SAS® 9.4 Language Interfaces to Metadata, Third Edition

SAS® Documentation
October 20, 2022

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 Language Interfaces to Metadata, Third Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 Language Interfaces to Metadata, Third Edition

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

January 2023

9.4-P8:lrmeta

Contents

PART 3 Metadata LIBNAME Engine

PART 4 Procedures

PART 5 DATA Step Functions

What's New in SAS 9.4 Language Interfaces to Metadata

Overview

The metadata DATA step function documentation has been updated for SAS 9.4M8.

Beginning with SAS 9.4M7, the length of a table object name created with the METALIB procedure has been extended. The documentation has been updated.

Beginning with SAS 9.4M6, the behavior of the METALIB procedure's DBAUTH statement has changed. The documentation has been updated.

Beginning with SAS 9.4M5, the METALIB procedure uses the SASTableName attribute to locate SAS data sources for update processing. The change enables the procedure to better handle a table object for which a prefix is defined. The documentation was enhanced.

Beginning with SAS 9.4M3, the METADATA procedure has a new argument that enables you to direct SAS Metadata Server status queries directly to the master node in a clustered server configuration. The METADATA procedure has a new cluster synchronization checking feature. The METALIB procedure has a new statement: DBAUTH. The METADATA LIBNAME engine has two new LIBNAME statement options: DBUSER= and DBPASSWORD=. The arguments supported with the engine's METAOUT= LIBNAME option and data set option have changed. There is a new metadata DATA step function. There are documentation enhancements.

Beginning in SAS 9.4M2, the METADATA procedure supports new XML elements for getting information about SAS Metadata Server alert conditions and grace periods. The METALIB procedure was enhanced to update library ownership. There are documentation enhancements.

In SAS 9.4, the SAS Metadata Server is available in a single SAS Metadata Server configuration or in a clustered SAS Metadata Server configuration. The METAOPERATE procedure, the METADATA procedure, the METACONNECT= system option, and the METAPROFILE system option are enhanced to support the new clustered configuration. The METAAUTORESOURCES system option and the METADATA LIBNAME engine have new functionality. There is a new metadata DATA step function. The documentation is enhanced. For more information, see "SAS 9.4" on page xii.

SAS 9.4M8

The following metadata DATA step function documentation has been updated for SAS 9.4M8:

- "Best Practices" for using the DATA step functions for metadata. The example has been modified.
- "METADATA_SETASSN Function" on page 247, to define the term "multiple association".
- "METASEC_GETNAUTH Function", to better describe masks.

SAS 9.4M7

The METALIB procedure now allows the creation of a table object name that is up to 60 characters. A table object's name includes the SAS table name and an optional prefix text string. The prefix text string is added with the PREFIX statement.

The METALIB procedure documentation has been modified. See "How PROC METALIB Works" on page 136 and "Considerations When Creating SASLibrary Objects" on page 138.

A new example has been added to the documentation: "Example: Creating Metadata for a JSON" on page 27.

SAS 9.4M6

The behavior of METALIB procedure's DBAUTH statement has changed. The database credentials specified in the DBAUTH statement now override any other predefined authentication types. For more information, see "DBAUTH Statement" on page 129.

The documentation for PROC METALIB has been enhanced to clarify how the procedure works with folders. See "SASLibrary Objects and Folders" on page 138.

SAS 9.4M5

PROC METALIB was changed to use the SASTableName attribute to locate a physical SAS data source for update processing. In previous releases, the procedure used the Name attribute to locate physical SAS data sources for update processing. The change affects several processes. For more information, see “How PROC METALIB Works” on page 136, “EXCLUDE or SELECT Statement” on page 130, and “PREFIX Statement” on page 134.

The PROC METALIB topic, “What Metadata Is Updated?” on page 137, was updated.

The requirements for creating a pre-assigned library definition that can be updated are documented in “*library-identifier*” on page 127.

SAS 9.4M3

Enhancements to PROC METADATA

Beginning with SAS 9.4M3, PROC METADATA supports an OPTIONS= argument. The following XML element is supported in the OPTIONS= argument of a PROC METADATA request:

<CLUSTER/>
: supported in requests that also specify METHOD=STATUS only: specifies to send the query in the IN= argument to the master node. This argument enables you to query the master node without knowing its connection parameters. For more information, see “Metadata Server Configurations and PROC METADATA” on page 96 and “Example 7: Get Information about the Server Cluster with PROC METADATA” on page 118.

Cluster synchronization checking is a new feature available in the SAS Management Console Metadata Manager Analyze/Repair wizard and the sas-analyze-metadata batch tool beginning with SAS 9.4M3. For more information about this feature, see the documentation for the Analyze/Repair wizard and the sas-analyze-metadata batch tool. The Analyze/Repair wizard and the sas-analyze-metadata batch tool are documented in the *SAS Intelligence Platform: System Administration Guide*

The following IServer Status XML element is supported in the IN= parameter of a PROC METADATA request that specifies METHOD=STATUS to get the results of cluster synchronization check:

<SynchCheck><Results *OptionalAttribute(s)=" "*/></SynchCheck>
reports the results of the last synchronization check on the slave node that received the Status query. By default, the query returns the Id and Name values of all repositories that were examined. In addition, the query returns a <Container/> XML element that specifies the name of any metadata type containers in which an error was found.

For more information about the <SynchCheck><Results/></SynchCheck> XML element, see the documentation for the IServer Status method in the *SAS Open Metadata Interface: Reference and Usage*. For more information about how this element is used in a PROC METADATA request, see "Example 7: Get Information about the Server Cluster with PROC METADATA" on page 118.

Enhancements to PROC METALIB

Beginning with SAS 9.4M3, the METALIB procedure has a new statement, DBAUTH. The DBAUTH statement specifies database authentication credentials for libraries that have an authentication type of Prompt in their server definitions. For more information, see "DBAUTH Statement" on page 129.

Enhancements to the METADATA LIBNAME Engine

Beginning with SAS 9.4M3, the metadata engine supports DBUSER= and DBPASSWORD= arguments. These options enable you to override database authentication credentials that are stored in metadata. Chapter 8, "Reference for the Metadata Engine," on page 69.

The METAOUT=META LIBNAME option and the METAOUT=META data set option are no longer supported. These options returned utility information about tables defined in metadata. That is, when METAOUT=META was set, PROC CONTENTS would return information about a table, but you could not read data in the table. See "METAOUT= Argument" on page 73 and "METAOUT= Data Set Option" on page 74.

New Metadata DATA Step Function

Beginning with SAS 9.4M3, a new metadata DATA step function, METADATA_GETURI, returns a URL for the application specified by the SoftwareComponent object using information from the SAS Metadata Repository. For more information, see "METADATA_GETURI Function" on page 235.

Documentation Enhancements

- We explain how tables that have user-defined formats should be used with the metadata engine. See “Features That Are Not Supported” on page 63.
- We clarify how column permissions are applied by the metadata engine in “The Metadata Engine and Authorization” on page 64.
- The description of the metadata LIBNAME statement’s METAOUT=DATAREG option has been clarified. METAOUT= can be specified as a LIBNAME option or as a data set option. See “METAOUT= Argument” on page 73 and “METAOUT= Data Set Option” on page 74.
- We list the attributes that PROC METALIB updates for a PhysicalTable metadata object in “What Metadata Is Updated?” on page 137.
- We have expanded the best practices for using metadata functions that get values. For more information, see “Best Practices” on page 192.
- We describe how to use the URLENCODE function to encode table names that have special characters before using the metadata DATA step functions. Metadata objects are cached by URIs. When the URI is created, the metadata DATA step logic decodes reserved URL characters, which can change the table name when the name has special characters. For more information, see “Best Practices” on page 192.

SAS 9.4M2

Enhancements to PROC METADATA

Beginning with SAS 9.4M2, the following XML elements from the IServer Status method are supported in the IN= parameter of a PROC METADATA call with METHOD=STATUS to get information about the alert condition and the grace period:

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

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

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

For more information, see "Example 6: Get Information about the Server's Alert Email Notification Subsystem with PROC METADATA" on page 115.

Enhancements to PROC METALIB

Beginning with SAS 9.4M2, the METALIB procedure is enhanced. The procedure checks for and updates a table object's library ownership if the table object is using a different library definition than the one with which it was created. This is useful when importing and exporting data. For more information, see "How PROC METALIB Works" on page 136.

Documentation Enhancements

- The documentation for the METAPASS= system option was updated to describe how passwords are displayed in the log. See "METAPASS= System Option" on page 49.
- The metadata engine documentation clarifies what support is available for utility functionality on a third-party DBMS. See "Features That Are Not Supported" on page 63.
- The requirements for the pathname parameter of the PROC METALIB FOLDER= statement were clarified. See "FOLDER= or FOLDERID= Statement" on page 131.

SAS 9.4

Enhancements to PROC METAOPERATE

Beginning in SAS 9.4, PROC METAOPERATE works in a single SAS Metadata Server configuration or in a clustered SAS Metadata Server configuration. In a clustered SAS Metadata Server configuration, three or more metadata servers are available for processing metadata requests.

When a clustered metadata server configuration is detected, actions that change the availability and content of the SAS Metadata Server are executed uniformly on all server nodes. Two new, optional arguments enable administrators to direct requests to a specific node in the cluster:

NOCLUSTER
supported in the STOP action, enables an administrator to stop the metadata server specified in the connection options. This enables an administrator to temporarily remove the specified server from the cluster if it needs maintenance.

NOREDIRECT
supported in the STATUS action, enables an administrator to direct the status query to the metadata server specified in the connection options.

Certain administrative tasks, such as metadata server restore and recovery, are different in a clustered SAS Metadata Server configuration than they are in a single SAS Metadata Server configuration. For more information, see “Metadata Server Configurations and PROC METAOPERATE” on page 168 and “Recovery in a Clustered Server Configuration” on page 173.

Enhancements to PROC METADATA

PROC METADATA has the following new features in support of the clustered metadata server configuration.

In a clustered metadata server configuration, all requests, including server status queries, are assigned by a load balancer to the server nodes in the cluster. This is appropriate for requests issued through METHOD=DOREQUEST, which primarily read and write metadata. A new argument is supported with METHOD=STATUS to enable an administrator to direct server status queries to a specific server node:

NOREDIRECT
specifies to execute the status query on the metadata server specified in the connection options.

The following IServer Status XML elements can be issued through PROC METADATA when METHOD=STATUS is specified to get information about a SAS Metadata Server cluster:

<CLUSTER *attributes*/>
returns values for specified cluster attributes. The valid attributes are:

CLUSTERGUID=" "
returns the cluster’s unique identifier.

DEFINED_NODES=" "
returns the number of servers defined in the cluster.

CURRENT_NODES=" "
returns the number of servers that are known to the server that received the query.

HAS_FIRST_NODE=" "
returns a YES or NO indicating whether the server defined as Node 1 is available to the cluster.

HAS_QUORUM=" "
returns a YES or NO indicating whether a quorum exists.

LIST=" "
returns an integer indicating the number of servers that are known to the server that received the query, in addition to ClusterNode XML elements that describe each server. The ClusterNode XML elements include the server name, host name, port number, a Self attribute, and a Flags attribute for each server. The Self attribute identifies the receiving server with a “Y” or a “N”. The Flags attribute indicates whether the node is a slave server or the master server.

<CLUSTERSTATE/>
returns the value STARTING, QUORUM, or LOSTQUORUM. STARTING means that the cluster is waiting for more server nodes to start up and complete the quorum. QUORUM means that a sufficient number of server nodes are operating for the cluster to remain in service. LOSTQUORUM means that the cluster does not have enough server nodes to remain in service.

<OMA MAXIMUM_CLUSTER_NODES=" "/>
returns the maximum number of server nodes that are supported in the cluster as configured in the omaconfig.xml file.

If any of these XML elements are submitted in a single SAS Metadata Server configuration, the elements return requested information about the single server, where appropriate. Otherwise, they are ignored.

For more information about PROC METADATA use in a clustered metadata server configuration, see “Metadata Server Configurations and PROC METADATA” on page 96 and “Getting Information about Server Backups ” on page 99.

Enhancements System Options

- The METAPROFILE and METACONNECT= system options support the use of a default server connection profile to connect to the SAS Metadata Server. The default server connection profile is created for every installation that has a SAS Metadata Server by SAS 9.4 configuration processes. In a clustered SAS Metadata Server configuration, the default server connection profile can assist with server connection and re-connection. The profile is helpful whether users connect with the profile or use the METASERVER= and METAPORT= options to connect. For more information, see “Specifying a Stored Connection Profile” on page 40, “METAPROFILE System Option” on page 51, and “METACONNECT= System Option” on page 45.
- Processing of the METAAUTORESOURCES system option has changed so that libraries assignments stored in metadata are always applied before library assignments in the AUTOEXEC file. For more information, see “METAAUTORESOURCES System Option” on page 43.

Enhancements to the Metadata LIBNAME Engine

The metadata LIBNAME engine supports extended attributes on SAS data sets and libraries when the METAOUT=DATA option is set. For more information, see “The Metadata Engine and Extended Attributes” on page 66.

New Metadata DATA Step Function

A new metadata DATA step function, METADATA_APPPROP, gets the value of a specified property for a specified SoftwareComponent or DeployedComponent. For more information, see “METADATA_APPPROP Function” on page 218.

Documentation Enhancements

- Information has been added about the permissions that the metadata LIBNAME engine enforces to control data access. For more information, see “Permissions That Affect Data Access through the Metadata Engine” on page 65.
- The documentation explains how PROC METADATA and PROC METAOPERATE can be used to check and change the addressees and email server configuration in the SAS Metadata Server’s alert email notification subsystem. For more information, see “Using Alert Email XML Elements” on page 175 and “Example 6: Get Information about the Server’s Alert Email Notification Subsystem with PROC METADATA” on page 115.

PART 1

Introduction

1

What Are the Metadata Language Elements?

Overview of Metadata Language Elements

SAS Open Metadata Architecture enables an administrator to define metadata objects that are common to one or more SAS client applications. For example, you can describe data sources and set security that supplements protections from the host environment and other systems.

In most cases, an administrator maintains the metadata by using products like SAS Management Console, SAS Data Integration Studio, or SAS Enterprise Guide. However, an administrator can also maintain metadata by running a SAS program in batch or from the SAS windowing environment. The code that can be submitted in a SAS session uses the SAS metadata language elements.

Many of the metadata language elements enable you to maintain metadata that defines a data source. A convention in the SAS Open Metadata Architecture is to refer to data in terms of SAS libraries, tables, rows, and columns.

- A data source is defined in metadata as a table.
- SAS tables are organized by being stored in a library.
- In SAS documentation, a row in a table is often called an observation, and a column is called a variable.

A SAS Metadata Server manages access to metadata in SAS metadata repositories. Some of the metadata language elements can be used to monitor and maintain the SAS Metadata Server.

This book is a reference to the metadata language elements. For information about metadata and SAS Metadata Server administration tasks, see the *SAS Intelligence Platform: System Administration Guide*.

The SAS metadata language elements described in this book include:

System options
: Use the system options to set defaults for metadata access. They are organized into three groups: connection to the SAS Metadata Server, client encryption, and resources.

Metadata LIBNAME statement
: As with other SAS engines, an administrator can assign a libref to serve as a shorthand for users. With the metadata engine, the underlying LIBNAME information is stored in metadata objects. The metadata engine helps implement security across an enterprise.

Data set options for the metadata engine
: You can apply these data set options to one table, rather than to an entire library.

Procedures
: You can use the following procedures to perform many common maintenance tasks on metadata and the SAS Metadata Server.

  - PROC METALIB automates the creation and update of table metadata for a specified SAS library. (The SAS library must be defined in a SAS Metadata Repository using SAS Management Console or SAS Data Integration Studio, first.)
  - PROC METADATA enables clients to submit XML-formatted SAS Open Metadata Interface method calls that read and write metadata objects of all SAS Metadata Model metadata types from within SAS. It also enables you to issue status requests that query the SAS Metadata Server's configuration, the server's backup configuration and history, and the server's availability. PROC METADATA returns XML output that mirrors the input, except the requested values are filled in. To process the output with SAS, you can define an XML map that can be read with the XML LIBNAME engine.
  - PROC METAOPERATE pauses, resumes, refreshes, backs up, recovers, and stops the SAS Metadata Server.

