifiers and values. The identifier attribute and value that you specify in the associated object definition determine whether a new associated object is created, whether an existing associated object's attributes are modified, or whether the request adds an association to an existing object.

Function= Attribute

The Function= attribute specifies how the SAS Metadata Server should process the associated object definition submitted for the association name, depending on whether the association name supports a single or a multiple association. This attribute is supported only in the association name element of a metadata property string that is passed to the UpdateMetadata method.

A single association refers to an association name that has a 0:1 or 1:1 cardinality defined for it in the metadata type documentation. Only one association of that name is supported between the specified metadata types.

A multiple association refers to an association name that has a 0 to many or 1 to many cardinality. This is denoted as 0:* and 1:* in the metadata type documentation. A multiple association supports many associations between the specified metadata types.

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 11.1 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.

Directive	Supported for Single Associations?	Supported for Multiple Associations?	Description
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.

How the Function Directives Affect Association Ordering

This example shows how the UpdateMetadata method applies the various Function= directives. Assume that a Column object exists that has an Extensions association to several objects. The Extensions association name enables an association to be created to any type of object. For the purpose of this example, the number of associations and their order in the association list is important, so they are identified numerically. For example:

```
extension1
extension2
extension3
extension4
```

If we were to execute an UpdateMetadata method call that modified extension1 and added extension5 with Function="APPEND," the directive causes the following changes to the association list:

```
extension1
extension2
extension3
extension4
extension5
extension6
```

APPEND never modifies existing associations. Any new associations are added to the end of the existing association list. Any associations that you attempt to modify are treated as new and listed as truly new associations. Extension1 was copied, assigned a new ID value, and added to the end of the list as extension6.

If we were to issue the same UpdateMetadata method call, except with Function="MERGE," the directive would change the association list as follows:

```
extension2
extension3
extension4
extension5
extension1
```

Like APPEND, MERGE adds new associations to the end of the existing association list. Unlike APPEND, however, it modifies and moves specified existing associations instead of just copying them. Extension1 is moved to the end of the association list.

Function="MODIFY" would have no effect in this UpdateMetadata method call. MODIFY is not supported for multiple associations.

If we were to issue the same UpdateMetadata method call, except with Function="REPLACE," the directive would cause the following changes to the association list:

```
extension5
extension1
```

The existing association list is deleted and replaced with a new list that lists new associations first, and any existing associations in the order that is specified. The old extension1 is replaced with the modified extension1.

If we were to issue the same UpdateMetadata method call with Function="REMOVE," the directive would cause the following changes to the association list:

```
extension2
extension3
extension4
```

The extension1 association is removed from the association list. The UpdateMetadata method returns an error message regarding extension5, because it was not in the list.

Summary of Function Directives

- To add associations to an existing association list without modifying the properties of the existing associated objects, specify Function="APPEND".
- To add new associations and modify the properties of the existing associated objects, specify Function="MERGE".
- To delete an existing association or association list with a new association or association list, specify Function="REPLACE".
- To remove an association without modifying any of the other associations in an existing association list, specify Function="REMOVE".
- To modify the properties of the associated object in a single association, specify Function="MODIFY".

Associated Object Identifier and Value

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 11.2 Identifiers and Values Supported in an Associated Object Definition

Identifier and Value	Result
Id="" Id="*$SymbolicName*" or no identifier	Create an association and the associated object. For more information about symbolic names, see “Symbolic Names” on page 208.
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.

Examples of How the Object Identifiers Work

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.

The following 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.

The following 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

To delete an association, specify the REPLACE or REMOVE directives in the Function= attribute of the association name element.

- The REPLACE directive replaces any existing associations with the specified associated object definition or list of associated object definitions. Use this directive with caution to prevent accidentally overwriting associations that you want to keep.
- The REMOVE directive removes the specified associated object definition from the list of associations maintained for that association name without affecting other associations.

Deleting an association does not delete the associated object, unless the associated object is a dependent object. A dependent object requires an association to another object to exist. Metadata types that have required associations are noted in the metadata type documentation as having a 1: 1 cardinality in their Associations table.

If you want to delete an object's associated objects and its associations, you must delete each associated object individually using the DeleteMetadata method. For more information, see Chapter 17, "Deleting Metadata Objects," on page 297.

The UpdateMetadata method does not include 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 232.

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 10, "Adding Metadata Objects," on page 205. 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 229:

```
<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>
```

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>
```

Additional Information

For more information about the UpdateMetadata method, see “UpdateMetadata” on page 110.

CHAPTER
12

Overview of Querying Metadata

Supported Queries

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 9.2, 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 about the SAS Metadata Server:

- 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

For more information, see "Status" on page 194.

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.

The GetNamespaces method enables you to get the namespaces programmatically. The GetTypes method gets all of the metadata types in a namespace. The GetTypeProperties method gets all possible properties for the specified metadata type. The SAS Metadata Model is a hierarchical model. The GetSubtypes method gets subtypes of a specified metadata type. The IsSubtypeOf method method determines whether one metadata type is a subtype of another metadata type. Together, 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 creating metadata objects. For more information, see "GetNamespaces" on page 96, "GetTypes" on page 105, "GetTypeProperties" on page 104, "GetSubtypes" on page 102, and "IsSubtypeOf" on page 108.

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. 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 Get the Registered Repositories" on page 238.

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 Repository Access and Status Information" on page 238.

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 a Repository's Regular Attributes" on page 240.

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 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 information about the read methods, see "GetMetadata" on page 89 and "GetMetadataObjects" on page 93.

For usage information, see Chapter 13, "Using GetMetadata to Get the Properties of a Specified Metadata Object," on page 243, Chapter 14, "Using GetMetadataObjects to Get All Metadata of a Specified Metadata Type," on page 263, and Chapter 15, "Filtering a GetMetadataObjects Request," on page 277.

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 GetSubtypes to Get a Metadata Type's Subtypes

To identify a metadata type's subtypes, use the GetSubtypes method. The following is an example of a GetSubtypes request that lists the subtypes of the AbstractExtension metadata type:

```
<GetSubtypes>
  <Supertype>AbstractExtension</Supertype>
   <Subtypes/>
   <NS>SAS</NS>
   <Flags>0</Flags>
   <Options/>
</GetSubtypes>
```

The <SUPERTYPE>, <NS>, <FlAGS>, and <OPTIONS> elements are input parameters. In the request, the <SUPERTYPE> element specifies the

AbstractExtension metadata type. The <NS> element specifies the SAS namespace. This example doesn't specify any flags or options. The <SUBTYPES> element is an output parameter.

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

```
<!-- Using the GETSUBTYPES method. -->

<SubTypes>
 <Type Id="Extension" Desc="Object Extensions" HasSubtypes="0"/>
 <Type Id="NumericExtension" Desc="Numeric Extension" HasSubtypes="0"/>
</SubTypes>
```

The AbstractExtension metadata type has two subtypes: Extension and NumericExtension. The HasSubtypes= attribute indicates these metadata types do not have any subtypes of their own.

Using GetRepositories to Get the Registered Repositories

You can get the repositories that are registered on a SAS Metadata Server by issuing the GetRepositories method. 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:

```
<!-- Using the GETREPOSITORIES method. -->

<GetRepositories>
   <Repositories>
     <Repository  Id="A0000001.A55WR3E8" Name="Foundation" Desc=" " DefaultNS="SAS"/>
     <Repository  Id="A0000001.A59XXOKF" Name="Custom" Desc=" " DefaultNS="SAS"/>
     <Repository  Id="A0000001.A5NJI601" Name="Custom2" Desc=" " DefaultNS="SAS"/>
     <Repository  Id="A0000001.A5J5NMEG" Name="Custom3" Desc=" " DefaultNS="SAS"/>
     <Repository  Id="A0000001.A5G61RLY8" Name="Project" Desc=" " DefaultNS="SAS"/>
  </Repositories>
</GetRepositories>
```

The current SAS Repository Manager has five repositories registered in it: Foundation, Custom, Custom2, Custom3, and Project.

Using GetRepositories to Get Repository Access and Status Information

To list the values of the repositories Path=, RepositoryType=, RepositoryFormat=, Access=, PauseState=, and CurrentAccess= attributes, issue the GetRepositories method with the OMI_ALL (1) flag set.

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:

```
<!-- Using the GETREPOSITORIES method. -->

<GetRepositories>
     <Repositories>
        <Repository  Id="A0000001.A0000001" Name="REPOSMGR" Desc="The Repository Manager"
DefaultNS="SAS" RepositoryType=" " RepositoryFormat=" 11"
Access="OMS_FULL" CurrentAccess="OMS_FULL" PauseState=" "  Path="rposmgr"  />
        <Repository  Id="A0000001.A55WR3E8" Name="Foundation" Desc=" "
DefaultNS="SAS"
RepositoryType="FOUNDATION " RepositoryFormat=" 11" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState=" "  Path="C:\SAS\FoundationServers\Lev1\
SASMain\MetadataServer\MetadataRepositories\Foundation"/>
        <Repository  Id="A0000001.A59XXOKF" Name="Custom" Desc=" " DefaultNS="SAS"
RepositoryType="CUSTOM" RepositoryFormat=" 11" Access="OMS_READONLY"
CurrentAccess="OMS_READONLY" PauseState="READONLY"  Path="C:\SAS\FoundationServers\
Lev1\SASMain\MetadataServer\MetadataRepositories\Custom"/>
        <Repository  Id="A0000001.A5NJI601" Name="Custom2" Desc=" " DefaultNS="SAS"
 RepositoryType="CUSTOM"  RepositoryFormat=" 11" Access="OMS_ADMIN"
CurrentAccess="OMS_ADMIN" PauseState="ADMIN"  Path="C:\SAS\FoundationServers\Lev1\
SASMain\MetadataServer\MetadataRepositories\Custom2"/>
        <Repository  Id="A0000001.A5J5NMEG" Name="Custom3" Desc=" " DefaultNS="SAS"
RepositoryType="CUSTOM"  RepositoryFormat=" 11" Access="OMS_OFFLINE"
CurrentAccess="OMS_OFFLINE" PauseState="OFFLINE"  Path="C:\SAS\FoundationServers\
Lev1\SASMain\MetadataServer\MetadataRepositories\Custom3"/>
        <Repository  Id="A0000001.A5G61RLY8" Name="Project" Desc=" " DefaultNS="SAS"
RepositoryType="PROJECT "  RepositoryFormat=" 11" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState=" "   Path="C:\SAS\FoundationServers\Lev1\
SASMain\MetadataServer\MetadataRepositories\Project"/>
    </Repositories>
  </GetRepositories>
```

In the output, note the following:

- There are five repositories registered in addition to the SAS Repository Manager. The SAS Repository Manager 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 in SAS 9.2, 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). There are three custom repositories and one project repository. The repository named Custom is registered with read-only access. The repository named Custom2 is registered with administrative access. The repository named Custom3 is registered as offline. The Project repository is registered with full access.
- All of the repositories have the same format (11).

- 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, the CurrentAccess= value might return a value of READONLY or OFFLINE.

Here is an example of the output from a GetRepositories method call that was issued on a SAS Metadata Server paused to an ADMIN state:

```
<!-- Using the GETREPOSITORIES method. -->

<GetRepositories>
     <Repositories>
        <Repository  Id="A0000001.A0000001" Name="REPOSMGR"
Desc="The Repository Manager" DefaultNS="SAS" RepositoryType=" "RepositoryFormat=" 11"
Access="OMS_FULL" CurrentAccess="OMS_FULL" PauseState="ADMIN" Path="rposmgr"/>
        <Repository  Id="A0000001.A55WR3E8" Name="Foundation" Desc=" " DefaultNS="SAS"
RepositoryType="FOUNDATION"  RepositoryFormat=" 11" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState="ADMIN" Path="C:\SAS\FoundationServers\
Lev1\SASMain\MetadataServer\MetadataRepositories\Foundation"/>
        <Repository  Id="A0000001.A59XXOKF" Name="Custom" Desc=" " DefaultNS="SAS"
RepositoryType="CUSTOM"  RepositoryFormat=" 11" Access="OMS_READONLY"
CurrentAccess="OMS_READONLY" PauseState="ADMIN(READONLY)" Path="C:\SAS\
FoundationServers\Lev1\SASMain\MetadataServer\MetadataRepositories\Custom" />
        <Repository  Id="A0000001.A5NJI601" Name="Custom2" Desc=" " DefaultNS="SAS"
RepositoryType="CUSTOM"  RepositoryFormat=" 11" Access="OMS_ADMIN"
CurrentAccess="OMS_ADMIN" PauseState="ADMIN" Path="C:\SAS\FoundationServers\
Lev1\SASMain\MetadataServer\MetadataRepositories\Custom2"/>
        <Repository  Id="A0000001.A5J5NMEG" Name="Custom3" Desc=" " DefaultNS="SAS"
RepositoryType="CUSTOM" RepositoryFormat=" 11" Access="OMS_OFFLINE"
CurrentAccess="OMS_OFFLINE" PauseState="OFFLINE" Path="C:\SAS\FoundationServers\Lev1\
SASMain\MetadataServer\MetadataRepositories\Custom3"/>
        <Repository  Id="A0000001.A5G61RLY8" Name="Project" Desc=" " DefaultNS="SAS"
RepositoryType="PROJECT "  RepositoryFormat=" 11" Access="OMS_FULL"
CurrentAccess="OMS_FULL" PauseState="ADMIN" Path="C:\SAS\FoundationServers\Lev1\
SASMain\MetadataServer\MetadataRepositories\Project"/>
   </Repositories>
 </GetRepositories>
```

In this output, note the following:

- The PauseState= attribute of all repositories except Custom3, which has a value of OFFLINE, have the word ADMIN added. A server pause is intended to change the repository's registered availability to a more restrictive state. A repository cannot be changed to a less restrictive state than OFFLINE.
- The Custom repository was changed to ADMIN(READONLY), meaning it is still in read-only mode. However, only administrators can read it.

Using GetMetadata to Get a Repository's Regular Attributes

To get general information about a repository, such as its Engine=, MetadataCreated=, and MetadataUpdated= values, 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:

```
<!-- Using the GETMETADATA method. -->

<GetMetadata>
   <Metadata>
         <RepositoryBase  Id="A0000001.A55WR3E8" Name="Foundation" Access="0"
Desc=" " Engine="BASE" MetadataCreated=''09Apr2008:18:06:57" MetadataUpdated=
"18Sep2008:18:47:54" Options=" " Path="C:\SAS\FoundationServers\Lev1\SASMain\
MetadataServer\MetadataRepositories\Foundation"
RepositoryFormat="11" RepositoryType="FOUNDATION">
         <DependencyUsedBy/>
           <DependencyUses/>
      </RepositoryBase>
   </Metadata>
  <NS>REPOS</NS>
  <Flags>1</Flags>
   <Options/>
</GetMetadata>
```

The GetMetadata method returns the following attribute values that are not returned by the GetRepositories method: Engine=, MetadataCreated=, MetadataUpdated=, and Options=. The value Access="0" is comparable to OMS_FULL and means ONLINE with full access.

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.

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. See “GetMetadataObjects” on page 93.

CHAPTER

13 Using GetMetadata to Get the Properties of a Specified Metadata Object

Introduction to the GetMetadata Method

To get properties for a metadata object, the SAS Open Metadata Interface provides the GetMetadata method. The default behavior of the GetMetadata method is to get the metadata object with whatever properties are specified in the INMETADATA parameter. The properties can include the XML attributes of the specified metadata object and association names. For example, consider the following GetMetadata request, which is formatted for the DoRequest interface:

```
<GetMetadata>
 <Metadata>
   <Column Id="A53TPPVI.A5000001" Name="" Desc="" ColumnType="" SASFormat="">
    <Table/>
   </Column>
 </Metadata>
 <NS>SAS</NS>
 <Flags>0</Flags>
 <Options/>
</GetMetadata>
```

In the <METADATA> element, the property string specifies to get the following:

- The 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.

- The Name=, Desc=, ColumnType=, and SASFormat= attributes of the Column object.
- Any objects that are associated to the Column object through the Table association name. A Column object can have one table object associated with it. For a list of the table metadata types supported under the Table association name, as well as other association names defined for the Column metadata type and other metadata types, see 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:

```
<!-- Using the GETMETADATA method. -->

<Column Id="A53TPPVI.A5000001" Name="City" Desc="City of Sales Office"
  ColumnType="12" SASFormat="$Char32.">
  <Table>
    <PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"/>
   </Table>
</Column>
```

The SAS Metadata Server returns values for the requested attributes of the specified Column object, and general, identifying information (Id= and Name=) about the associated PhysicalTable object.

To get additional properties for the specified Column object and its associated objects, the GetMetadata method supports the following flags:

- OMI_ALL (1)—Gets all of the attributes and associations of the specified object, and general, identifying information about any associated objects.
- OMI_ALL_SIMPLE (8)—Gets all of the attributes of the specified object.
- OMI_SUCCINCT (2048)—Omits properties that do not contain a value or that contain a null value.
- OMI_TEMPLATE (4)—Instructs the SAS Metadata Server to check the <OPTIONS> element for user-defined templates which define additional metadata properties to return. The templates can request additional properties for the specified metadata object, as well as attributes and associations for associated metadata objects. Templates are specified in a <TEMPLATES> element.
- OMI_INCLUDE_SUBTYPES (16)—When set with the OMI_TEMPLATE flag, gets properties for metadata objects that are subtypes of the specified metadata type, and gets the properties of the specified metadata object.

GetMetadata and Cross-Repository References in SAS 9.2

The SAS Metadata Server enables clients to create associations between objects in different repositories. An association that is defined between objects that exist in different repositories is referred to as a cross-repository reference. Between SAS 9.1.3 and SAS 9.2, the way that cross-repository references are retrieved has changed.

In SAS 9.1.3, the SAS Metadata Server relied on a dependency model that defined which repositories could have cross-repository references created between them. This dependency model required that a directionality be specified for queries requesting cross-repository references. Repositories that had dependencies defined existed in a repository chain. The default behavior of the GetMetadata method was to return cross-repository references from the specified repository and from repositories above it in the repository chain. To get cross-repository references under the specified repository in the repository chain, a client had to set the OMI_DEPENDENCY_USED_BY (16384)

flag. The output included cross-repository references to objects that were in custom and project repositories.

In SAS 9.2, the SAS Metadata Server has been changed to support an independent repository model. In this model, the following is true:

- Repository dependencies are ignored.
- Cross-repository references can be created between the foundation repository and any custom repositories. The foundation repository and custom repositories are considered public repositories because their content is intended for general use by clients.
- A GetMetadata method issued on an object in a public repository that requests information about associated objects automatically returns associated objects from all public repositories.
- The OMI_DEPENDENCY_USED_BY flag is set only if you want to include cross-repository references to objects in project repositories in the results. Project repositories are considered private unless specifically requested.

A GetMetadata request that is issued in a project repository returns associated objects that are in the project repository, and returns cross-repository references from all of the public repositories that the project repository services.

Expanding a GetMetadata Request to Get All of An Object's Attributes

To get all of a metadata object's attributes, 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. The following is an example of a GetMetadata request that sets the OMI_ALL_SIMPLE flag:

```
<GetMetadata>
 <Metadata>
   <Column Id="A53TPPVI.A5000001"/>
 </Metadata>
 <NS>SAS</NS>
 <!--OMI_ALL_SIMPLE flag -->
 <Flags>8</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 namespace in which to process the request.
- 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="A53TPPVI.A5000001" BeginPosition="0" ColumnLength="32"
  ColumnName="City" ColumnType="12" Desc="City of Sales Office"
  EndPosition="0" IsDiscrete="0" IsNullable="0" LockedBy=""
  MetadataCreated="05Feb2002:09:37:00" MetadataUpdated="05Feb2002:09:37:00"
  Name="City" SASAttribute="" SASColumnLength="32" SASColumnName="City"
  SASColumnType="C" SASExtendedColumnType="" SASExtendedLength="0"
```

```
  SASFormat="$Char32." SASInformat="$32." SASPrecision="0" SASScale="0"
  SortOrder="" SummaryRole=""/>
```

The GetMetadata method gets all attributes for the specified Column object, including attributes for which values have not been defined. The output does not include any associations. To limit the output to attributes that have values defined, 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> element. 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 of an Object's Properties

To get all of a metadata object's properties (attributes and associations), set the OMI_ALL (1) flag in the GetMetadata request. 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 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="A53TPPVI.A5000001" BeginPosition="0" ColumnLength="32"
  ColumnName="City" ColumnType="12" Desc="City of Sales Office" EndPosition="0"
  IsDiscrete="0" IsNullable="0" LockedBy="" MetadataCreated="05Feb2002:09:37:00"
  MetadataUpdated="05Feb2002:09:37:00" Name="City" SASAttribute=""
  SASColumnLength="32" SASColumnName="City" SASColumnType="C" SASExtendedColumnType=""
  SASExtendedLength="0" SASFormat="$Char32." SASInformat="$32." SASPrecision="0"
  SASScale="0" SortOrder="" SummaryRole="">
<AccessControls/>
<AnalyticColumns/>
<Changes/>
<DisplayForKey/>
<Documents/>
<Extensions/>
<ExternalIdentities/>
<ForeignKeyAssociations/>
<Groups/>
<Implementors/>
<Indexes/>
```

```
<Keys/>
<Keywords/>
<MLAggregations/>
<Notes/>
<PrimaryPropertyGroup/>
<Properties/>
<PropertySets/>
<QueryClauses/>
<ResponsibleParties/>
<SourceFeatureMaps/>
<SourceTransformations/>
<SpecSourceTransformations/>
<SpecTargetTransformations/>
<Table>
<PhysicalTable Id="A53TPPVI.A4000001" Name="Sales Offices"
  Desc="Sales offices in NW region"/>
</Table>
<TargetFeatureMaps/>
<TargetTransformations/>
<Timestamps/>
  <Trees/>
<UniqueKeyAssociations/>
<UsedByPrototypes/>
<UsingAggregations/>
<UsingPrototype/>
</Column>
```

The GetMetadata method gets all attributes and associations for the specified Column object, 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 one associated object defined. 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.

Expanding a GetMetadata Request to Get Properties of Associated Objects

A GetMetadata request gets requested attributes for the specified object and only the Id=, Name= and Desc= attributes of any associated objects that are requested directly in the <METADATA> element, or indirectly by the OMI_ALL (1) flag. To get additional attributes for associated objects, you must set the OMI_TEMPLATE (4) flag and specify a template in the GetMetadata request. A template is an additional property string that is specified in the OPTIONS parameter of the GetMetadata method within a <TEMPLATES> element.

The template can request additional attributes for the object specified in the <METADATA> element of the GetMetadata request, it can request specific attributes for associated objects requested in the <METADATA> element, and it can request additional associated objects. The attributes that are requested in the template are retrieved, in addition to the attributes that are requested in the <METADATA> element

or that are requested by other GetMetadata flags. For information on how to create a template, see “Using Templates” on page 260.

The following is an example of a GetMetadata request that sets the OMI_TEMPLATE flag and specifies a template to request additional attributes for both the specified object and associated objects requested in the <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 of the metadata type’s attributes, and the association name Table.
- The <NS> element specifies the namespace.
- The <FLAGS> element specifies the number representing the OMI_TEMPLATE flag.
- The <OPTIONS> element contains a <TEMPLATES> element and two templates. The first template specifies additional attributes to retrieve for the Column object identified in the <METADATA> element. The second template specifies additional 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

The GetMetadata method supports search criteria to filter the associated objects that it returns. The search criteria can be specified in both the <METADATA> element and in the <TEMPLATES> element. The search criteria enable you to retrieve only associated objects that are of a specified metadata type, or are of a specified metadata type and meet specified attribute criteria.