DATA step functions
: The DATA step functions cover the same metadata functionality as PROC METADATA, and return data to the DATA step, which can then be arranged in a SAS data set. Because the DATA step functions execute within a DATA step, you can use the output from one function as the input to another function.

The SAS commands METABROWSE, METACON, and METAFIND are documented in the online Help that is available from the SAS windowing environment.

When to Use Metadata Language Elements

Submitting a batch program can be helpful for repetitive metadata maintenance tasks. You might want to run reports automatically overnight, when usage of the SAS Metadata Server is low. The language elements are flexible and can be adapted to almost any metadata maintenance task.

SAS language elements that return information about the SAS Metadata Server's availability and configuration can be issued from the windowing environment at any time by users with administrative access to the server.

What Can I Report on in a SAS Metadata Repository?

The SAS Metadata Repository stores logical data representations of items such as the libraries, tables, information maps, and cubes that are used by SAS applications, as well as the information assets that are created by SAS applications. It stores information about system resources such as servers and the users who access data and metadata, and the rules that govern who can access what. You can create reports that track changes to all of these resources.

2

Using Language Elements That Read and Write Metadata

Overview of Using SAS Language Elements That Read and Write Metadata

PROC METADATA, PROC METALIB, and the metadata DATA step functions can be used to create metadata in the SAS Metadata Repository. PROC METADATA and the metadata DATA step functions enable you to read metadata from the SAS Metadata Repository.

PROC METALIB automates the process of creating table objects for a SAS library that you define in the SAS Metadata Repository. Use SAS Management Console to create the library definitions. The New Library wizards provides templates that collect the specific information needed to connect to a particular data source.

In order to create or read any metadata object with PROC METADATA and metadata DATA step functions, you must know the SAS Metadata Model metadata types that represent the objects in the SAS Metadata Repository. That information is not provided in this book. For more information, see the SAS Metadata Model: Reference. Most resources and information assets in the SAS Metadata Repository are described by a logical metadata definition. This logical metadata definition

includes multiple associated SAS Metadata Model metadata types, not just one. For applications to effectively share metadata, and for SAS tools to effectively import and export definitions, they must use common logical metadata definitions.

The SAS Intelligence Platform includes a type dictionary that SAS applications and solutions use to standardize the usage of common and shared resources and information assets in their applications and solutions. For resources that are persisted in metadata, this type dictionary standardizes their logical metadata definitions. The content of these logical metadata definitions is internalized so that SAS can change the definitions as needed.

This chapter describes how the type dictionary affects read and write requests made with PROC METADATA and the metadata DATA step functions.

Objects Included in the Dictionary

The type dictionary includes objects that describe common and shared resources and information assets. The dictionary also includes objects that need to be displayed in the SAS Management Console **Folders** tab and objects that need to be imported and exported. Examples of object types that are included in the type dictionary are: Table, Library, Information Map, SAS Report, Stored Process, Stored Process Server, Stored Process Report, Workspace Server, Job, Cube, User, and User Group.

Not all objects in the type dictionary can be imported and exported, and they do not all display in the SAS Management Console **Folders** tab.

What Is the SAS Type Dictionary?

The SAS type dictionary consists of a set of object type definitions. The dictionary is located in the **Types** subfolder of the **System** folder in the SAS Management Console **Folders** tab. Open the folder to see a complete list of the object types that are managed by the type dictionary.

In addition to the object's *logical metadata definition*, a *type definition* contains the information that is necessary to display and manage instances of an object type in a SAS application.

A goal of the type dictionary is to hide the details of logical metadata definitions and managing object instances from applications and users. The type definition publishes the name of the primary metadata type used to represent the object in the SAS Metadata Repository. However, it internalizes the information needed to expand the logical metadata definition. In the SAS Metadata Repository, the name of the type definition used to create any logical metadata definition is stored in the primary metadata object's PublicType= attribute.

How the Type Dictionary Affects SAS Language Elements

Creating Metadata

You should use SAS wizards and procedures, such as PROC METALIB and PROC OLAP. to automate the creation of metadata when possible. By using SAS wizards and procedures, you can be assured that the logical metadata definitions conform to the type dictionary. A program that uses PROC METADATA and metadata DATA step functions must create its own logical metadata definitions, and, as a result, these logical metadata definitions might not conform to the type dictionary.

Advantages of using logical metadata definitions that conform to the type dictionary include:

- The metadata objects are displayed on the SAS Management Console **Folders** tab. Not only are the objects visible in the GUI, but the folder container gives context to the object in the SAS Metadata Repository.
- The metadata objects can be managed using the functionality available on the **Folders** tab. You can copy, paste, delete, import a SAS package, and export a SAS package. Importing and exporting are not available for all object types.
- The metadata objects can be searched using the functionality on the SAS Management Console **Search** tab.

Reading Metadata

To retrieve object instances that conform to the type dictionary with SAS language elements for metadata, you identify the object instance by its primary metadata type and metadata identifier or name. To list objects that conform to the dictionary, you specify the object's primary metadata type and type name.

The type dictionary makes it easy to identify the primary metadata type and type name of an object type in the dictionary. In the dictionary, open the type definition of the object that you are interested in.

- The object's primary metadata type is specified in the **Metadata Type** field on the **Advanced** tab of its properties.
- The object type name is stored in the **TypeName** field on the **Advanced** tab.

For information about metadata identifiers or names, see "What Is a Metadata Identifier?" on page 13. Also, see "Obtaining Metadata Names and Identifiers" on page 13.

PROC METADATA enables you to get information about an object type instance by submitting the SAS Open Metadata Interface GetMetadata method. In the GetMetadata method's <Metadata/> parameter, specify the object's primary metadata type and name or identifier. In the GetMetadata method's <Flags/> parameter, set the OMI_FULL_OBJECT (2) flag. The OMI_FULL_OBJECT flag specifies to return an object's full logical metadata definition using the definition from the type dictionary. For an example of a PROC METADATA request that sets the OMI_FULL_OBJECT flag, see "Example 3: Request the Metadata for One Object" on page 105. Metadata DATA step functions do not support the ability to return an object's full logical metadata definition.

Many type definitions use the same metadata type as their primary metadata type. For example, Information Map and SAS Report both use the Transformation metadata type as their primary metadata type, among others. PROC METADATA enables you to list instances of an object type by submitting the SAS Open Metadata Interface GetMetadataObjects method. In the GetMetadataObject method's <Type/> parameter, specify the primary metadata type of the object type that you want to list. In the <Flags/> parameter, specify the OMI_XMLSELECT (128) flag, and in the <Options/> parameter, specify the <XMLSelect/> element and a search string. In the search string, specify to search for object instances that have the TypeName value in their @PublicType= attribute. For an example of a PROC METADATA request that uses the type dictionary to list objects, see "Example 4: Request the Metadata for One Type of Object" on page 109.

Metadata DATA step functions use a uniform resource identifier (URI) to identify an object. To list objects using the type dictionary, use this URI form:

```
omsobj: type?@PublicType='value'
```

The *type* is the **MetadataType** value, and *value* is the **TypeName** value from the type definition. For more information about URIs, see "What Is a URI?" on page 14. Also, see "METADATA_GETNOBJ Function" on page 229.

SAS provides the METADATA_PATHOBJ function for getting metadata objects in folders. Specify the **TypeName=** value in the **`DefType`** argument. For more information, see "METADATA_PATHOBJ Function" on page 242.

Deleting Metadata

SAS metadata interfaces automatically use the type dictionary to delete metadata when the specified metadata object is a PrimaryType subtype in the SAS Metadata Model and stores a value in the PublicType= attribute, unless you specify a user-defined template in the request. When you specify to delete the primary metadata object, the DeleteMetadata method also deletes all associated objects that are identified internally in the object's type definition.

Note: If the specified metadata object is a SecondaryType subtype in the SAS Metadata Model or is a PrimaryType subtype but does not store a value in the PublicType= attribute, then only the specified metadata object is deleted, unless you specify a user-defined template. For more information about PrimaryType and SecondaryType subtypes, see the *SAS Metadata Model: Reference*.

Repairing Metadata Objects

The SAS Management Console Analyze and Repair wizard uses type definitions from the dictionary to repair metadata objects. For more information about the Analyze and Repair wizard, see “Analyzing and Repairing Metadata” in *SAS Intelligence Platform: System Administration Guide*.

3

Metadata Object Identifiers and URIs

What Is a Metadata Identifier?

Every SAS Metadata Model metadata object in a SAS Metadata Repository has a unique identifier. The 17-character identifier consists of two parts, separated by a period. It is often represented in documentation as *reposid*.*objectid*. An example is `A52V87R9.A9000001`.

- The first eight characters (`A52V87R9` in the example) identify the SAS Metadata Repository in which the object is stored.
- The ninth character is always a period.
- The second set of eight characters (`A9000001` in the example) identifies the object in the repository.

Obtaining Metadata Names and Identifiers

Most of the metadata language elements require you to identify an object by its name or identifier. If you need the name or identifier of a single object, and you

know where the object is located in SAS Management Console or in SAS Data Integration Studio, then this task is simple. The metadata identifier is shown in the object's Properties window. For more information, see the Online Help that is available from the product.

Another way to locate an object is to issue the METABROWSE command to open the Metadata Browser window, or issue the METAFIND command to open the Metadata Find window. For more information, select **Using This Window** from the **Help** menu in the SAS windowing environment.

To retrieve a series of metadata identifiers programmatically, you can use the "METADATA_RESOLVE Function" on page 245 if you are processing within a DATA step.

Another choice is to submit a GetMetadataObjects method call with PROC METADATA, and then use the XML LIBNAME engine to import the procedure's XML output as a SAS data set. For a PROC METADATA example that retrieves object IDs, see "Example: Creating a Report with the METADATA Procedure and the XML Engine" on page 16.

What Is a URI?

For many of the metadata language elements, you can specify a metadata resource by its name or identifier. Some of the language elements accept a Uniform Resource Identifier (URI), which is a standard from SAS Open Metadata Architecture. The following URI formats are supported:

ID
is the metadata object identifier. Some language elements support the 8-character identifier, and some support the full 17-character identifier, which references both the repository and the object. Examples are `A9000001` and `A52V87R9.A9000001`. In general, the ID format is the least efficient.

type/*ID*
is the metadata type name and metadata object identifier. Some language elements support the 8-character object identifier, and some support the full 17-character repository and object identifier. Examples are `SASLibrary/A9000001` and `SASLibrary/A52V87R9.A9000001`. In general, the type/ID format is the most efficient.

type?@*attribute*='*value*'
is the metadata type name, followed by a search string. For metadata language elements, the search string is in the form of an *attribute*='*value*' pair. Examples are `SASLibrary?@libref='mylib'` and `Transformation?@PublicType='Report'`. The first example returns SASLibrary objects that store the value "mylib" in the Libref= attribute. The second example returns Transformation objects that store the value "Report" in the PublicType= attribute, which corresponds to the SAS Report type definition in the SAS type dictionary. For more information, see "What Is the SAS Type Dictionary?" on page 8. Some language elements require the entire value to be enclosed in quotation marks.

See the language elements in this book for important usage details.

4

Examples: Using Metadata Language Elements to Create Reports

Overview of the Examples

This section contains three examples. The first two examples, “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 16 and “Example: Creating a Report with the DATA Step” on page 21, show reports that can be created with metadata language elements. For information about the concepts involved in using the metadata language elements and examples of other ways the SAS language elements can be used, see the appropriate SAS language section.

The third example, “Example: Creating Metadata for a JSON” on page 27, shows how to download, read, and create metadata to control access to data that is read with the JSON engine. The JSON engine is a read-only, write-once engine. This topic describes the JSON engine output and concepts to be aware of when creating metadata for JSON.

Example: Creating a Report with the METADATA Procedure and the XML Engine

This example creates a report about all the tables in a user's library, including the tables' column names, librefs, and engines.

PROC METADATA requests the column names, and so on, from metadata, and writes the values in an XML file. Then, the XML LIBNAME engine uses an XMLMap to read the XML file and create SAS data sets. When the information is in SAS data sets, an administrator can run SAS code like DATA steps and procedures. This example uses PROC PRINT to create an HTML report.

To be clear, the files that are used in this example are described in the following list. The XML files are temporary and exist during the session only. However, you can also create permanent files.

Note: The XMLV2 engine accepts only permanent XML files as input.

- the user's library, which contains an unknown number of tables
- an input XML file, which is created by a DATA step to query the metadata
- an output XML file, which is created by PROC METADATA and contains information about the user's tables
- an XMLMap, created by a DATA step
- two SAS data sets, created by the XML LIBNAME engine and an XMLMap
- a third SAS data set, created by a DATA step MERGE
- an HTML report, created by ODS (Output Delivery System) statements

The METADATA procedure is documented in this book; see METADATA Procedure on page 87. The XML LIBNAME engine and XMLMaps are not documented in this book; see *SAS XMLV2 and XML LIBNAME Engines: User's Guide*.

The example begins by connecting to the SAS Metadata Server, updating the metadata about the library, and creating the input XML file.

```
/* submit connection information to server */

options metaport=8561
        metaserver="a123.us.company.com"
        metauser="myuserid"
        metapass="mypasswd";

/* Run PROC METALIB to be sure the metadata is
current.                  */
/* The library must be registered already in the SAS Metadata
Server.    */
```

```
/* Use the library name that is defined in the metadata, not the
libref. */

proc metalib;
  omr (library="mylib");
  report;
run;

/* Assign filerefs and libref. */
filename query temp;
filename rawdata temp;
filename map temp;
libname myxml xml xmlfileref=rawdata xmlmap=map;

/* Create temporary query file.                          */
/* 2309 flag plus template gets table name, column name,  */
/* engine, libref, and object IDs. The template specifies */
/* attributes of the nested objects.                      */

data _null_;
   file query;
   input;
   put _infile_;
   datalines;
<GetMetadataObjects>
   <Reposid>$METAREPOSITORY</Reposid>
   <Type>PhysicalTable</Type>
   <Objects/>
   <Ns>SAS</Ns>
   <!-- OMI_ALL (1) + OMI_TEMPLATE(4) +
OMI_GET_METADATA(256) + OMI_SUCCINCT(2048) flags -->
   <Flags>2309</Flags>
   <Options>
    <Templates>
      <PhysicalTable/>
      <Column SASColumnName=""/>
      <SASLibrary Engine="" Libref=""/>
    </Templates>
   </Options>
</GetMetadataObjects>
;;
run;
proc metadata
     in=query
     out=rawdata;
run;
```

The next section of example code creates a temporary text file that contains the XMLMap. The map enables the XML LIBNAME engine to process the XML returned by the metadata query as two data sets, ColumnDetails and LibrefDetails.

In the ColumnDetails data set, the observation boundary (TABLE-PATH) is at Column. Putting the boundary at Column is necessary because the PhysicalTable elements have multiple Column elements. If you need to read multiple elements, you must set the observation boundary at that element, so the XML LIBNAME engine can create multiple observations for the element.

Because the observation boundary is set at Column, each observation stops at Column, and any elements that follow Column are not properly read. Therefore, another data set is required. The LibrefDetails data set contains the SASLibrary elements. Later in the code, the ColumnDetails and LibrefDetails data sets are merged into a final data set.

The XMLMap is created in the following code to illustrate the process. This code creates a version 1.2 XMLMap. You can use a graphical user interface, such as SAS XML Mapper, to generate a current map. For more information, see *SAS XMLV2 and XML LIBNAME Engines: User's Guide*.

```
data _null_;
   file map;
   input;
   put _infile_;
   datalines;

<?xml version="1.0" ?>
  <SXLEMAP version="1.2">

   <TABLE name="ColumnDetails">
   <TABLE-PATH syntax="xpath">
     /GetMetadataObjects/Objects/PhysicalTable/Columns/Column
   </TABLE-PATH>

   <COLUMN name="SASTableName" retain="yes">
     <PATH>
        /GetMetadataObjects/Objects/PhysicalTable/@SASTableName
     </PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>14</LENGTH>
   </COLUMN>

   <COLUMN name="Columns">
     <PATH>
        /GetMetadataObjects/Objects/PhysicalTable/Columns/Column/
@SASColumnName
     </PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>12</LENGTH>
   </COLUMN>

   <COLUMN name="Column IDs">
     <PATH>
        /GetMetadataObjects/Objects/PhysicalTable/Columns/Column/@Id
     </PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>17</LENGTH>
   </COLUMN>

   </TABLE>


   <TABLE name="LibrefDetails">
```

```
   <TABLE-PATH syntax="xpath">
     /GetMetadataObjects/Objects/PhysicalTable/TablePackage/SASLibrary
   </TABLE-PATH>

   <COLUMN name="SASTableName">
     <PATH>
        /GetMetadataObjects/Objects/PhysicalTable/@SASTableName
     </PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>14</LENGTH>
   </COLUMN>

   <COLUMN name="Libref">
     <PATH>

/GetMetadataObjects/Objects/PhysicalTable/TablePackage/SASLibrary/
@Libref
     </PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>10</LENGTH>
   </COLUMN>

   <COLUMN name="Engine">
     <PATH>

/GetMetadataObjects/Objects/PhysicalTable/TablePackage/SASLibrary/
@Engine
     </PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>10</LENGTH>
   </COLUMN>

   </TABLE>

</SXLEMAP>
;

/* Optional: print XML mapped data sets before the merge. */

title 'Tables and their Columns';
proc print data=myxml.ColumnDetails;
run;

title 'Tables and their Librefs';
proc print data=myxml.LibrefDetails;
run;


/* Create data sets that contain the metadata */

libname mybase base 'c:\myxml\data';

data mybase.ColumnDetails;
```

```
  set myxml.ColumnDetails;
run;

data mybase.LibrefDetails;
  set myxml.LibrefDetails;
run;

/* Sort by table name.  */

proc sort data=mybase.ColumnDetails out=mybase.ColumnDetails;
  by  SASTableName;
run;

proc sort data=mybase.LibrefDetails out=mybase.LibrefDetails;
   by SASTableName;
run;

/* Merge into one data set. */

data mybase.final;
   merge mybase.ColumnDetails mybase.LibrefDetails ;
   by SASTableName;
run;
```

After ColumnDetails and LibrefDetails are merged into the final data set, an ODS step creates the HTML report:

```
title 'Table Metadata';
filename reports 'c:\myxml\reports\';

proc print data=mybase.final;
run;
```

Here is a portion of the report:

Table Metadata

Obs	SASTableName	Columns	ColumnID	Libref	Engine
1	AUTO	State	A5JTOPDN.B500000X	mylib	BASE
2	AUTO	Region	A5JTOPDN.B500000Y	mylib	BASE
3	AUTO	Division	A5JTOPDN.B500000Z	mylib	BASE
4	AUTO	Type	A5JTOPDN.B5000010	mylib	BASE
5	AUTO	Total	A5JTOPDN.B5000011	mylib	BASE
6	CENSUS	Density	A5JTOPDN.B5000012	mylib	BASE
7	CENSUS	CrimeRate	A5JTOPDN.B5000013	mylib	BASE
8	CENSUS	State	A5JTOPDN.B5000014	mylib	BASE
9	CENSUS	PostalCode	A5JTOPDN.B5000015	mylib	BASE
10	CENSUS1990	Density	A5JTOPDN.B5000016	mylib	BASE
11	CENSUS1990	CrimeRate	A5JTOPDN.B5000017	mylib	BASE
12	CENSUS1990	State	A5JTOPDN.B5000018	mylib	BASE
13	CENSUS1990	PostalCode	A5JTOPDN.B5000019	mylib	BASE
14	EDUCATION	State	A5JTOPDN.B500001A	mylib	BASE
15	EDUCATION	Code	A5JTOPDN.B500001B	mylib	BASE
16	EDUCATION	DropoutRate	A5JTOPDN.B500001C	mylib	BASE
17	EDUCATION	Expenditures	A5JTOPDN.B500001D	mylib	BASE
18	EDUCATION	MathScore	A5JTOPDN.B500001E	mylib	BASE
19	EDUCATION	Region	A5JTOPDN.B500001F	mylib	BASE

For examples of other types of information that you can obtain with PROC METADATA, see Chapter 11, “METADATA Procedure,” on page 87.

Example: Creating a Report with the DATA Step

This example creates an HTML report about servers that are defined in the repository.

```
%macro server_report (metaserver=abc.company.com,
                      metaport=8561,
                      usr=myuserid,
                      pw=mypasswd,
                      includeopt=N,
                      htmlloc=c:\reports\myservers.htm
                      );

options metaserver="&metaserver"
        metarepository="Foundation"
        metaport=&metaport
        metauser="&usr"
        metapass="&pw";

data _null_;
   length ver $20;
   ver=left(put(metadata_version(),8.));
   put ver=;
   call symput('METAVER',ver);
run;

/* could not connect to metadata server */
%if %eval(&metaver<=0) %then
   %do;
      %put ERROR: could not connect to &metaserver &metaport. ;
      %put ERROR: check connection details, userid and password.;
      %return;
   %end;


      data
      server_connections(keep=id name vendor productname softwareversion
                         hostname port con_name app_pro com_pro authdomain)
      server_options (keep=name server_opts)
      ;
         length mac_uri dom_uri con_uri urivar uri $500
         id  $17 name vendor productname $50 softwareversion $10 port $4
         authdomain authdesc hostname con_name $40
         app_pro com_pro propname $20 pvalue pdesc $200 server_opts $500
         assn attr value $200;
         nobj=1;
         n=1;

         nobj=metadata_getnobj("omsobj:ServerComponent?@Id contains '.'",n,uri);
         do i=1 to nobj;
            nobj=metadata_getnobj("omsobj:ServerComponent?@Id contains
'.'",i,uri);
            put name=;
            put '-----------------------------------';

            rc=metadata_getattr(uri,"Name",Name);
            rc=metadata_getattr(uri,"id",id);
            rc=metadata_getattr(uri,"vendor",vendor);
            rc=metadata_getattr(uri,"productname",productname);
            rc=metadata_getattr(uri,"softwareversion",softwareversion);
```

```
            hostname=' ';

            nummac=metadata_getnasn(uri,
                                    "AssociatedMachine",
                                    1,
                                    mac_uri);
            if nummac then

                  rc=metadata_getattr(mac_uri,"name",hostname);


            numcon=metadata_getnasn(uri,
                                    "SourceConnections",
                                    1,
                                    con_uri);

            port=' ';
            con_name=' ';
            app_pro=' ';
            com_pro=' ';

            if numcon>0 then

                  do k=1 to numcon;
                     numcon=metadata_getnasn(uri,
                                             "SourceConnections",
                                             k,
                                             con_uri);
                     /* Walk through all the notes on this machine object. */
                     rc=metadata_getattr(con_uri,"port",port);
                     rc=metadata_getattr(con_uri,"hostname",hostname);
                     rc=metadata_getattr(con_uri,"name",con_name);
                     rc=metadata_getattr(con_uri,"applicationprotocol",app_pro);

rc=metadata_getattr(con_uri,"communicationprotocol",com_pro);

                     numdom=metadata_getnasn(con_uri,
                                             "Domain",
                                             1,
                                             dom_uri);
                    put numdom=;
                    if numdom >=1 then
                       do;
                          rc=metadata_getattr(dom_uri,"name",authdomain);
                          rc=metadata_getattr(dom_uri,"desc",authdesc);
                       end;
                    else
                       authdomain='none';

                    put authdomain=;
                    output server_connections;
                  end;


            else
               do;
```

```
            put 'Server with no connections=' name;
            if hostname ne ' ' then
               output server_connections;
         end;

      server_opts='none';
      numprop=metadata_getnasn(uri,
                               "Properties",
                               1,
                               con_uri);
      do x=1 to numprop;
         numcon=metadata_getnasn(uri,
                                 "Properties",
                                 x,
                                 con_uri);
         /* Walk through all the notes on this machine object. */
         rc=metadata_getattr(con_uri,"propertyname",propname);
         rc=metadata_getattr(con_uri,"name",pdesc);
         rc=metadata_getattr(con_uri,"defaultvalue",pvalue);
         server_opts=cat(trim(pdesc),' : ',trim(pvalue));

         output server_options;

      end;

   end;

run;

proc sort data=server_connections;
   by name;
run;

proc sort data=server_options;
   by name;
run;

proc transpose data=server_options out=sopts prefix=opt;
   by name ;
   var server_opts;
run;

%if &includeopt=Y %then
   %do; /* include server options on the report */
      data server_report;
         length server_opts $70.;
         merge server_connections server_options;
         by name;
      run;
   %end; /* include server options on the report */

%else
   %do;
      data server_report;
         length server_opts $1.;
         set server_connections;
```

```
            run;
         %end;

         ods listing close;
         ods html body="&htmlloc";
            title "Report for Metadata Server &metaserver:&metaport,
&sysdate9";
            footnote ;

         proc report data=server_report
              nowindows headline headskip split='*' nocenter;
            column name vendor productname softwareversion hostname port
                   con_name app_pro com_pro authdomain
                   %if &includeopt=Y %then
                      %do; /* include server options on the report */
                         server_opts
                      %end; /* include server options on the report */
                   ;
            define name            / group flow missing "Server*Name";
            define vendor          / group flow missing "Vendor";
            define productname     / group flow missing "Product";
            define softwareversion / group missing "Version";
            define port            / group missing "Port";
            define hostname        / group missing "Host Name";
            define con_name        / group missing "Connection*Name";
            define authdomain      / group missing "Authentication*Domain";
            define app_pro         / group missing "App*Protocol";
            define com_pro         / group missing "Com*Protocol";

            %if &includeopt=Y %then
               %do; /* include server options on the report */
                  define server_opts / group missing "Server Options";
               %end; /* include server options on the report */

            break after name  /  style=[BACKGROUND=CCC];

            %if &includeopt=Y %then
               %do; /* include server options on the report */
                  compute after name ;
                  line ' ';
                  line server_opts $70.;
                  line ' ';
                  endcomp;
               %end; /* include server options on the report */

         run;

         ods html close;
         ods listing;

 /* connected to metadata server */


%mend;

%server_report (metaserver=abc.company.com,
```

```
metaport=8561,
usr=sasadm@saspw,
pw=xxxxxx,
includeopt=N,
htmlloc=c:\reports\myservers.htm
);
```

Here is the HTML report:

Report for Metadata Server abc.company.com:8561,04JAN2011

Server Name	Vendor	Product	Version	Host Name	Port	Connection Name	App Protocol	Com Protocol	Authentication Domain
FrameworkServer - SAS Framework Data Server	SAS Institute	SAS Framework Data Server	2.1	abc.company.com	2203	Connection: SAS Framework Data Server	Bridge	TCP	none
Object Spawner - vmw0116	SAS Institute	SAS Workspace Spawner	9.3		8801	A51EZTIPPortBank_0	PortBank	TCP	none
					8811	A51EZTIPPortBank_1	PortBank	TCP	none
					8821	A51EZTIPPortBank_2	PortBank	TCP	none
				abc.company.com	8581	Connection: Object Spawner - vmw0116	Operator	TCPIP	DefaultAuth
Operating System Services - abc.company.com	SAS Institute	OS	9.3	abc.company.com	8451	Connection: Operating System Services -	Bridge	TCP	DefaultAuth
SAS Content Server	SAS Institute	Http Server	9.3	abc.company.com	8080	Connection: SAS Content Server	http	tcp	DefaultAuth
SAS Distributed In-Process Services Scheduling Ser	SAS Institute Inc.	SAS Distributed In-Process Services	1.0	abc.company.com					DefaultAuth
SAS In-Process Services	SAS Institute Inc.	SAS In-Process Services	1.0	abc.company.com					DefaultAuth
SASApp - Pooled Workspace Server	SAS Institute	SAS Pooled Workspace	9.3	abc.company.com	8701	Connection: SASApp - Pooled Workspace Se	Bridge	TCP	DefaultAuth
SASApp - Stored Process Server	SAS Institute	SAS Stored Process	9.3	abc.company.com	8601	Connection: SASApp - Stored Process Serv	Bridge	TCP	DefaultAuth
SASApp - Workspace Server	SAS Institute	SAS Workspace	9.3	abc.company.com	8591	Connection: SASApp - Workspace Server	Bridge	TCP	DefaultAuth
SASMeta - Metadata Server	SAS Institute	Metadata Server	9.3	abc.company.com	8561	Connection: SASMeta - Metadata Server	Bridge	TCP	DefaultAuth
						Connection: SASMeta - Metadata Server (C	SBIP	TCP	DefaultAuth
SASMeta - SAS DATA Step Batch Server			9.3	abc.company.com					
SASMeta - Workspace Server	SAS Institute	SAS Workspace	9.3	abc.company.com	8591	Connection: SASMeta - Workspace Server	Bridge	TCP	DefaultAuth

For more examples of reports that you might want to create with SAS metadata DATA step functions, see “Examples: DATA Step Functions for Reading Metadata” on page 198.

Example: Creating Metadata for a JSON

Many web sites offer REST APIs that enable you to download data as JSON. You can download JSON from the web with SAS by using PROC HTTP. You can read JSON with SAS by using the JSON LIBNAME engine. Beginning with SAS 9.4M7, you can create metadata for a JSON by using PROC METALIB. You can also use metadata to manipulate a JSON with the metadata LIBNAME engine.

This example uses a JSON obtained from openweathermap.org to illustrate the process of downloading, reading, and creating metadata for a JSON. Then it shows a simple way that you might you use the metadata LIBNAME engine to manipulate the data described by the metadata. The example downloads the One Call API that is available from the website. This API provides daily weather information based on geographical coordinates. This example uses the coordinates for Austin, Texas. The API is executed with settings that return temperatures as Fahrenheit and wind speeds in miles per hour. For more information about this API, see https://openweathermap.org/.

Before using PROC HTTP, create a download directory and submit a FILENAME statement to assign a fileref to the location. In the FILENAME statement, specify a name for the downloaded JSON file. The FILENAME statement in this example assigns the fileref Response to a file named Weather.json that is created in the C:/weather directory. PROC HTTP specifies the URL for the API request in the URL= argument, specifies the GET method, and writes output to fileref Response.

```
filename response 'C:/weather/weather.json';

proc http  url="https://api.openweathermap.org/data/2.5/onecall?
lat=30.266666
&lon=-97.733330&exclude=minutely&units=imperial
 &appid=a8c4f692a4f2b7a9842f7d03e54f3821"
  method= "GET"
  out=response;
run;
```

The following LIBNAME statement enables you to read the JSON with the JSON engine:

```
libname weather JSON fileref="response";
```

The code assigns the libref Weather to the JSON, specifies the JSON engine, and specifies fileref Response to supply input to the engine. When you specify no other parameters in a JSON engine LIBNAME statement, the engine scans the target directory for a map file. If a map file is found, the engine uses it to read the JSON. If a map cannot be found, the automapper creates one.

To view the tables in the JSON library, use PROC DATASETS:

```
proc datasets lib=weather;
run;
quit;
```

Here is the output from the PROC DATASETS request:

Figure 4.1 *Data Sets in JSON Library Weather*

Directory	
Libref	WEATHER
Engine	JSON
Access	READONLY
Physical Name	C:\weather\weather.json

#	Name	Member Type
1	ALLDATA	DATA
2	CURRENT	DATA
3	CURRENT_WEATHER	DATA
4	DAILY	DATA
5	DAILY_FEELS_LIKE	DATA
6	DAILY_TEMP	DATA
7	DAILY_WEATHER	DATA
8	HOURLY	DATA
9	HOURLY_WEATHER	DATA
10	ROOT	DATA

Weather.json contains nine tables that describe the ordinals in the JSON and an ALLDATA table, which consolidates information about the ordinals in one data set. The ALLDATA data set also includes properties that you can use to manipulate the data. This document does not illustrate the use of those properties. For information about the properties, see the JSON engine documentation in *SAS Global Statements: Reference*.

For this example, we use JSON tables Daily and Daily_Temp. To view the properties of the tables, use PROC CONTENTS.

```
proc contents data=weather.daily;
run;

proc contents data=weather.daily_temp;
run;
```

Here are the properties of table Weather.Daily.