The search criteria is specified in a string in the association name element of the XML property string in one of the following forms:

```
<AssociationName search="Object"/>


<AssociationName search="Object[AttributeCriteria]"/>
```

- *Object* can be an * (asterisk) or a SAS Metadata Model metadata type that is a valid associated object 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 name instructs the SAS Metadata Server to get only associated objects of the specified metadata type.
- [*AttributeCriteria*] is an attribute specification that conforms to the syntax documented for the <XMLSELECT> option in "Attribute Criteria Component Syntax" on page 279. When attribute criteria are specified, GetMetadata retrieves only associated objects indicated by *Object* that also meet the specified attribute criteria.

This syntax has changed from SAS 9.1, which supported search criteria in the following form:

```
<AssociationName search="AttributeCriteria"/>
```

The SAS 9.2 syntax improves performance by enabling users to limit the number of metadata types on which the attribute criteria are evaluated. The older syntax form is still supported. It is the same as specifying the following:

```
"*[AttributeCriteria]"
```

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>
```

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. △

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 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.

Including Objects from Project Repositories in a Public Query

In SAS 9.2, a GetMetadata method call that requests associated objects and is issued in the foundation repository or a 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 associations for the requested object, and all attributes of the associated objects returned by OMI_ALL.

When OMI_ALL_SIMPLE (8) is set with OMI_TEMPLATE (4), 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) is set with OMI_TEMPLATE (4), 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 is set with OMI_TEMPLATE (4) and a template, 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 Templates

A template is a property string that you specify in a <TEMPLATES> element. In a GetMetadata or GetMetadataObjects method call, the <TEMPLATES> element is passed in the <OPTIONS> element to request additional properties for the metadata type specified in the main element of the method call, its subtypes, or its associated objects. The purpose of the template is to expand or to filter the properties requested by other GetMetadata and GetMetadataObjects parameters.

This section describes how to create a template and contains examples of GetMetadata and GetMetadataObjects requests that specify templates.

Creating a Template

A template is an XML property string that specifies the information that should be returned for a metadata type. The string includes the attributes and associations that should be returned for the metadata type. The string should not request properties for associated objects. Instead, additional templates should be created to request properties for associated objects.

For example, the following is a template for a PhysicalTable object that gets the object's ID, the date when the table was created, and the Column objects associated with the table:

```
<PhysicalTable Id="" MetadataCreated="">
   <Columns/>
</PhysicalTable>
```

To get properties for the requested Column objects, you should submit an additional template that looks like the following:

```
<Column Name="" SASFormat=""/>
```

The SAS Metadata Server gets the Name= and SASFormat= values for the Column objects requested by the first template.

Templates are passed to the SAS Metadata Server in a <TEMPLATES> element in the <OPTIONS> element of a method call. When the <TEMPLATES> element is used in a GetMetadata method call, the OMI_TEMPLATE (4) flag must also be set to instruct the SAS Metadata Server to check for the <TEMPLATES> element.

When the <TEMPLATES> element is used in a GetMetadataObjects method call, both the OMI_GET_METADATA (256) flag and the OMI_TEMPLATE (4) flag must be set. The OMI_GET_METADATA flag instructs the SAS Metadata Server to issue a GetMetadata call for each object that is returned by the GetMetadataObjects method to apply the templates specified in the <TEMPLATES> element.

The following is an example of a <TEMPLATES> element that passes the two templates previously described:

```
<Templates>
   <PhysicalTable Id="" MetadataCreated="">
     <Columns/>
   </PhysicalTable>
   <Column Name="" SASFormat=""/>
</Templates>
```

The metadata type specified in a template can be the same metadata type specified in the <METADATA> element of the GetMetadata request (or the <TYPE> element of a GetMetadataObjects request), a subtype, or the metadata type of an associated object. In the preceding examples, the object requested in the main element is assumed to be a PhysicalTable metadata object.

The order of the templates in the <TEMPLATES> element is not important unless the OMI_INCLUDE_SUBTYPES (16) flag is also set. The default behavior of the SAS Metadata Server is to search for objects of every metadata type listed in the <TEMPLATES> element, and to get the specified properties if objects are found. When OMI_INCLUDE_SUBTYPES is set, the SAS Metadata Server cycles through the templates iteratively, beginning with the first template and proceeding in order to the last template, and gets the specified properties of the specified metadata type and its subtypes. If a template for a subtype is found before a template for its supertype, then the subtype's template is applied, and there is no more searching for the supertype.

Specifying Search Criteria in a Template to Filter Associated Objects

Both the GetMetadata and GetMetadataObjects methods support specifying search criteria for the association names in the <METADATA> element and in the <TEMPLATE> element to filter the associated objects that are retrieved by a request. The search criteria enable you to get only associated objects that are the specified metadata type, or are the specified metadata type and also meet specified attribute criteria.

For information about using search criteria strings in the GetMetadata method, see "Filtering the Associated Objects That Are Returned By a GetMetadata Request" on page 249. For information about using search criteria strings in the GetMetadataObjects method, see "Filtering the Associated Objects That Are Retrieved By a GetMetadataObjects Request" on page 290.

CHAPTER

14 Using GetMetadataObjects to Get 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:

- "Expanding a GetMetadataObjects Request to Return Additional Properties" on page 264
- "Expanding a GetMetadataObjects Request to Include Subtypes" on page 272
- "Expanding a GetMetadataObjects Request to Include Additional Repositories" on page 273
- Chapter 15, "Filtering a GetMetadataObjects Request," on page 277
- "Filtering the Associated Objects That Are Retrieved By a GetMetadataObjects Request" on page 290

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 275.

Expanding a GetMetadataObjects Request to Return Additional Properties

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. Like the GetMetadataObjects method, when OMI_GET_METADATA is set without specifying any other GetMetadata flags, the GetMetadata method returns the Id= and Name= attributes for each metadata object that is returned by GetMetadataObjects. Specifying one or more other GetMetadata flags with OMI_GET_METADATA enables you to get specific properties or 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 265.
- 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 268.
- 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 268.
- 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 270 and "Example of Retrieving Associated Objects for All Objects" on page 270.

Specifying the 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> element.

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 specifed 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 “Using Templates” on page 260.

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 Retrieved By a GetMetadataObjects Request" on page 290.

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 264.

Expanding a GetMetadataObjects Request to Include Additional Repositories

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.

The behavior of these flags has changed between SAS 9.1.3 and SAS 9.2. In SAS 9.1.3, the flags specified to include objects from repositories that existed either above or below the specified repository in the repository chain. In SAS 9.2, their behavior is much simpler and is as follows:

- 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:

```
<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, and then second from all custom repositories in the order that 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 that 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 that 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 metadata objects of a specifed 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 that get additional properties and filter objects can be specified in the REPOS namespace as well.
- Output is returned in the <OBJECTS> element.