The CONTENTS Procedure

Data Set Name	WEATHER.DAILY	Observations	.
Member Type	DATA	Variables	13
Engine	JSON	Indexes	0
Created	.	Observation Length	104
Last Modified	.	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	Default		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
11	clouds	Num	8
8	dew_point	Num	8
3	dt	Num	8
7	humidity	Num	8
2	ordinal_daily	Num	8
1	ordinal_root	Num	8
6	pressure	Num	8
13	rain	Num	8
4	sunrise	Num	8
5	sunset	Num	8
12	uvi	Num	8
10	wind_deg	Num	8
9	wind_speed	Num	8

Here are the properties of table Weather.Daily_Temp:

The CONTENTS Procedure

Data Set Name	WEATHER.DAILY_TEMP	Observations	.
Member Type	DATA	Variables	8
Engine	JSON	Indexes	0
Created	.	Observation Length	64
Last Modified	.	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	Default		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	day	Num	8
7	eve	Num	8
5	max	Num	8
4	min	Num	8
8	morn	Num	8
6	night	Num	8
1	ordinal_daily	Num	8
2	ordinal_temp	Num	8

To view the observations in the daily data sets, use PROC PRINT.

```
proc print data=weather.daily;
 run;

proc print data=weather.daily_temp;
 run;
```

Here is the PROC PRINT output for table Weather.Daily.

Figure 4.2 *PROC PRINT Output for Table Weather.Daily*

Obs	ordinal_root	ordinal_daily	dt	sunrise	sunset	pressure	humidity	dew_point	wind_speed	wind_deg	clouds	uvi	rain
1	1	1	1593712800	1593689596	1593740194	1015	41	67.82	11.65	185	1	12.63	.
2	1	2	1593799200	1593776021	1593826590	1015	28	57.76	7.52	196	4	13.74	.
3	1	3	1593885600	1593862446	1593912985	1013	22	56.07	4.00	245	3	11.68	.
4	1	4	1593972000	1593948873	1593999378	1010	25	58.21	7.76	319	30	11.60	0.28
5	1	5	1594058400	1594035300	1594085770	1011	27	60.78	4.70	209	12	11.77	.
6	1	6	1594144800	1594121727	1594172160	1013	33	64.17	9.60	185	0	11.86	.
7	1	7	1594231200	1594208156	1594258549	1014	51	70.30	10.89	183	68	11.87	.
8	1	8	1594317600	1594294585	1594344937	1015	36	65.91	12.48	174	50	11.70	.

Here is the PROC PRINT output for table Weather.Daily_Temp.

Obs	ordinal_daily	ordinal_temp	day	min	max	night	eve	morn
1	1	1	95.23	80.42	95.61	80.42	95.61	95.23
2	2	2	96.06	74.71	99.84	82.11	96.39	74.71
3	3	3	100.69	77.18	104.97	85.48	101.53	77.18
4	4	4	100.35	79.70	103.77	84.92	100.47	79.70
5	5	5	100.51	81.03	100.62	81.41	96.51	81.03
6	6	6	97.77	76.82	101.30	80.58	97.45	76.82
7	7	7	90.90	78.15	97.95	79.54	95.18	78.15
8	8	8	97.30	79.50	101.37	79.50	97.39	79.68

Before you can create metadata, you must create a library definition for the JSON file in the SAS Metadata Repository. You can create a library definition in the metadata repository by using SAS Management Console or SAS Data Integration Studio. For instructions to create a library definition, see “Establishing Connectivity to a JSON File” in *SAS Intelligence Platform: Data Administration Guide*.

After you have defined the library in the metadata repository, you can create table metadata. To create table metadata. you can use the Register Tables feature that is available in the products. Or, you can use the SAS METALIB procedure. This example uses PROC METALIB. A library definition is represented in a SAS Metadata Repository by a SASLibrary object. The METALIB procedure takes the name or the metadata identifier of a SASLibrary object as input. In this example, the SASLibrary object is named Weather. Then, it creates table objects, as specified, for the specified SASLibrary object.

The following PROC METALIB code creates metadata objects for tables Weather.Daily and Weather.Daily_Temp in the SAS Metadata Repository:

```
proc metalib;
   omr=(library=weather server="computer.company.com" port="8561"
user="yourid"
pw="urpassword");
   select ("daily" "daily_temp");
   folder="/User Folders/yourid/My Folder/Weather";
report;
run;
```

By default, PROC METALIB creates metadata in the location stored in the library definition. This example specifies the FOLDER statement to store the metadata in a personal folder instead.

The procedure prints output to the log, by default. The REPORT statement prints output to Results. Here is the output of the REPORT statement:

The METALIB Procedure

Summary Report for Library WEATHER
Repository Foundation
24JUL2020

Metadata Summary Statistics	
Total tables analyzed	2
Tables Updated	0
Tables Added	2
Tables matching data source	0
Tables not found	0
Tables not processed	0

Tables Added		
Metadata Name	**Metadata ID**	**SAS Name**
DAILY	A5NQDMHG.B8000025	daily
DAILY_TEMP	A5NQDMHG.B8000026	daily_temp

The procedure created table objects for the tables specified in the SELECT statement only.

After creating the table objects in the repository, examine the properties of the objects in SAS Management Console or SAS Data Integration Studio. Only users who have a metadata identify defined on the SAS Metadata Server can read the table objects. You can limit the number further by defining permissions on the objects' **Authorization** tab.

The following is an example of how you manipulate metadata objects in an application with the metadata LIBNAME engine.

```
libname mle meta library="weather";

data weather;
  merge mle.daily mle.daily_temp;
  by ORDINAL_daily;
run;
```

The LIBNAME statement specifies the metadata LIBNAME engine (META), the name of the target SASLibrary object (Weather), and associates a libref (MLE) with the connection. The libref MLE is used to identify the tables in the DATA step.

This particular DATA step creates a temporary SAS data set also named Weather by merging the Daily and Daily_Temp tables. The tables are merged using the variable ORDINAL_Daily. The JSON engine includes at least one shared ordinal in every output table to enable you to merge the tables in the JSON.

The JSON engine is a read-only, read-once engine. The JSON tables cease to exist at the end of the SAS session. They are re-created the next time the library is assigned. Meanwhile, the library and table metadata persist. The SASLibrary object reassigns the library using the map that was defined.. If you choose to create a new or custom map file, be sure to create a new library definition and table objects in the SAS Metadata Repository.

PART 2

System Options

5

Introduction to System Options for Metadata

Overview of System Options for Metadata

SAS provides a family of system options to define the default SAS Metadata Server. The following table shows the system options by category.

Table 5.1 *System Options by Category*

Category	System Options
Connection	METACONNECT= METAPASS= METAPORT= METAPROFILE METAPROTOCOL= METAREPOSITORY=

Category	System Options
	METASERVER= METASPN= METAUSER=
Encryption	METAENCRYPTALG METAENCRYPTLEVEL
Resource	METAAUTORESOURCES

To determine what system option settings are active in your SAS session, you can issue the OPTIONS command on the command line. Or submit the following procedure statement:

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

To return information about the options' value and scope, including how they were set, set the **value** option:

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

Usually, these system options are set in a configuration file or at invocation. Some of the options can be changed at any time; see the options documentation. The metadata system options affect every server that uses an Integrated Object Model (IOM) connection to the metadata server. IOM servers include the SAS Workspace Server, SAS Pooled Workspace Server, SAS Stored Process Server, and SAS OLAP Server as well as any Base SAS session that connects to the metadata server.

For general information about SAS system options, see the *SAS Language Reference: Concepts*. For information about configuration files, see the SAS Companion for your operating environment. For information about administration, see *SAS Intelligence Platform: System Administration Guide*.

Connection Options

Introduction to Connection Options

The connection properties are required to establish a connection to the metadata server. You can establish a connection in the following ways:

- Set the connection properties directly with the METASERVER=, METAPORT=, METAUSER=, METAPASS=, and METAREPOSITORY= system options, or the METASERVER=, METAPORT=, and METASPN= system options; see "Specifying Connection Properties Directly" on page 39.
- Specify a stored metadata server connection profile with the METAPROFILE and METACONNECT= options. This is the recommended way to connect to the

metadata server beginning in SAS 9.4. See “Specifying a Stored Connection Profile” on page 40.

- Specify connection properties when you issue a metadata procedure; see “Overview of Procedures for Metadata” on page 83.
- Specify connection properties when you issue the metadata LIBNAME statement; see “LIBNAME Statement for the Metadata Engine ” on page 69.
- When you are running interactively, you can be prompted for connection values. Prompting occurs when either METASERVER= or METAPORT= are not specified. Prompting also occurs when METAUSER= or METAPASS= are not specified, and a trusted peer or Integrated Windows authentication (IWA) connection is rejected. For information about trusted peer and IWA connections, see *SAS Intelligence Platform: Security Administration Guide*.

If the connection fails, check the connection properties to be sure you have specified or omitted quotation marks exactly as documented.

Specifying Connection Properties Directly

Connection Options

The METAPASS=, METAPORT=, METAPROTOCOL=, METAREPOSITORY=, METASERVER=, METASPN=, and METAUSER= system options each specify a connection property. Typically, these values are set in a configuration file. METAPROTOCOL= is optional as there is currently only one supported value, which is the default.

Example: Configuration File

To set the default metadata server to use the password **sasuser1**, port **8561**, repository **Foundation**, metadata server **a123.us.company.com**, and user ID **myuserid**, you would add the following lines to the configuration file:

```
-METAPASS  "sasuser1"
-METAPORT  8561
-METAREPOSITORY "Foundation"
-METASERVER "a123.us.company.com"
-METAUSER  "myuserid"
```

Example: OPTIONS Statement

The following OPTIONS statement, which can be added to an autoexec file or directly to a SAS program, has the same effect as the configuration file example:

```
options metapass="sasuser1"
        metaport=8561
```

```
metarepository="Foundation"
metaserver="a123.us.company.com"
metauser="myuserid";
```

Specifying a Stored Connection Profile

Connection Options

Instead of specifying individual connection options for the metadata server, you can use the METAPROFILE option. The METAPROFILE option invokes a connection profile that is stored in an XML document. The stored connection profile contains metadata server connection properties, such as the name of the host computer on which the metadata server is invoked and the TCP port. A profile can also contain the user ID and password of the requesting user and the default metadata repository, although this information is not required. This information can be supplied later with the METAUSER=, METAPASS=, and METAREPOSITORY= system options.

METAPROFILE is configured before SAS is started or specified at SAS invocation. SAS recommends use of a profile along with the direct connection options in clustered SAS Metadata Server configurations.

Reasons to use METAPROFILE include:

- Beginning in SAS version 9.4, configuration processes create a default server connection profile that can be used in METAPROFILE. This default server connection profile contains the metadata server host name and port number of the SAS Metadata Server or metadata server nodes at the time of initial configuration.
- When the default server connection profile is active, the SAS session is aware of the host and port values of all of the server nodes in the cluster, and uses the information to assist in server connection. The connection logic first looks at any METASERVER= and METAPORT= values that are specified and tries to connect. If the connection fails, it consults the profile. If the profile contains a cluster definition *and* that definition includes the node specified in METASERVER= and METAPORT=, then the SAS session will try the other nodes in the cluster. The profile provides fallback information in case of a cluster node failure. Use of the METASERVER= and METAPORT= options without METAPROFILE will not handle cluster node failures.
- Use of the default server connection profile does not prevent users from also specifying direct connection options. The direct server connection options take precedence over the properties in the stored connection profile. That is, if you specify METAUSER= and METAPASS= options in the SAS session and the profile contains credentials, the credentials in the profile are ignored. If you specify a different metadata server in the METASERVER= and METAPORT= options, the server(s) in the profile are ignored. SAS will not redirect connections for servers that are not part of the cluster definition.

By default, the SAS system uses the first connection profile in the XML document specified in METAPROFILE. Most XML documents contain one connection profile. The METACONNECT= system option enables you to specify a connection profile in

XML documents that contain more than one connection profile. Setting METACONNECT=NONE disables METAPROFILE processing. You might want to disable METAPROFILE processing to prevent re-routing of administrative requests that must be executed on a specific server node. For more information, see "METACONNECT= System Option" on page 45.

In most cases, there is no need to override the default server connection profile.

Default Server Connection Profile

The default server connection profile is a shared XML document named metadataConfig.xml that is created in the top-level configuration directory of every computer that hosts a SAS Metadata Server and every configuration directory that defines a SAS server. This XML document contains a single connection profile that lists the host and port address(es) of the metadata server or metadata server nodes at the time of configuration.

The default server connection profile does not contain user ID and password information or a default metadata repository. If needed, you can supply these in a configuration file, at SAS invocation, or as system options, with the METAUSER=, METAPASS=, and METAREPOSITORY= options.

Example: Configuration File

To configure access to the default server connection profile, you would add the following line to the configuration file:

```
-METAPROFILE "C:\<sas-config>\Lev1\metadataConfig.xml"
```

Example: SAS Invocation

Here is an example of how the default server connection profile could be specified at SAS invocation:

```
<sas-cmd> -metaprofile 'C:\<sas-config>\Lev1\metadataConfig.xml'
-metauser 'myuserid' -metapass 'sasuser1'
```

User-Defined Connection Profiles

The METAPROFILE option supports the use of XML documents that contain user-defined connection profiles. A user-defined connection profile is a connection profile that is created with the Metadata Server Connections dialog box. This dialog box enables you to save (export) one or more user-defined connection profiles to a permanent XML document. This dialog box is opened by executing the SAS windowing environment command `METACON`. For more information about the METACON command, see the Help in the SAS windowing environment.

Note: The Metadata Server Connections dialog box currently does not support editing user-defined connection profiles that contain a cluster definition.

User-defined connection profiles should not be created in the metadataConfig.xml file.

Example: Configuration File

Here is a configuration file example that invokes a user-defined connection profile named Mike's profile:

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

Encryption Options

The METAENCRYPTALG and METAENCRYPTLEVEL options are used to encrypt communication with the metadata server. You do not have to license SAS/SECURE software if you specify the SAS proprietary algorithm. For more information, see “METAENCRYPTALG System Option” on page 46 and “METAENCRYPTLEVEL System Option” on page 48.

Resource Option

The METAAUTORESOURCES option identifies resources to be assigned at SAS start-up. The resources are defined in SAS metadata. For example, in SAS Management Console, you can define a list of librefs (SAS library references) that are associated with the LogicalServer, ServerComponent, or ServerContext object. METAAUTORESOURCES points to the object and assigns the associated libraries at start-up.

For more information, see “METAAUTORESOURCES System Option” on page 43.

6

System Options for Metadata

Dictionary

METAAUTORESOURCES System Option

Identifies the metadata resources that are assigned when a SAS server is started.

Valid in:	configuration file, SAS invocation
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META

Syntax

METAAUTORESOURCES *server-object*

Syntax Description

server-object
is the name or URI of a LogicalServer, ServerComponent, or ServerContext metadata object in a repository on the SAS Metadata Server. The maximum length is 32,000 characters. If you specify either single or double quotation marks, they are not saved as part of the value.

METAAUTORESOURCES accepts the following name and URI formats:

name
specifies the metadata name of the object. An example is the following:

```
-metaautoresources 'SASApp'
```

This format is supported for a ServerContext object only. For LogicalServer and ServerComponent objects, use one of the following URI formats:

OMSOBJ:*identifier.identifier*
specifies the metadata identifier of the object. An example is the following:

```
-metaautoresources "omsobj:A5HMMB7P.AV000005"
```

OMSOBJ:*type*/*ID*
specifies the metadata type name and metadata identifier of the object. An example is the following:

```
-metaautoresources "omsobj:ServerComponent/A5HMMB7P.AV000005"
```

OMSOBJ:*type*?@*attribute*='*value*'
specifies the metadata type name, followed by a search string, which is in the form of an *attribute*='*value*' pair. An example is the following:

```
-metaautoresources "OMSOBJ:ServerComponent?@Name='My Server'"
```

Details

METAAUTORESOURCES identifies metadata resources that are assigned when you start the SAS Metadata Server. In this release, the option is used to assign libraries. In future releases, additional resources might be supported.

In SAS Management Console, when you define a library, you can assign a server. METAAUTORESOURCES specifies the server object and assigns the associated libraries at start-up.

The SAS Management Console Data Library Manager allows administrators to specify the pre-assignment source in the library definition. In SAS 9.3, the PreAssignmentType property affected METAAUTORESOURCES as follows:

- Libraries marked as pre-assigned by external configuration (for example, the AUTOEXEC file) were ignored by METAAUTORESOURCES.
- Libraries marked as pre-assigned by the native library engine were assigned using the native library engine defined for that library in metadata.

- Libraries marked as pre-assigned by the metadata LIBNAME engine were assigned by the metadata LIBNAME engine. The metadata LIBNAME engine is a proxy library engine that enforces access controls placed on the library and its tables and columns as defined in metadata.