CHAPTER

15

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 Retrieved By a GetMetadataObjects Request" on page 290.

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> element for search criteria specified in an <XMLSELECT> element. The <XMLSELECT> search syntax enables you to filter objects based on the following:

- attribute criteria
- association path criteria
- a combination of the two criteria

- a concatenation of each criteria

Attribute criteria enable you to select only objects that contain specified attribute values. For example:

- You can specify to 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 to select objects that have a Name= attribute value of John Doe or Jane Doe (exclusive search).
- You can specify to 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 to select only Document objects that have a Reports association to a Report object.
- You can specify to select only 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 criteria. 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, the method filters the objects that are selected based on two or more associations that are directly defined for the specified metadata type. It joins the association paths using an implicit AND operator. As a result, the method selects only objects that meet all of the criteria specified in all of the association paths.

- An example of a concatenated association path is specifying to select Document objects that have a Reports association to a Report object that has a Name= attribute value of Sales, as well as a ResponsibleParties association to a ResponsibleParty object that has a Persons association to a Person object that has a Name= value of John Doe.

The method ignores any objects that do not have both a Reports association and a ResponsibleParties association and meet the criteria in both association paths.

The ability to concatenate association path criteria is new in SAS 9.2.

<XMLSELECT> Element Form and Search Criteria Syntax

The <XMLSELECT> element is specified within the <OPTIONS> element 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> element can be specified as one of the following:

```
Object[AttributeCriteria]
AttributeCriteria
(AttributeCriteria)
```

A statement that concatenates attribute criteria is specified as one of the following:

```
Object[AttributeCriteria  and|or  AttributeCriteria]

AttributeCriteria and|or  AttributeCriteria

(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 is specified as:

```
Object[AssociationPath1][AssociationPath2][AssociationPath3]
```

A description of each syntax component is provided in the following sections.

Object Component Syntax

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 name can be the same metadata type name 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 *SAS Metadata Model: Reference*.
- An * (asterisk) is a shorthand method of referring to the metadata type specified in the TYPE parameter.

Attribute Criteria Component Syntax

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 are described in the following table:

Symbol or Mnemonic	Description
=, 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.
gt, GT	Greater than the character string, numeric, or datetime value. MISSING value not supported.
ge, GE	Greater 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.
le, LE	Less than or equal to the character string, numeric, or datetime value. MISSING value not supported.
?, contains, or CONTAINS	Contains the specified character string. Numeric and datetime values are not supported. MISSING character value is supported.
=:	Begins with the specified character string. Numeric and datetime values are not supported. MISSING character value is supported.

Note: The NE, GE, and LE operators (and their lowercase aliases) are new in SAS 9.2. Also in SAS 9.2, GT and LT have been expanded to operate on character string values and numeric values. The expanded functionality allows comparisons to be performed on ranges of character string and numeric values. △

- '*value*' is a character or numeric string enclosed within single quotation marks.

  Character Strings:

  Searches of character strings compare the <XMLSELECT> 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 collation 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.

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 an <XMLSELECT> search string to GMT values for you. However, the datetime values returned by the server look different than 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 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. △

AssociationPath Component Syntax

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][AssociationPathn]
```

Each *AssociationPath* is the following:

```
[AssociationPathLevel/AssociationPathLeveln]
```

And, *AssociationPathLevel* is the following:

```
AssociationName/AssociatedObject[AttributeCriteria]
```

In the syntax, *Object* can be a metadata type name or an asterisk, as described in "Object Component Syntax" on page 279.

Each *AssociationPath* specifies one association of *Object* to evaluate. Specifying two *AssociationPath* components concatenates the criteria and indicates that two associations will be evaluated. Specifying a third *AssociationPath* indicates that three associations will be evaluated. The brackets enclosing each *AssociationPath* are required and function like an AND operator.

An *AssociationPath* specification can include one or more association path levels. The first *AssociationPathLevel* identifies the association of *Object* that will be evaluated. Subsequent levels are supported to enable you to filter 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 an Association Path" on page 282. Each *AssociationPathLevel* within the *AssociationPath* is separated from the other levels by a slash (/).

AttributeCriteria is optional in a *AssociationPathLevel*.

Understanding an Association Path

To understand how an association path is evaluated, we must consider each association path level that is specified. The *AssociationPathLevel* specifies an association 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 *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 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 subtypes. 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 *AssociationPathLevel* components in an *AssociationPath* get their context from the *AssociatedObject* in the preceding level.

- When the preceding associated object is a metadata type, then *AssociationName* must be an association name that is valid for that metadata type.
- When the preceding associated object is an *, then *AssociationName* can be any association name defined for one of the metadata types supported by the preceding association name.

Consider the following *AssociationPathLevel*:

```
Report[ReportLocation/Document[@TextType='XML']]
```

AssociationName in a second *AssociationPathLevel* specification must be an association that is valid for the Document metadata type. The *AssociatedObject* in the second *AssociationPathLevel* must be a metadata type that is supported by the second *AssociationName* or an *.

Consider the following *AssociationPathLevel*:

```
Report[ReportLocation/*]
```

AssociationName in a second *AssociationPathLevel* specification can be an association name that is valid for any of the 19 metadata types supported by the ReportLocation association. The *AssociatedObject* in the second *AssociationPathLevel* must be a metadata type that is supported by the second *AssociationName* or an *.