In SAS 9.4, the METAAUTORESOURCES system option is checked first at server start-up. Any libraries that are marked in metadata as pre-assigned by the native library engine or by the metadata LIBNAME engine are assigned by the SAS server. Libraries specified in the AUTOEXEC file are assigned after the METAAUTORESOURCES assignments have completed.

The METAAUTORESOURCES option is set automatically for any SAS Workspace Server or SAS Stored Process Server started by the object spawner. The option can be set manually for any other batch, interactive, or server SAS session using a command-line or configuration file option. For information about pre-assigning SAS libraries, see the *SAS Intelligence Platform: Data Administration Guide*.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

METACONNECT= System Option

Identifies one named profile from the metadata connection profiles for connecting to the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Default:	Blank-value
Requirement:	Valid only when METAPROFILE is set

Syntax

METACONNECT= "blank-value" | *named-connection* | NONE

Syntax Description

“blank-value”
specifies the first connection profile in the XML document specified in METAPROFILE. This is the default value when the METAPROFILE option is specified and METACONNECT= is not used.

named-connection
is the name of a connection profile in the XML document specified in METAPROFILE. The maximum length is 256 characters. Quotation marks are optional.

NONE
is a special option that disables METAPROFILE processing.

Details

This system option is one of a category of system options that define a connection to the SAS Metadata Server.

METACONNECT= specifies the name of a connection profile in the XML document specified in METAPROFILE. Most XML documents contain only one named connection profile, so METACONNECT= is not required.

METACONNECT=NONE is provided to turn off the re-routing of connections that occurs when the default server connection profile is active in clustered SAS Metadata Server configurations. This option ensures that administrative requests, like stopping one server from the cluster, fail if they cannot be executed on the intended server node. For an example of how METACONNECT=NONE is specified, see “Example 8: Stop One Server in a Metadata Server Cluster” on page 185.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

- “Example: Configuration File” on page 41

System Options

- “METAPROFILE System Option” on page 51

METAENCRYPTALG System Option

Specifies the type of encryption to use when communicating with the SAS Metadata Server.

Valid in:	configuration file, SAS invocation
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Alias:	METAENCRYPTALGORITHM
Default:	SASPROPRIETARY

Syntax

METAENCRYPTALG *algorithm* | NONE

Syntax Description

algorithm
specifies the algorithm that SAS clients use to communicate with the SAS Metadata Server. The following algorithms can be used:

- RC2
- RC4
- DES
- TripleDES
- SAS Proprietary (alias SAS)
- AES

NONE
Does not specify an encryption algorithm.

Details

The SAS IOM supports encrypted communication with the metadata server. Use the METAENCRYPTALG and METAENCRYPTLEVEL system options to define the type and level of encryption that SAS clients use when they communicate with the metadata server.

If you specify an encryption algorithm other than SAS Proprietary (alias SAS), you must have SAS/SECURE software. SAS/SECURE is a product in the SAS System. In SAS 9.4, SAS/SECURE is included with Base SAS software. In prior releases of SAS, SAS/SECURE was an add-on product that was licensed separately. This change makes strong encryption available in all deployments (except where prohibited by import restrictions).

For more information about the encryption algorithms, see the *Encryption in SAS*.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

System Options

- “METAENCRYPTLEVEL System Option” on page 48

METAENCRYPTLEVEL System Option

Specifies the level of encryption when communicating with the SAS Metadata Server.

Valid in:	configuration file, SAS invocation
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Default:	CREDENTIALS

Syntax

METAENCRYPTLEVEL EVERYTHING | CREDENTIALS

Syntax Description

EVERYTHING
specifies to encrypt all communication with the metadata server.

CREDENTIALS
specifies to encrypt only login credentials. This is the default.

Details

The SAS IOM supports encrypted communication with the metadata server. Use the METAENCRYPTLEVEL and METAENCRYPTALG system options to define the level and type of encryption that SAS clients use when they communicate with the metadata server.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

System Options

- “METAENCRYPTALG System Option” on page 46

METAPASS= System Option

Specifies the password for the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Note:	The OPTIONS procedure displays passwords in the SAS log as eight Xs, regardless of the actual password length.

Syntax

METAPASS= "*password*"

Syntax Description

"*password*"
is the password for the user ID on the metadata server. The maximum length is 512 characters. The quotation marks are optional.

Note We recommend that you encode the password before specifying it in METAPASS=. To encode the password, use the PWENCODE procedure. Specify SAS005 encryption to disguise the text string. The metadata server decodes the encoded password. For more information, see the PWENCODE procedure in *Base SAS Procedures Guide*.

Details

This system option is one of a category of system options that define a connection to the metadata server.

When you are running interactively, you can be prompted for connection properties. Prompting occurs when either METASERVER= or METAPORT= are not specified. Prompting also occurs when METAUSER= or METAPASS= are not specified, and a trusted peer or IWA connection is rejected. For information about trusted peer and IWA, see the *SAS Intelligence Platform: Security Administration Guide*.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

- “Example: Configuration File” on page 39

System Options

- “METAPORT= System Option” on page 50
- “METASERVER= System Option” on page 54
- “METAUSER= System Option” on page 57

METAPORT= System Option

Specifies the TCP port for the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Range:	1–65535

Syntax

METAPORT=*number*

Syntax Description

number
is the TCP port that the metadata server is listening to for connections. The default port number that is configured for the metadata server at installation is 8561. Installers are not required to use this value, so you must specify METAPORT= to connect to the metadata server. An example is `metaport=8561`. Do not quote this value.

Details

This system option is one of a category of system options that define a connection to the metadata server.

When you are running interactively, you can be prompted for connection values. Prompting occurs when either METASERVER= or METAPORT= are not specified. Prompting also occurs when METAUSER= or METAPASS= are not specified, and a trusted peer or IWA connection is rejected. For information about trusted peer and IWA, see the *SAS Intelligence Platform: Security Administration Guide*.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

- “Example: Configuration File” on page 39

System Options

- “METAPASS= System Option” on page 49
- “METASERVER= System Option” on page 54
- “METASPN= System Option” on page 55
- “METAUSER= System Option” on page 57

METAPROFILE System Option

Identifies the XML document that contains connection profiles for the SAS Metadata Server.

Valid in:	configuration file, SAS invocation
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META

Syntax

METAPROFILE "*XML-document*"

Syntax Description

"*XML-document*"

is the pathname of the XML document that contains connection profiles for connecting to the SAS Metadata Server. The pathname is the physical location that is recognized by the operating environment. Quotation marks are required.

Beginning in SAS 9.4, all configurations that include a SAS Metadata Server define a shared file named metadataConfig.xml file in the top-level configuration directory and in every configuration directory that contains a SAS server. This metadataConfig.xml file contains a single connection profile that lists the host and port addresses of the metadata server or metadata server nodes at the time of configuration. The connection profile in the metadataConfig.xml file is referred to as the “default server connection profile”. Use of the metadataConfig.xml file is recommended for clustered SAS Metadata Server configurations, but it is not required.

Details

This system option is one of a category of system options that define a connection to the SAS Metadata Server. Instead of specifying individual connection options for the metadata server, you can use the METAPROFILE option.

METAPROFILE must be configured before SAS is started or specified at SAS invocation. It specifies the pathname of the XML document that contains one or more connection profiles. A connection profile contains metadata server connection properties, such as the name of the host computer on which the metadata server is invoked, the TCP port, and (optionally) the user ID and password of the requesting user.

The SAS System uses the first connection profile in the specified XML document to establish the metadata server connection by default. The METACONNECT= system option might be needed if there is more than one connection profile in the XML document. METACONNECT=NONE is provided to disable METAPROFILE processing. For more information, see METACONNECT=.

The default server connection profile in metadataConfig.xml reflects the initial metadata server configuration. When a metadata server node is added to or removed from the cluster, an administrator must run the sas-upgrade-metadata-profile batch utility to update the default server connection profile. For information about the utility, see *SAS Intelligence Platform: System Administration Guide*.

The default server connection profile does not contain user ID and password information. You can provide this information with the METAUSER= and METAPASS= options.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

- “Example: Configuration File” on page 41

System Options

- “METACONNECT= System Option” on page 45

METAPROTOCOL= System Option

Specifies the network protocol for connecting to the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Default:	BRIDGE

Syntax

METAPROTOCOL=*BRIDGE*

Syntax Description

BRIDGE
specifies that the connection to the metadata server uses the SAS Bridge protocol. In this release, it is the only supported value and the default value, so there is no need to specify this system option.

Details

This system option is one of a category of system options that define a connection to the metadata server.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

"Example: Configuration File" on page 39

METAREPOSITORY= System Option

Specifies the SAS Metadata Repository to use with the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Default:	Foundation

Syntax

METAREPOSITORY= "*name***"**

Syntax Description

"*name*"
is the name of the repository to use. The maximum length is 32,000 characters. The quotation marks are optional.

Details

This system option is one of a category of system options that define a connection to the metadata server.

You can use the $METAREPOSITORY substitution variable in the input XML with PROC METADATA. The variable resolves to the metadata identifier of the repository that is named by this option.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

System Options

- “METAPASS= System Option” on page 49
- “METAPORT= System Option” on page 50
- “METASERVER= System Option” on page 54
- “METAUSER= System Option” on page 57

METASERVER= System Option

Specifies the host name or address of the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META

Syntax

METASERVER= "*address*"

Syntax Description

"*address*"
is the host name or network IP address of the computer that hosts the metadata server. An example is `metaserver="a123.us.company.com"`. The value `localhost` can be used when connecting to a metadata server on the same computer. The maximum length is 256 characters. The quotation marks are optional, unless there are special characters in the address.

Details

This system option is one of a category of system options that define a connection to the metadata server.

When you are running interactively, you can be prompted for connection properties. Prompting occurs when either METASERVER= or METAPORT= are not specified. Prompting also occurs when METAUSER= or METAPASS= are not specified, and a trusted peer or IWA connection is rejected. For information about trusted peer and IWA, see the *SAS Intelligence Platform: Security Administration Guide*.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

- “Example: Configuration File” on page 39

System Options

- “METAPASS= System Option” on page 49
- “METAPORT= System Option” on page 50
- “METASPN= System Option” on page 55
- “METAUSER= System Option” on page 57

METASPN= System Option

Specifies the service principal name (SPN) for the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META
Default:	Generated in the form SAS/*machine-name*

Syntax

METASPN=*SPN-name*

Syntax Description

SPN-name
is the SPN for the principal that runs the metadata server. The maximum length is 256 characters. The following formats are supported for *SPN-name*: `SAS/machine-name` or `SAS/machine-name.company.com`. `SAS` is the name of the service and represents the service type.

Details

Integrated Windows Authentication (IWA) is enabled for a SAS session by specifying the SAS option -SSPI in a configuration file or at SAS invocation. When the SAS session is invoked with the -SSPI option and a user attempts to connect to the SAS Metadata Server without specifying METAUSER= and METAPASS=, the SAS System creates a default SPN and makes an IWA connection. For more information about the default SPN and special requirements for using IWA on UNIX, see *SAS Intelligence Platform: Security Administration Guide*. If the IWA connection fails, the server attempts to make a trusted peer connection.

The METASPN system option is provided for the rare circumstance in which the default SPN cannot be used by the SAS System. This can occur on Windows systems when SAS servers are not being run under the local system account. The local system account can register the default SPN—a domain user account cannot. For more information about SPNs and SAS servers, see *SAS Intelligence Platform: Security Administration Guide*.

METASPN= is used with METASERVER= and METAPORT=. If you specify METAUSER= and METAPASS=, then the METASPN= value is *not* used.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

Example: Default Form

Here is an example that shows a METASPN= value in the default form:

```
-METASERVER "a123.company.com"
-METAPORT 9999
-METASPN "SAS/a123.company.com"
```

See Also

System Options

- “METASERVER= System Option” on page 54
- “METAPORT= System Option” on page 50

METAUSER= System Option

Specifies the user ID for connecting to the SAS Metadata Server.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Communications: Metadata
PROC OPTIONS GROUP=	META

Syntax

METAUSER= "*userid*"

Syntax Description

"*userid*"
is the user ID for connecting to the metadata server. The maximum length is 256 characters. The quotation marks are optional, unless the user ID includes a special character, such as "sasadm@saspw".

Details

This system option is one of a category of system options that define a connection to the metadata server.

When you are running interactively, you can be prompted for connection properties. Prompting occurs when either METASERVER= or METAPORT= are not specified. Prompting also occurs when METAUSER= or METAPASS= are not specified, and a trusted peer or IWA connection is rejected. For information about trusted peer and IWA, see the *SAS Intelligence Platform: Security Administration Guide*.

In a network environment, METAUSER= must specify a fully qualified user ID in the form of SERVERNAME\USERID. For information about user definitions, see the *SAS Intelligence Platform: Security Administration Guide*.

Operating Environment Information: In a configuration file or at SAS invocation, the syntax for SAS system options is specific to your operating environment. For more information, see the SAS documentation for your operating environment.

See Also

- “Example: Configuration File” on page 39

System Options

- “METAPASS= System Option” on page 49
- “METAPORT= System Option” on page 50
- “METASERVER= System Option” on page 54

PART 3

Metadata LIBNAME Engine

7

Introduction to the Metadata LIBNAME Engine

Overview of the Metadata LIBNAME Engine

The metadata engine is similar to other SAS engines. In a batch file or in the SAS windowing environment, you can submit a LIBNAME statement that assigns a libref and the metadata engine. You then use that libref throughout the SAS session where a libref is valid.

However, unlike other librefs, the metadata engine's libref is not assigned to the physical location of a SAS library. The metadata engine's libref is assigned to a set of metadata objects that are registered in the SAS Metadata Server. These metadata objects must already be defined by an administrator with a product like SAS Management Console.

The objects contain the specifications that you would normally submit with a LIBNAME statement. The metadata engine uses the information in the objects to

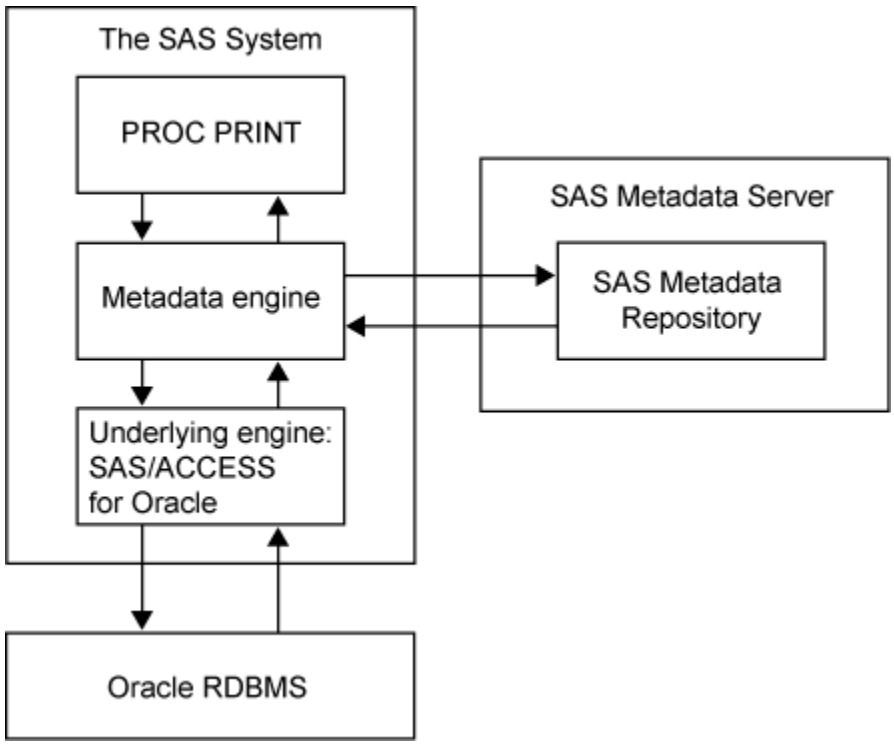
construct a LIBNAME statement that specifies the data source, the engine that processes the data (referred to as the “underlying engine”), and options.

After you submit the metadata LIBNAME statement, you can reference the metadata engine's libref in your SAS code. The metadata engine calls the underlying engine to process the data.

In other words, the metadata LIBNAME statement takes the place of your usual LIBNAME statement and creates the usual LIBNAME statement from information in metadata.

The following diagram illustrates this process. In the example, an Oracle data library is already defined in metadata. You reference the Oracle data library with the metadata LIBNAME statement, and the metadata engine constructs a LIBNAME statement that assigns the SAS/ACCESS interface to Oracle as the underlying engine. Then, when you submit the PRINT procedure, the metadata engine issues a request to the SAS Metadata Repository for the library member's metadata, and uses the Oracle engine to run the PROC PRINT.

Figure 7.1 *Metadata Engine Process*



Supported Features

The metadata engine supports the following features:

- Processes tables and views from SAS and third-party DBMSs (database management systems) by using an underlying engine. The metadata engine

supports only tables and views, and does not support other SAS files such as catalogs.

- Applies library options that are set in the metadata by an administrator.
- Supports SQL implicit pass-through.
- PROC DATASETS and PROC CONTENTS process requests using the SAS Metadata Repository instead of the underlying engine. Therefore, when you use the DATASETS procedure to list all members in a library, the engine gets a listing of only members that have metadata populated in the repository. When you execute the CONTENTS procedure, the table and column attributes that are returned are from the repository. Any formats, informats, or labels that are stored in the metadata are applied to the underlying data.
- Enables you to bypass the engine's metadata-only processing by setting alternate METAOUT= values in the LIBNAME statement or as a data set option. When certain METAOUT= values are set, the user is not restricted to tables that have been defined in the repository. However, there is restricted functionality for third-party DBMS tables that are not defined in metadata. For more information, see "METAOUT= Argument" on page 73.
- Enforces authorizations that are set in the metadata by an administrator.

Features That Are Not Supported

- The metadata engine does not create or update metadata. If you use the METAOUT= argument with a value that enables you to add, delete, or modify the structure of tables, you must use PROC METALIB or the Register Tables feature in SAS Management Console to create and update the metadata.
- The metadata engine does not support utility functionality on a third-party DBMS. For example, the PROC DATASETS RENAME and MODIFY statements are not supported on the DBMS.
- The metadata engine does not support DBMS threaded reads, even if the underlying SAS/ACCESS engine has support for this.
- The metadata engine does not support use of the LOCK statement with concatenated libraries.
- The metadata engine does not support format catalogs. As a result, specifying a metadata engine libref in the FMTSEARCH= option has no effect. If you are defining tables that rely on user-defined formats, use a different libref for the format catalogs than you use for the input table and any output tables. The libref for the format catalogs should not use the metadata engine. Otherwise, the user-defined formats will not be available.
- PROC COPY cannot be used to copy DBMS views that are described by metadata when using the metadata LIBNAME engine in its default mode. PROC COPY creates a DBMS table as output, regardless of whether the input is a table or a view. The metadata does not allow duplication because it preserves the input file's identification as a view. To copy a DBMS view, set the METAOUT=DATA option. METAOUT=DATA allows tables to be created in the library that are not described by metadata.

Advantages of Using the Metadata Engine

Using the metadata engine provides the following advantages:

- The metadata engine is a single point of access to many heterogeneous data sources. If an administrator has registered the metadata with the metadata server, a user or application can specify the appropriate metadata engine libref, and omit specifications for the underlying engine. In many cases, the user can change the data source for their SAS program by simply changing the libref. The user can ignore the syntax, options, behavior, and tuning that are required by the underlying engines, because the administrator has registered that information in the metadata.
- The metadata engine, in conjunction with the metadata server's authorization facility, enables an administrator to control access to data. The Create, Read, Write, and Delete permissions are enforced only if the metadata engine is used to access the data. See “The Metadata Engine and Authorization” on page 64.
- Some data sources do not store column formats, informats, labels, and other SAS information. This information is stored by the metadata server and is included with the data that is accessed by the metadata engine.

The Metadata Engine and Authorization

An administrator uses a product like SAS Management Console to set authorization. This security model is a metadata-based authorization layer that supplements security from the host environment and other systems. The metadata engine enforces the authorizations that are set in metadata, but it does not create or update any authorization. An advantage of this behavior is that an administrator can use the metadata engine as a means to provide library and table security.

The administrator can use authorization in the following ways for member-level and column-level security:

- The administrator can associate authorizations with any metadata resource in a repository. The metadata engine enforces effective permissions (which is a calculation of the net effect of all applicable metadata layer permission settings) for libraries and tables.
- The administrator can associate different authorizations with individual libraries and tables. For example, suppose a library has 20 tables defined in the repository. The administrator restricts access to five of the tables, because the five tables contain sensitive information. Only a few users can access all 20 tables. Most users can access only 15 tables.

The metadata engine differs from native engines in the way that it applies column permissions. You must have Write access to all of the columns specified in a request to update data with the metadata engine, even if all of the columns in the request are not being updated. For example, an Update request that includes a Read-only column in the WHERE clause will fail. Most native engines support the use of a Read-only column in the WHERE clause.

The metadata authorizations that are enforced by the metadata engine control the actions that users can perform on data that is accessed with the engine; the engine does not prevent other SAS programs from accessing the data.

Permissions That Affect Data Access through the Metadata Engine

The metadata engine enforces the following permissions in the metadata authorization layer:

Read
: specifies whether a user can read the data described by the metadata resource.

Write
: specifies whether a user can update the data described by the metadata resource.

Create
: specifies whether a user can add data to the resource described by the metadata object.

Delete
: specifies whether a user can delete data in the resource described by the metadata object.

The metadata engine enforces the Read, Write, Create, and Delete permissions on SASLibrary and PhysicalTable objects. The following table summarizes how the metadata engine enforces these permissions for these objects when a permission is denied:

Table 7.1 *Authorization Behavior of the Metadata Engine*

Metadata Object	Create Permission Behavior	Read Permission Behavior	Write Permission Behavior	Delete Permission Behavior
SASLibrary	The user cannot add tables to the library. A message is issued stating that the user is not authorized to add tables to	Behavior is not applicable for this object.	Behavior is not applicable for this object.	The user cannot delete tables from the library. A message is issued stating that the user is not authorized to delete tables

Metadata Object	Create Permission Behavior	Read Permission Behavior	Write Permission Behavior	Delete Permission Behavior
	the library. Processing terminates.			from the library. Processing terminates.
PhysicalTable	The user cannot add data to the table. A message is issued stating that the user is not authorized to add data to the table. Processing terminates.	The user cannot read data in the table. A message is issued stating that the user is not authorized to read data in the table. Processing terminates.	The user cannot update data in the table. A message is issued stating that the user is not authorized to update data in the table. Processing terminates.	The user cannot delete data from the table. A message is issued stating that the user is not authorized to delete data from the table. Processing terminates.

Security on a Column object is not enforced in the metadata engine. Therefore, the metadata engine cannot prevent a user from viewing the contents of a specific column in a table. To hide a column in a table from a user, the user's ReadMetadata permission on the column must be Deny. The ReadMetadata permission is enforced by the SAS Metadata Server.

Note: This enforcement supplements protections that are provided by the metadata server and other authorization layers. For more information, see *SAS Intelligence Platform: Security Administration Guide*.

The Metadata Engine and Extended Attributes

The metadata engine recognizes extended attributes that are defined in SAS data sets and libraries only when the METAOUT=DATA option is specified. When METAOUT=DATA is specified, the metadata engine will create, read, update, and delete extended attributes in a SAS data set. The metadata engine does not create metadata about extended attributes in the SAS Metadata Repository. However, security implemented through the metadata engine for folders and tables applies to the extended attributes.

How the Metadata Engine Constructs a LIBNAME Statement

As noted in “Overview of the Metadata LIBNAME Engine” on page 61, the metadata engine uses information from metadata to construct a LIBNAME statement for a SAS library.

When you submit a metadata LIBNAME statement, you assign a libref to a SASLibrary metadata object. The SASLibrary object is the primary object from which all other metadata is obtained. The metadata defines attributes of the data, such as table and column names. The metadata identifies the underlying engine that processes the data, and how the engine should be assigned.

8

Reference for the Metadata Engine

LIBNAME Statement for the Metadata Engine

Overview: Metadata LIBNAME Statement

To learn how the metadata engine works, see Chapter 7, “Introduction to the Metadata LIBNAME Engine,” on page 61.

The SAS Metadata Server must be running before you submit the metadata LIBNAME statement. The required SAS library metadata must already exist in the metadata server. (If you specify the “METAOUT= Argument” on page 73 with the value "DATA", table metadata is not required.) This SASLibrary metadata can be created with the New Library wizard in the SAS Management Console Data Library Manager.

In the syntax, wherever quotation marks are optional, they can be single or double quotation marks.

Syntax: Metadata LIBNAME Statement

Syntax

LIBNAME *libref*
META
LIBID=*identifier* | **LIBRARY**=*name* |
LIBURI="*URI-format*"
<*server-connection-arguments*>
<DBUSER=*user-name* DBPASSWORD=*password* >
<METAOUT=ALL | DATA | DATAREG>
;

Required Arguments

libref
specifies a SAS name that serves as a shortcut name to associate with metadata in the SAS Metadata Repository on the metadata server. This name must conform to the rules for SAS names. A libref cannot exceed eight characters.

META
is the name of the metadata engine.

LIBID=<">*identifier*<"> | LIBRARY=<">*name*<"> | LIBURI="*URI-format*"
specifies a SASLibrary object, which defines a SAS library. This SAS library contains the data that you want to process.

LIBID=<">*identifier*<">
specifies the 8- or 17-character metadata identifier of the SASLibrary object. Examples are `libid=`*AW000002* and `libid=`*"A57DQR88.AW000002"*. For more information, see Chapter 3, “Metadata Object Identifiers and URIs,” on page 13.

LIBRARY=<">*name*<">
specifies the value in the SASLibrary object's Name= attribute. An example is `library=`*mylib*. The maximum length is 256 characters.

Alias LIBRNAME=

LIBURI="*URI-format*"
specifies a URI, which is a standard from SAS Open Metadata Architecture. For more information, see Chapter 3, “Metadata Object Identifiers and URIs,” on page 13. The following URI formats are supported.

LIBURI="*identifier.identifier*"
specifies the full 17–character metadata identifier, which references both the repository and the object. This syntax is equivalent to specifying both LIBID= and REPID=. An example is `liburi="A57DQR88.AW000002"`.

LIBURI="SASLibrary/*identifier.identifier*"
specifies the SASLibrary object type, followed by the full 17-character metadata identifier. This syntax is equivalent to specifying both LIBID= and REPID=. An example is `liburi="SASLibrary/A57DQR88.AW000002"`.

LIBURI="SASLibrary?@*attribute*='*value*'"
specifies the SASLibrary object type, followed by a search string. Examples are `liburi="SASLibrary?@libref='mylib'"` and `liburi="SASLibrary?@engine='base'"`.

Requirement You must enclose the LIBURI= value in quotation marks.

Server Connection Arguments

The following LIBNAME statement arguments for the metadata engine establish a connection to the metadata server. For more information, see "Introduction to Connection Options" on page 38.

METASERVER=<">*host-name*<">
specifies the host name or network IP address of the computer that hosts the metadata server. The value `localhost` can be used if the SAS session is connecting to the metadata server on the same computer. If you do not specify this argument, the value of the METASERVER= system option is used. For more information, see "METASERVER= System Option" on page 54. The maximum length is 256 characters.

Alias HOST= or IPADDR=

PASSWORD=<">*password*<">
specifies the password for the user ID on the metadata server. If you do not specify this argument, the value of the METAPASS= system option is used. For more information, see "METAPASS= System Option" on page 49. The maximum length is 256 characters.

Alias METAPASS= or PW=

PORT=<">*number*<">
specifies the TCP port that the metadata server listens to for connections. This port number was used to start the metadata server. If you do not specify this argument, the value of the METAPORT= system option is used. For more information, see "METAPORT= System Option" on page 50. The range of allowed port numbers is 1–65535. The metadata server is configured with a default port number of 8561.

Alias METAPORT=

REPID=<">*identifier*<"> | REPNAME=<">*name*<">
specifies the repository that contains the SASLibrary object. If you specify both REPID= and REPNAME=, REPID= takes precedence over REPNAME=. If you do not specify REPID= or REPNAME=, the value of the METAREPOSITORY= system option is used; for more information, see "METAREPOSITORY= System Option" on page 53 . The default for the METAREPOSITORY= system option is `Foundation`.

REPID=<">*identifier*<">
specifies an 8-character identifier. This identifier is the first half of the SASLibrary's 17–character identifier, and is the second half of the repository's identifier. For more information, see Chapter 3, “Metadata Object Identifiers and URIs,” on page 13.

REPNAME=<">*name*<">
specifies the value in the repository's Name= attribute. The maximum length is 256 characters.

Alias METAREPOSITORY= or REPOS= or REPOSITORY=

USER=<">*userid*<">
specifies the user ID for an account that is known to the metadata server. For information about user definitions, see the *SAS Intelligence Platform: Security Administration Guide*. If you do not specify this argument, the value of the METAUSER= system option is used; see “METAUSER= System Option” on page 57 . The maximum length is 256 characters.

Alias ID= or METAUSER= or USERID=

Optional Database Connection Arguments

The DBUSER= and DBPASSWORD= arguments enable users to provide credentials programmatically for libraries whose server definitions have an authentication type of Prompt defined. These credentials override other predefined authentication types, including credentials that are stored in AuthenticationDomains. The credentials must be valid on the target database. These arguments are valid beginning with SAS 9.4M3.

DBUSER=<">*user-name*<">
specifies the name of a database user account that can access the library on the database.

Note A user name value is changed to uppercase text if it is not enclosed in quotation marks. If a value is case sensitive or contains special characters, it has to be enclosed in either single or double quotation marks.

DBPASSWORD=<">*password*<">
specifies the password associated with the specified database user account.

Alias DBPASS=, DBPASSWD=

Note A password value is changed to uppercase text if it is not enclosed in quotation marks. If a value is case sensitive or contains special characters, it has to be enclosed in either single or double quotation marks.

Tip You can encode the password with PROC PWENCODE. For more information about the PWENCODE procedure, see *Base SAS Procedures Guide*.

METAOUT= Argument

METAOUT=ALL | DATA | DATAREG
specifies the metadata engine's output processing of tables in the data source.

ALL
specifies that you can read, create, update, and delete observations in an existing physical table that is defined in metadata. You cannot create or delete a physical table, or alter a physical table's columns. This is the default behavior.

Interaction The user is restricted to only the tables and columns that have been defined in the repository.

DATA
specifies that you can read, create, alter, update, and delete physical tables. The user can access any table, regardless of whether it has been defined in the repository.

Restrictions You must have sufficient operating system or database privileges in order to access the unregistered physical tables.

The metadata LIBNAME engine does not support variable projection for tables in a third-party DBMS that are not defined in metadata. The engine retrieves all columns for undefined tables, regardless of the operation. If you need to select, insert data into, update, drop, or keep specific columns, register your DBMS tables in the SAS Metadata Repository. Variable projection is supported in Base SAS data sets that are not defined in metadata.

Interaction The user is not restricted to tables that have been defined in the repository, although there is some restricted functionality for tables that are not defined in metadata.

DATAREG
specifies that you can read, update, alter, and delete a physical table that is defined in metadata. You can also create and delete physical tables that are not defined in metadata, but you cannot read or update the new physical tables until they are defined in metadata. This value is like the value for DATA, except tables and columns that are not defined in the repository are not visible.

Default ALL

Restriction The following descriptions refer to the physical table. Metadata is read-only with the metadata engine. When you create, update, or delete physical data with the metadata engine, you must perform an additional step if you want to update the metadata. You must use a product like SAS Management Console or the METALIB procedure to update the metadata.

Interactions As a LIBNAME statement argument, the behavior applies to all members in the library, and remains for the duration of the library

assignment. To specify METAOUT= behavior for an individual table, use the METAOUT= data set option.

If metadata for a table is defined, any authorizations for that table are enforced, regardless of the METAOUT= value.

SAS Data Set Options for the Metadata Engine

METAOUT= Data Set Option

Overview

The METAOUT= data set option for the metadata engine specifies access to an individual table in the data source.

Note: You can use the METAOUT= argument for the LIBNAME statement to specify behavior for an entire library. When the option is specified for a library, the behavior applies to all members in the library, and remains for the duration of the library assignment.

Note: For library procedures such as PROC DATASETS, you must specify METAOUT= as an argument in the LIBNAME statement. You cannot specify it as a data set option.

Syntax

METAOUT=ALL | DATA | DATAREG
specifies the metadata engine's output processing of tables in the data source.