The following is an example of an *AssociationPath* that specifies multiple *AssociationPathLevel* components 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 *AssociationPathLevel* components:

1. ResponsibleParties/ResponsibleParty
2. Persons/Person
3. Locations/Location

The following is an example of an *AssociationPath* that specifies multiple *AssociationPathLevel* components 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 *AssociationPathLevel* components:

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, ExternalTable, PhysicalTable, QueryTable, Report, SharedDimension, TableCollection, and WorkTable objects that are owned by John Doe.

To determine what metadata types have subtypes, see the SAS Metadata Model documentation.

Understanding Concatenated Association Paths

An <XMLSELECT> search string that includes concatenated *AssociationPath* criteria must specify an association name that is valid for the *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 *AssociationPath* components are joined by an implied AND operator. Therefore, only objects that meet the criteria in the combined *AssociationPath* components will be selected. When the OMI_INCLUDE_SUBTYPES flag is set, only objects that have all of the specified attribute 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.

Sample Search Strings For Common Filters

Single Attribute Search on the Metadata Type in the <TYPE> Element

Both of the following <XMLSELECT> 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> Element

The following <XMLSELECT> 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> 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> element selects WorkTable objects that have a missing numeric value in the NumRows= attribute:

```
<XMLSELECT search="WorkTable[@NumRows='.']"/>
```

The following <XMLSELECT> element selects WorkTable objects that have a blank string in the MemberType= attribute:

```
<XMLSelect search="WorkTable[@MemberType='']"/>
```

Specifying Concatenated Attributes

The following <XMLSELECT> 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> 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> 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> 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> 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> 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.

Examples of Search Strings That Filter Objects Based on UsageVersion

The SAS 9.2 Metadata Model defines an optional 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 before SAS 9.2, or that the object is not versioned. Most SAS 9.2 Phase 2 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 new SAS 9.2 comparison operators described in "Attribute Criteria Component Syntax" on page 279 can be used to specify version criteria for the UsageVersion= attribute.

In the <XMLSELECT> 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 an <XMLSELECT> Element

The following example shows how the <XMLSELECT> element is specified in a GetMetadataObjects method call. The <XMLSELECT> element 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> element and a search string  -->
   <Options>
      <XMLSelect search="*[ResponsibleParties/*[@Role='Owner']/Persons/*
     [@Name='Joe Accountant']][Reports/Report[@Name='Sales']/Groups/Group
     [@Name='Accountant']]"/>
   </Options>
</GetMetadataObjects>
```

The request sets the OMI_XMLSELECT (128) flag and specifies the <XMLSELECT> search string in the <OPTIONS> element. Output is returned in the <OBJECTS> element.

Filtering the Associated Objects That Are Retrieved By a GetMetadataObjects Request

Search criteria that are specified in the <XMLSELECT> 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_GETMETADATA (256) and OMI_TEMPLATE (4) flags and specifying a search criteria string in the association name subelement of a template that requests associated objects.

The search criteria specified in an association name subelement supports a reduced form of the XMLSELECT syntax. The search criteria string supports object and attribute criteria in the following forms:

```
<AssociationName search="Object"/>
```

```
<AssociationName search="Object[AttributeCriteria]"/>
```

- *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.
- [*AttributeCriteria*] is an attribute specification that conforms to the syntax documented in "Attribute Criteria Component Syntax" on page 279. When attribute criteria are specified, GetMetadataObjects gets only associated objects specified by *Object*, which also meet the specified attribute criteria.

This syntax is a change from SAS 9.1, which supported search criteria in the following form:

```
<AssociationName search="AttributeCriteria"/>
```

The SAS 9.2 syntax improves performance by enabling users to limit the number of metadata types on which the attribute criteria are evaluated. The older syntax is still supported in SAS 9.2, and is the same as specifying "*[*AttributeCriteria*]".

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 were found that met the selection criteria.

Example of Using XMLSELECT and Template Filter Criteria in the Same Method Call

The following is an example of a GetMetadataObjects request that uses an <XMLSELECT> element to filter the initial set of objects that are retrieved, and a template to filter the associated objects:

```
<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> element and a <TEMPLATES> element. The <XMLSELECT> element specifies to get only Document objects that contain the word Customer in the Name= attribute. 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 was found that included the name Customer in the Name= attribute. One ResponsibleParty object was found that had the Role= attribute value of OWNER.

CHAPTER
16

Metadata Locking Options

Overview of Metadata Locking Options

The SAS Open Metadata Interface enables you to control concurrent access to metadata 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.

The change management functionality is implemented in SAS Data Integration Studio. For more information, see the SAS Data Integration Studio documentation.

To use these mechanisms, a user must have a registered identity on the SAS Metadata Server. For more information, see "User and Group Management" in the *SAS Intelligence Platform: Security Administration Guide*.

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. The flags represent the simplest concurrency control provided by the SAS Open Metadata Architecture. Before making an update, issue a GetMetadata method call with the OMI_LOCK flag on the objects that you wish to modify. The specified object and any associated objects specified by GetMetadata flags and options are locked. Each objects' LockedBy= attribute is updated with the metadata identifier representing the caller. The objects remain locked from update by users other than the calling identity until an UpdateMetadata method is issued with the OMI_UNLOCK or OMI_UNLOCK_FORCE flags.

OMI_UNLOCK unlocks a lock held by the caller and is the recommended method of unlocking a lock. OMI_UNLOCK_FORCE unlocks a lock held by another user and is intended to be used as an emergency override mechanism. While locked, objects can be read by other users. They cannot be updated by other users until the user holding the lock completes his update.

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” on page 89 and “UpdateMetadata” on page 110.

CHAPTER

17

Deleting Metadata Objects

Using the DeleteMetadata Method to Delete Application Metadata Objects

The DeleteMetadata method removes metadata from a SAS Metadata Repository. In SAS 9.2, a DeleteMetadata method that is issued in the SAS namespace removes the specified object. If the OMI_TEMPLATES (4) flag is set, DeleteMetadata also deletes associated objects that are specified in a user-defined template. The template is passed to the SAS Metadata Server in a <TEMPLATE> element within a <TEMPLATES> element in the OPTIONS parameter.