ALL
specifies that you can read, create, update, and delete observations in an existing physical table that is defined in metadata. You cannot create or delete a physical table, or alter a physical table's columns. This is the default behavior.

Interaction The user is restricted to only the tables and columns that have been defined in the repository.

DATA

specifies that you can read, create, alter, update, and delete a physical table. The user can access any table, regardless of whether it has been defined in the repository.

Restrictions You must have sufficient operating system or database privileges in order to access the unregistered physical tables.

The metadata LIBNAME engine does not support variable projection for tables in a third-party DBMS that are not defined in metadata. The engine retrieves all columns for undefined tables, regardless of the operation. If you need to select, insert data into, update, drop, or keep specific columns, register your DBMS tables in the SAS Metadata Repository. Variable projection is supported in Base SAS data sets that are not defined in metadata.

Interaction The user is not restricted to tables that are defined in metadata, although there is some restricted functionality for tables that are not defined in metadata.

DATAREG

specifies that you can read, update, alter, and delete a physical table that is defined in metadata. You can also create and delete physical tables that are not defined in metadata, but you cannot read or update the new physical tables until they are defined in metadata. This value is like the value for DATA, except tables and columns that are not defined in the repository are not visible.

Default ALL

Restriction The preceding descriptions refer to the physical table. Metadata is read-only with the metadata engine. When you create, update, or delete physical data with the metadata engine, you must perform an additional step if you want to update the metadata. You must use a product like SAS Management Console or the METALIB procedure to update the metadata.

Interaction If metadata for a table is defined, any authorizations are enforced for that table, regardless of the METAOUT= value.

9

Examples for the Metadata Engine

Example: Submitting the LIBNAME Statement

This example shows three metadata LIBNAME statements. One statement uses defaults, one specifies metadata server connection parameters in the LIBNAME statement, and the third specifies database connection parameters in the LIBNAME statement.

- The following LIBNAME statement uses defaults. The connection information for the SAS Metadata Server is obtained from the metadata system options. Data source connection information is obtained from metadata.

  ```
  libname metaeng meta library=mylib;
  ```

- This example specifies the LIBNAME statement options for the metadata engine to connect to the metadata server. Data source connection information is obtained from metadata.

  ```
  libname myeng meta library=mylib
     repname=temp metaserver='a123.us.company.com' port=8561
     user=idxyz pw=abcdefg;
  ```

- This example specifies the LIBNAME statement options for the metadata engine to connect to the target database. The connection information for the SAS

Metadata Server is obtained from the metadata system options. The credentials specified in the DBUSER= and DBPASSWORD= arguments override data source connection information that is stored in metadata.

```
libname oralib meta library=oralib dbuser=orauser dbpassword=orapw;
```

Example: Before and After the Metadata Engine

Overview

This example shows how data can be accessed with the SAS/ACCESS Interface to Oracle and then, for comparison, shows how the same data can be accessed with the metadata engine. The code accesses Oracle data, lists the tables that exist in the data source, and prints the contents of one table.

Using the SAS/ACCESS Interface to Oracle Engine Directly

To use the SAS/ACCESS Interface to Oracle engine directly to access the data, you submit statements like the following, which require that you know how to use the Oracle engine and that you know the appropriate options to access the data:

```
libname oralib oracle user=myuser pw=mypw
   path=ora_dbms preserve_tab_names=yes
   connection=sharedread schema=myschema; 1

proc datasets library=oralib; 2
quit;

proc print data=oralib.Sales (readbuff=1000); 3
run;

data work.temp;
   set oralib.Sales (dbindex=myindex); 4
run;
```

1 Identifies an Oracle library that contains the Oracle tables that you want to process.

2 Lists all of the Oracle tables that are available.

3 Displays the Oracle Sales table.

4 Attempts to use the specified index to improve performance.

Using the Metadata Engine

You can access the same data using the metadata engine. However, when using the metadata engine, you do not have to know how to use the Oracle engine, or know the appropriate options to access the data. You do not need to be aware that you are using an Oracle database.

Using SAS Management Console or SAS Data Integration Studio, an administrator creates metadata in a SAS Metadata Repository for your Oracle environment. The metadata engine interprets this metadata and locates your data. You do not have to know how to connect to the metadata server or the repository, because this information can be provided by the metadata system options.

Here is what happens when you use the metadata engine to access the Oracle data:

1. You submit the following LIBNAME statement for the metadata engine. LIBRARY= identifies the SASLibrary object that defines information about the Oracle library. This SASLibrary object serves as an anchor point for obtaining other metadata.

   ```
   libname metaeng meta library=mylib;
   ```

   The metadata server connection properties are specified by metadata system options, so they are omitted from the LIBNAME statement.

2. The metadata engine queries the repository. The query retrieves information from the SASLibrary object that is specified by LIBRARY=. Connection and schema information are returned by the query.

3. From the information returned by the metadata query, the metadata engine is able to generate the following LIBNAME statement, which is the same LIBNAME statement that is shown at the beginning of this example:

   ```
   libname oralib oracle user=myuser pw=mypw
      path=ora_dbms preserve_tab_names=yes
      connection=sharedread schema=myschema;
   ```

4. With the generated LIBNAME statement, the metadata engine uses the Oracle engine anytime it needs to access the Oracle data. For example, to view the tables that exist, you would submit the following:

   ```
   proc datasets library=metaeng;
   quit;
   ```

   The metadata engine sends a query to the repository. The query requests all members of the SASLibrary that was specified by LIBRARY=. The metadata engine returns only those members that are defined in the repository. Any Oracle table that is not defined in the metadata is not displayed. (If METAOUT=DATA, all tables are displayed, regardless of whether they are defined in metadata.)

5. For the following PRINT procedure, the metadata engine sends a request to the repository for the metadata that is associated with the Sales table.

   ```
   proc print data=metaeng.Sales;
   run;
   ```

The metadata engine returns the columns that are defined in the metadata. Therefore, if the Sales table has 20 columns, and only five columns are defined in the metadata, then you see only five columns. (If METAOUT=DATA, all columns are displayed, regardless of whether they are defined in the metadata.)

6 A SASLibrary metadata object also stores index information for tables. Any use of the metadata engine that uses indexes causes a query to the repository that requests index information. The index metadata must match the physical index on the table. The metadata engine uses the index information that is stored in the repository:

```
data work.temp;
   set metaeng.Sales;
run;
```

PART 4

Procedures

10

Introduction to Procedures for Metadata

Overview of Procedures for Metadata

As with the other metadata language elements, you can use the metadata procedures in a batch SAS program or in the SAS windowing environment. You can also perform these tasks with a product like SAS Management Console.

The procedures enable you to create and maintain the metadata in a SAS Metadata Repository.

- The METADATA procedure sends a method call, in the form of an XML string, to the SAS Metadata Server.
- The METALIB procedure updates metadata to match the tables in a library.
- The METAOPERATE procedure performs administrative tasks on the metadata server.

To submit the procedures, you must establish a connection with the metadata server. You can specify connection information in the procedure, in system options, or in a dialog box. For more information, see “Connection Options ” on page 38.

Comparison of the METADATA Procedure and the METAOPERATE Procedure

The METADATA procedure can be used to perform some of the same informational tasks as the METAOPERATE procedure. The benefit of using PROC METAOPERATE is simpler syntax. The benefit of using PROC METADATA is a broader range of tasks. (PROC METADATA supports all of the parameters of the methods that it submits. Some of these parameters are not supported by PROC METAOPERATE.) In addition, PROC METADATA creates XML output that you can use in another program (for example, to run reports).

Here is an example that uses PROC METAOPERATE to check whether the SAS Metadata Server is paused or running:

```
proc metaoperate
   action=status;
run;
```

The SAS Metadata Server returns the following information to the SAS log:

```
NOTE: Server a123.us. company.com SAS Version is 9.4.
NOTE: Server a123.us. company.com SAS Long Version is 9.04.01M0P11062012.
NOTE: Server a123.us. company.com Operating System is XP_PRO.
NOTE: Server a123.us. company.com Operating System Family is DNTHOST.
NOTE: Server a123.us. company.com Operating System Version is Service Pack 3.
NOTE: Server a123.us. company.com Client is janedoe.
NOTE: Server a123.us. company.com Metadata Model is Version 15.01.
NOTE: Server a123.us. company.com is RUNNING on 14Nov2012:11:28:57.
```

PROC METADATA can perform a similar check in two ways. You can use the following code:

```
proc metadata
in=' <Status>
<Metadata>
</Metadata>
</Status>';
run;
```

Or, you can submit this code (note the blank space in the IN= argument):

```
proc metadata
    method=status
    in=' ';
run;
```

The SAS Metadata Server returns the following information to the SAS log in the form of XML. The status parameters differ slightly from those returned by PROC METAOPERATE.

```
<ModelVersion>15.01</ModelVersion><PlatformVersion>9.4.0.0</PlatformVersion>
```

```
<ServerState>ONLINE</ServerState><PauseComment/><ServerLocale>en_US</ServerLocale>
```

PROC METADATA supports two interfaces for submitting requests: the SAS Open Metadata Interface DoRequest method, which accepts XML-formatted method calls as input, and the SAS Open Metadata Interface Status method, which supports only XML elements that are supported by the Status method: The first PROC METADATA example is an example of a Status method call that is submitted through the DoRequest method. There is an implied METHOD=DOREQUEST argument in the request. The second PROC METADATA example shows how METHOD=STATUS is used to get default status information. The Status method is recommended for issuing server status queries, because it is simpler and because the DoRequest method is not available when the SAS Metadata Server is paused.

Consider this request, issued through default DoRequest interface:

```
proc metadata
in=' <Status>
      <Metadata>
       <OMA JOURNALSTATE=""/>
      </Metadata>
     </Status> ';
run;
```

The code returns the journal state as follows:

```
<Status><Metadata><OMA JOURNALSTATE="IDLE"/></Metadata></Status>
```

Here is the same example using METHOD=STATUS. (To get information with METHOD=STATUS, you submit the parameter directly in the IN= argument.)

```
proc metadata
    method=status
    in='<OMA JOURNALSTATE=""/>';
run;
```

This code returns the journal state:

```
<OMA JOURNALSTATE="IDLE"/>
```

If you have a simple query that is not supported by PROC METAOPERATE, and you do not want to assign an XML LIBNAME engine to parse the output of PROC METADATA, you can use the metadata DATA step functions.SAS provides the “METADATA_PAUSED Function” on page 244 to determine whether the SAS Metadata Server is paused. SAS provides the “METADATA_VERSION Function” on page 252 to get the model version number.

Chapter 11

METADATA Procedure

Overview: METADATA Procedure

What Does the METADATA Procedure Do?

The METADATA procedure sends an XML string to the SAS Metadata Server. Depending on the value in the METHOD= argument, DOREQUEST or STATUS, the IN= argument can contain a SAS Open Metadata Interface method call for reading or writing metadata. Or, it can contain an XML element that is supported in the SAS Open Metadata Interface IServer Status method. The IServer Status method supports options for monitoring the metadata server and its configuration.

In DOREQUEST mode (the default when the METHOD= argument is omitted), the procedure submits requests to the metadata server via the SAS Open Metadata Interface's IOMI DoRequest method, which is a messaging interface. This messaging interface accepts all methods from the SAS Open Metadata Interface's IOMI server interface, which consists of methods for reading and writing metadata. It also accepts the IServer Status method. However, this messaging interface is available only when the metadata server is in an online state (that is, the metadata server is not paused or in some transitory state). For more information about metadata server states, see *SAS Intelligence Platform: System Administration Guide*.

When METHOD=STATUS is specified, PROC METADATA submits IServer Status method requests directly to the metadata server. METHOD=STATUS enables you to get information about the metadata server when the metadata server is online, paused, and in a transitory state.

There are specific requirements for submitting requests through METHOD=DOREQUEST versus METHOD=STATUS.

When METHOD=STATUS is specified, PROC METADATA can get status information about a single SAS Metadata Server or a cluster of SAS Metadata Server nodes. The procedure can also be used to get information about metadata server backups, the metadata server's alert email notification system, and the metadata server's alert email reminder system.

The METAOPERATE procedure and the metadata DATA step functions can perform some of the same tasks as the METADATA procedure. For more information, see "Comparison of the METADATA Procedure and the METAOPERATE Procedure" on page 84.

Syntax: METADATA Procedure

Restriction: This procedure is not supported in SAS Viya.

Requirement: The metadata server must be running.

Notes: The OPTIONS= argument applies to SAS 9.4M3 and to later releases.

Be careful when you modify metadata objects because many objects have dependencies on other objects. For more information, see Chapter 2, “Using Language Elements That Read and Write Metadata,” on page 7. A product like SAS Management Console or SAS Data Integration Studio is recommended for the routine maintenance of metadata. Before you use PROC METADATA to create or modify metadata, perform a server backup.

See: “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 16

PROC METADATA <*server-connection-arguments*>
<METHOD=DOREQUEST | STATUS>
IN="*XML-string*" | *fileref*
<OUT=*fileref*>
<HEADER=NONE | SIMPLE | FULL>
<NOREDIRECT>
<OPTIONS="*XML-element*">
<VERBOSE>;

Statement	Task
PROC METADATA	Sends an XML string to the metadata server

PROC METADATA Statement

Sends an XML string to the SAS Metadata Server.

Syntax

PROC METADATA <METHOD=DOREQUEST> IN="*XML-method-call*" | *fileref* <*options*>;

PROC METADATA METHOD=STATUS IN="*XML-element*" | *fileref* <*options*>;

Summary of Optional Arguments

HEADER= NONE | SIMPLE | FULL
METHOD=DOREQUEST | STATUS
NOREDIRECT
OPTIONS= "*XML-element*"
OUT=*fileref*
VERBOSE

Required Argument

IN="*XML-string*" | *fileref*
specifies an input XML string or fileref. The type of XML string that is submitted depends on whether the METHOD= argument is specified and what its value is.

- If the METHOD= argument is omitted or if it specifies METHOD=DOREQUEST, IN= specifies an XML-formatted method call or IN= specifies an XML file that contains the method call.

  You form the method call as if you are submitting it in the inMetadata parameter of the DoRequest method. You can submit any method from the SAS Open Metadata Interface IOMI server interface and the IServer Status method. For more information, see "Formatting an XML Method Call for DoRequest" on page 93.

- If METHOD=STATUS, IN= specifies an XML element that is valid in the inMeta parameter of the IServer Status method or IN= specifies an XML file that contains the XML element. For more information, see "Submitting an XML Element with METHOD=STATUS" on page 95.

For an example of how a fileref is specified in the IN= argument, see "Example 4: Request the Metadata for One Type of Object" on page 109.

Note PROC METADATA does not support fixed-length records in the XML method call under z/OS. PROC METADATA returns an error on files with fixed-length records whether a fileref or XML string is used.

Optional Arguments

HEADER= NONE | SIMPLE | FULL
specifies whether to include an XML header in the output XML file. The declaration specifies the character-set encoding for web browsers and XML parsers to use when processing national language characters in the output XML file. For more information, see "Example 2: Add an Encoding to The Output XML File" on page 104.

NONE
omits an encoding declaration. Web browsers and parsers might not handle national language characters appropriately.

SIMPLE
inserts an XML header that specifies the XML version number: <?xml version="1.0"?>. This is the default value when the HEADER= argument is not specified.

FULL
inserts an XML declaration that represents the encoding that was specified when creating the output XML file. The source for the encoding varies, depending on the operating environment. In general, the encoding value is taken from the ENCODING= option specified in the FILENAME statement, or from the ENCODING= system option.

SAS attempts to use that encoding for the output XML file (and in the XML header). The encoding can vary. A single encoding can have multiple names or aliases that can appear in the XML header. These names might not be valid or recognized in all XML parsers. When generating the encoding attribute in the XML header, SAS attempts to use an alias that will be recognized by Internet Explorer. If the alias is not found, SAS attempts to use

a name that will be recognized by Java XML parsers. If the name is not found, SAS uses an alias by which SAS will recognize the encoding.

For information about encoding and transcoding, see *SAS National Language Support (NLS): Reference Guide*.

METHOD=DOREQUEST | STATUS

METHOD= is an optional argument that specifies whether PROC METADATA is submitting a metadata query or a server status or configuration query. See the IN= argument on page 90 for information about the requirements for each method. Use of METHOD=STATUS is recommended for getting server status and configuration information because it can connect to a metadata server that is in an online, paused, or transitory state. If the METHOD= argument is omitted, the default mode of operation is DOREQUEST.