Using DeleteMetadata to Delete a Specified SAS Metadata Model Object

The default behavior of the DeleteMetadata method is to delete the metadata object specified in the INMETADATA parameter. Submit a metadata property string that identifies the object to be deleted in the INMETADATA parameter. Identify the object to delete by its metadata type and 17–character metadata identifier. It is not necessary to specify the object's Name= value. Set the OMI_TRUSTED_CLIENT flag in the FLAGS parameter. The OMI_TRUSTED_CLIENT flag must be set in all requests that add, delete, or update metadata on the SAS Metadata Server.

In addition to the specified object, the DeleteMetadata method automatically deletes any associated objects that have a 1:1 cardinality to the specified metadata object as defined in the SAS Metadata Model. A 1:1 cardinality indicates a dependent relationship between two objects.

The DeleteMetadata does not list the Id= values of deleted associated objects in its output. Set the OMI_RETURN_LIST (1024) flag to include these Id= values in the output.

To delete multiple objects at once, stack their property strings in the INMETADATA parameter. When deleting multiple objects, do not specify associated objects that have a 1:1 cardinality in the DeleteMetadata method. If you specify them, the SAS Metadata Server attempts to locate objects that have already been deleted, and it aborts the delete operation when they are not found. You can prevent the delete operation from

being aborted by setting the OMI_IGNORE_NOTFOUND (134217728) flag. However, it is recommended that you do not specify the associated objects instead.

The following is an example of a DeleteMetadata request that deletes an individual metadata object, in this case, a SASLibrary object. The OMI_RETURN_LIST (1024) flag is set with the OMI_TRUSTED_CLIENT (268435456) flag so that the OUTMETADATA parameter returns the identifiers of any dependent objects that might be deleted with the SASLibrary object. The request is formatted for the INMETADATA parameter of the DoRequest method.

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

For a listing of SAS Metadata Model metadata types that can be deleted with the DeleteMetadata method, see the SAS 9.2 Metadata Model documentation.

For information about how to also delete associated objects in a DeleteMetadata request, see "Deleting Associated Objects Using a User-Defined Template" on page 298.

Deleting Associated Objects Using a User-Defined Template

A DeleteMetadata request that specifies associated objects to delete has the following characteristics:

1. It identifies the primary or top-level metadata object to delete in the INMETADATA parameter.
2. In addition to the metadata type and Id= value of the target object, the INMETADATA property string specifies a TemplateName= attribute that specifies the name of a user-defined template. For example, TemplateName="MyTemplate".
3. It sets the OMI_TEMPLATE (4) and OMI_TRUSTED_CLIENT (268435456) flags in the FLAGS parameter.
4. It submits a <TEMPLATE> element within a <TEMPLATES> element in the OPTIONS parameter that specifies the metadata type from the INMETADATA parameter and identifies the associations that you want to delete.
5. The opening <TEMPLATE> tag includes a TemplateName= attribute and value that matches the TemplateName= value in the INMETADATA parameter.

The following is an example of a DeleteMetadata request that submits a user-defined template. The request is formatted for the INMETADATA parameter of the DoRequest method.

```
<DeleteMetadata>
   <Metadata>
      <MetadataType  Id="reposid.objectid" TemplateName="myassns"/>
  </Metadata>
 <NS>SAS</NS>
<!--OMI_TEMPLATE +  OMI_TRUSTED_CLIENT + OMI_RETURN_LIST  -->
<Flags>268436484</Flags>
<Options>
    <Templates>
```

```
        <Template TemplateName="myassns">
            <MetadataType>
                <AssociationName1/>
                <AssociationName2/>
                <AssociationName3/>
            </MetadataType>
        </Template>
       </Templates>
    </Options>
</DeleteMetadata>
```

Note the inclusion of the attribute TemplateName="myassns" in both the metadata property string in the <METADATA> element and the <TEMPLATE> element in the OPTIONS parameter. The content of the template specifies the same metadata type as the property string in the <METADATA> element and specifies the names of the associations that you want to delete.

Note how the <TEMPLATE> element is enclosed within a <TEMPLATES> element in the OPTIONS parameter. The use of a <TEMPLATE> element within a <TEMPLATES> element is unique to the DeleteMetadata method. It is supported to enable multiple objects and their associated objects to be deleted in a DeleteMetadata request. The property strings in the <METADATA> element are scoped to the appropriate template in the OPTIONS parameter using the value in the TemplateName= attribute.

The following is an example of a DeleteMetadata request that specifies to delete two objects and their associated objects:

```
<DeleteMetadata>
   <Metadata>
      <MetadataType1 Id="reposid.objectid" TemplateName="Template1"/>
      <MetadataType2 Id="reposid.objectid" TemplateName="Template2"/>
  </Metadata>
 <NS>SAS</NS>
<!--OMI_TEMPLATE +  OMI_TRUSTED_CLIENT + OMI_RETURN_LIST-->
<Flags>268436484</Flags>
<Options>
    <Templates>
       <Template TemplateName="Template1">
            <MetadataType1>
                <AssociationName1/>
                <AssociationName2/>
                <AssociationName3/>
            </MetadataType1>
       </Template>
       <Template TemplateName="Template2">
            <MetadataType2>
                <AssociationName1/>
                <AssociationName2/>
                <AssociationName3/>
            </MetadataType2>
       </Template>
      </Templates>
  </Options>
</DeleteMetadata>
```

The second example specifies two property strings in the <METADATA> element. Note how the TemplateName= value in each string maps to a <TEMPLATE> element with a matching TemplateName= value in the OPTIONS parameter.

For more information about how to create a template, see “Using Templates” on page 260.

Deleting a Repository

The DeleteMetadata method call for deleting a repository is formatted similarly to a DeleteMetadata method call for deleting an application metadata object, 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 using the DeleteMetadata method. Use SAS Data Integration Studio to manage all 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.

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*.

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **suggest@sas.com**.