NOREDIRECT

NOREDIRECT is an optional argument that is used with METHOD=STATUS in a clustered SAS Metadata Server configuration. In a single SAS Metadata Server configuration, NOREDIRECT is ignored. NOREDIRECT temporarily overrides the cluster load balancer so that a request can be executed on the server node specified in the connection options only. Use NOREDIRECT with METHOD=STATUS when you want to get status information about a specific server node. For more information, see “Metadata Server Configurations and PROC METADATA” on page 96.

OPTIONS= "*XML-element*"

is an optional argument that is supported with METHOD=STATUS only. Currently, the OPTIONS= argument accepts one XML element: <CLUSTER/>.

<CLUSTER/>

specifies to send the query in the IN= argument to the master node. Use OPTIONS="<CLUSTER/>" when you want to list information about all nodes in the cluster. This argument enables you to query the master node without knowing its connection parameters. For more information, see “Metadata Server Configurations and PROC METADATA” on page 96.

OUT=*fileref*

specifies an XML file in which to store the output that is returned by the metadata server. The value must be a fileref, not a pathname. Therefore, you must first submit a FILENAME statement to assign a fileref to a pathname. In most cases, the output XML string is identical to the input XML string, with the addition of the requested values within the XML elements. If the OUT= argument is omitted, PROC METADATA output is written to the SAS log. For more information, see Results: METADATA Procedure on page 100. See also: “GetRepositories Request That Directs Output to an XML File” on page 102.

Notes PROC METADATA can generate large XML output. You might need to specify a large LRECL value or RECFM=N (streaming output) to avoid truncation of long output lines.

Under z/OS, fixed-length records in the XML method call are not supported by PROC METADATA. Specify RECFM=V (or RECFM=N as suggested above) when you create the XML method call.

VERBOSE

specifies to print the input XML string to the SAS log after it has been preprocessed.

Server Connection Arguments

Server connection arguments establish communication with the metadata server. If you omit these arguments, then the values of the metadata system options are used or the values can be obtained interactively. Note that server connections made with server connection arguments are not redirected to another server in the cluster when the default server connection profile is used. If the server specified in the server connection arguments is not available, then the connection will fail. If connection redirection is important for the request, use metadata system options to establish communication with the server instead. For more information, see “Connection Options ” on page 38.

PASSWORD="*password*"
is the password for the authenticated user ID on the metadata server. If you do not specify PASSWORD=, the value of the METAPASS= system option is used. For more information, see “METAPASS= System Option” on page 49. The maximum length is 512 characters.

Alias METAPASS= or PW=

PORT=*number*
is the TCP port that the metadata server listens to for requests. This port number was used to start the metadata server. If you do not specify PORT=, the value of the METAPORT= system option is used. For more information, see “METAPORT= System Option” on page 50. The range of allowed port numbers is 1–65535. The metadata server is configured with a default port number of 8561.

Alias METAPORT=

Requirement Do not enclose the value in quotation marks.

PROTOCOL=BRIDGE
specifies the network protocol for connecting to the metadata server. If you do not specify PROTOCOL=, the value of the METAPROTOCOL= system option is used. For more information, see “METAPROTOCOL= System Option” on page 52. In this release, the only supported value is BRIDGE, which specifies the SAS Bridge protocol. This is the server default, so there is no need to specify this argument.

Alias METAPROTOCOL=

Requirement Do not enclose the value in quotation marks.

REPOSITORY= "*name*"
is the name of the SAS Metadata Repository to use when resolving the $METAREPOSITORY substitution variable. PROC METADATA enables you to specify the substitution variable $METAREPOSITORY in your input XML. The substitution variable is resolved to the repository that you specify in REPOSITORY=. This value is the repository's Name= attribute. If you do not specify REPOSITORY=, the value of the METAREPOSITORY= system option is used. For more information, see “METAREPOSITORY= System Option” on page 53. The default for the METAREPOSITORY= system option is FOUNDATION. The maximum length is 32,000 characters.

Alias METAREPOSITORY= or REPOS=

SERVER="*host-name*"
is the host name or network IP address of the computer that hosts the metadata server. The value LOCALHOST can be used if the SAS session is connecting to the metadata server on the same computer. If you do not specify SERVER=, the value of the METASERVER= system option is used. For more information, see "METASERVER= System Option" on page 54. The maximum length is 256 characters.

Alias HOST= or IPADDR= or METASERVER=

USER="*authenticated-user-ID*"
is an authenticated user ID on the metadata server. The metadata server supports several authentication providers. For more information about authentication, see the *SAS Intelligence Platform: Security Administration Guide*. If you do not specify USER=, the value of the METAUSER= system option is used. For more information, see "METAUSER= System Option" on page 57. The maximum length is 256 characters.

Alias ID= or METAUSER= or USERID=

Usage: METADATA Procedure

Formatting an XML Method Call for DoRequest

Overview

The IN= argument of PROC METADATA submits one or more XML-formatted method calls to the SAS Metadata Server. You can submit any method that is supported by the DoRequest method of the SAS Open Metadata Interface, including:

- all methods in the IOMI server interface
- the IServer Status method

When multiple method calls are sent in one DoRequest submission, they must be enclosed within <Multiple_Requests></Multiple_Requests> elements.

For information about how to format a method call for DoRequest, see the documentation for the DoRequest method in *SAS Open Metadata Interface: Reference and Usage*. The IOMI server interface section of the book shows how to format each IOMI method for use in the DoRequest interface. The IOMI methods are documented with many usage examples.

PROC METADATA is among several clients that can submit the DoRequest method to the SAS Metadata Server. You are strongly advised to read *SAS Open Metadata*

Interface: Reference and Usage for help in understanding concepts such as flags, filters, and templates. The following topics provide a brief introduction to submitting method calls through the DoRequest interface.

The Entire Method Is an XML Element

With PROC METADATA, you submit a request as an XML string. In the request, a method is represented as an XML element. In the following example, the method is GetMetadataObjects. The request starts and ends with GetMetadataObjects tags. Do not include the DoRequest method in the XML string because the procedure calls DoRequest for you.

```
proc metadata
   in='<GetMetadataObjects>
    <Reposid>A0000001.A5UO0N94</Reposid>
         <Type>SASLibrary</Type>
         <Objects/>
         <NS>SAS</NS>
         <Flags>0</Flags>
         <Options/>
       </GetMetadataObjects>';
run;
```

The GetMetadataObjects method has parameters Reposid, Type, Objects, NS (namespace), Flags, and Options. The method parameters are submitted as XML subelements in the input XML method string.

A Metadata Object Is an XML Element

Some methods accept metadata objects as input. Within your XML string, metadata objects are represented as XML elements. Object attributes, if any, are XML tag attributes. In the following code, a PhysicalTable object has "NE Sales" in its Name= attribute:

```
<PhysicalTable Id="A5UO0N94.B20000TV" Name="NE Sales"/>
```

A Metadata Association Is an XML Element

Metadata associations are XML elements, which are nested within the primary object's XML element. In the following code, the PhysicalTable object has a Columns association to a Column object that has "Sales Associates" in its Name= attribute:

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

The Name= attribute in the Column XML element defines or identifies a particular Column metadata object.

Empty XML elements (that is, XML elements with no content between the start and end tags) can be expressed in XML shorthand as a singleton tag, like this: `<Columns/>`. In a GetMetadata request, an empty association name subelement instructs the metadata server to return all objects associated with the primary object under that association name.

Quotation Requirements

Single or double quotation marks can be used to submit the IN= XML method string. To ensure that the string is parsed correctly, it is recommended that any additional quotations within the string, such as those enclosing XML attribute values, be balanced. For example, if you submit the IN= string within single quotation marks, use double quotation marks for attribute values. If you use double quotation marks to submit the IN= string, use single quotation marks for attribute values.

When additional nesting of quotations is necessary, such as in a GetMetadataObjects <XMLSELECT search="*string*"/> element, use double apostrophes or double quotation marks as follows:

```
<XMLSelect search="*[@PublicType=''InformationMap.Relational'']"/>

<XMLSelect search=""*[@PublicType= 'InformationMap.Relational']""/>
```

See Also

Forming proper XML input can be a challenge. Use the following resources:

- See "Example: Creating a Report with the METADATA Procedure and the XML Engine" on page 16
- *SAS Open Metadata Interface: Reference and Usage* provides the following information:
  - which methods to use for common tasks
  - the DoRequest method and other methods in the IOMI server interface
  - the Status method in the IServer server interface
- The SAS Metadata Model: Reference shows the relationships among objects, associations, and attributes that you specify in XML tags.

Submitting an XML Element with METHOD=STATUS

When METHOD=STATUS, there is no need to specify all of the Status method's parameters in the IN= XML string. PROC METADATA accepts as input XML elements that are valid in the Status method's inMeta parameter. In the IN=

argument, specify only the XML elements for which you want to get values. For example:

```
proc metadata
   method=status
   in='<ServerState/>
   <PauseComment/>
   <OMA JournalState=""/>
   <OMA JournalHistoricalData=""/>
   <OMA SERVERSTARTPATH=" "/>';
run;
```

This example submits five XML elements that might be useful when the server is unavailable.

You can also use METHOD=STATUS to get information about server backups and recoveries, the server backup configuration, the metadata server's alert email notification subsystem, and clustered metadata server configurations. See the Examples section.

For a complete list of the XML elements that are available in the Status method, see the Status method documentation in *SAS Open Metadata Interface: Reference and Usage*.

Metadata Server Configurations and PROC METADATA

Beginning in SAS 9.4, the SAS Intelligence Platform supports single SAS Metadata Server configurations and clustered SAS Metadata Server configurations.

- In a single metadata server configuration, all metadata requests are received and processed by a single SAS Metadata Server.
- In a clustered metadata server configuration, three or more identical metadata servers are linked in such a way that provides metadata redundancy. The servers have a master-slave relationship. The slave nodes process read requests and forward update requests to the master node. The master node processes the updates and then propagates them to the slave nodes. If one node becomes unavailable, its load is transferred to another node. A load-balancing algorithm controls server connections.

The clustered metadata server configuration has no effect on PROC METADATA METHOD=DOREQUEST requests, except to improve metadata availability. All server nodes have identical metadata; the cluster simply increases the number of nodes that are available to process requests.

The load balancer can affect the information returned by a METHOD=STATUS request. In a clustered server configuration, the load balancer directs all queries to a slave node of the load balancer's choice, regardless of which server is specified in the connection options. Most of the time, the default connection is sufficient to answer questions that administrators have about the metadata server. In a clustered server configuration, all server nodes have identical configurations and are managed uniformly. For example, the default connection is sufficient for getting information about metadata system options, omaconfig.xml server configuration options, and for getting information about metadata server backups.

However, a slave node knows only about itself and the master node. The master node is the only node that has information about all of the nodes in the cluster. In order to get complete results for METHOD=STATUS queries that request a list or a count of the server nodes that are available (<CLUSTER CURRENT_NODES=" "/> and <CLUSTER LIST=" "/>), you must direct the requests to the master node. PROC METADATA supports two arguments to enable you to control where Status requests are sent in a clustered server configuration:

- The OPTIONS="<CLUSTER/>" argument was added to PROC METADATA in SAS 9.4M3 to enable you to send requests directly to the master node. You do not need to know connection parameters for the master node when you use this argument.
- The NOREDIRECT argument specifies to send the request to the server node indicated in the server connection options. In previous releases, the NOREDIRECT argument was used to send a request to the master node. Now, this argument can be used to send a request to a specific server node, regardless of whether it is a master or slave.

You might want to send a status request to a specific slave node for one of the following reasons:

- to get information about the server node's start path
- to check for alert conditions that could result in the automatic termination of the server node.

For more information about cluster XML elements, see "Getting Information about a SAS Metadata Server Cluster" on page 97.

Getting Information about a SAS Metadata Server Cluster

The following XML elements can be submitted through PROC METADATA with METHOD=STATUS to get information about the cluster:

<CLUSTER *attributes*/>
returns values for specified cluster attributes. The valid attributes are:

CLUSTERGUID=" "
returns the cluster's unique identifier. This value is the same for all nodes in the cluster.

DEFINED_NODES=" "
returns the number of servers defined in the cluster.

CURRENT_NODES=" "
returns the number of servers that are known to the server that received the query. This attribute returns different results, depending on whether it is processed by a slave node or by the master node.

HAS_FIRST_NODE=" "
returns a YES or NO indicating whether the server defined as Node 1 is available to the cluster. This value is the same for all nodes in the cluster.

HAS_QUORUM=" "
returns a YES or NO indicating whether a quorum exists.

LIST=" "
returns an integer indicating the number of servers that are known to the server that received the query, in addition to <CLUSTERNODE/> XML elements that describe each server. The <CLUSTERNODE/> XML elements include the server name, host name, port number, a Self attribute, and a Flags attribute for each server. The Self attribute identifies the receiving server with a “Y” or a “N”. The Flags attribute indicates whether the node is a slave node or the master node. The LIST= attribute returns information about two nodes if it is processed by a slave node. To get a complete listing of available servers, a request that contains this attribute should be directed to the master node.

<CLUSTERSTATE/>
returns the value STARTING, QUORUM, or LOSTQUORUM. STARTING means that the cluster is waiting for more server nodes to start up and complete the quorum. QUORUM means that a sufficient number of server nodes are operating for the cluster to service metadata requests. LOSTQUORUM means that the cluster does not have enough server nodes to service metadata requests.

<OMA MAXIMUM_CLUSTER_NODES=" "/>
returns the maximum number of server nodes that are supported in the cluster as configured in the omaconfig.xml file.

An administrator might use the cluster XML elements to answer the following questions:

- Fewer server nodes can be defined at cluster configuration than are supported. Administrators can add server nodes later up to the number specified in the <OMA MAXIMUM_CLUSTER_NODES=" "/> omaconfig.xml option plus one. The additional slot is for the master. These additional server nodes must be added using the SAS Deployment Wizard. Submitting this XML element in PROC METADATA with METHOD=STATUS returns the value of this omaconfig.xml option. If you have a need to add servers beyond this number, contact SAS Technical Support.
- An administrator can track the number of server nodes that are defined in the SAS Deployment Wizard and removed with the SAS Deployment Manager by submitting the <CLUSTER DEFINED_NODES=" " element.
- The administrator can submit the <CLUSTER LIST=" "/> element to the master node to get an accurate count of the currently available server nodes and their connection parameters.
- The quorum requires that at least half of the servers that are defined in the cluster be available to continue operating. If exactly half of the servers are available, then the server defined as Node 1 must be among those that are operating. The <CLUSTERSTATE/> and <CLUSTER DEFINED_NODES=" " CURRENT_NODES=" " HAS_FIRST_NODE=" " HAS_QUORUM=" "/> elements can be submitted to monitor the cluster’s status in relation to the quorum.

For an example of how the cluster-related XML elements and arguments are submitted, see “Example 7: Get Information about the Server Cluster with PROC METADATA” on page 118.

Getting Information about Server Backups

PROC METADATA can get information about server backups. For information about the XML elements that are submitted to get information about backups, see the IServer Status method in *SAS Open Metadata Interface: Reference and Usage*. For information about how the XML elements are submitted from PROC METADATA, see "Example 5: Get Server Backup Information with PROC METADATA" on page 111.

The XML elements that get information about backups behave the same in a clustered metadata server configuration as they do in a single metadata server configuration with one exception. An additional step is necessary in a clustered server configuration when you want to return the list of files that were included in the last backup.

The files that were included in a backup are listed by submitting the <METADATASERVERBACKUPMANIFEST/> XML element in the IN= argument with METHOD=STATUS. By default, this XML element reads the backup manifest file from the SAS Metadata Server's configuration directory. This works to get information about the last backup in a single server configuration. In a clustered server configuration, the master node performs backups. However, slave nodes receive all metadata queries, and the slave looks in its own configuration directory for the manifest file (and it's not there). In addition, the designated master node can change without notification, depending on the needs of the cluster.

Follow these steps to ensure that you are looking at the contents of the last backup:

1 Get the name of the last backup by issuing the following request:

```
proc metadata
method=STATUS
in='<MetadataServerBackupHistory
XPath="MetadataServerBackupManifest/Backups/
Backup[POSITION()=LAST()]"/>';
run;
```

The request returns information similar to the following:

```
<Backup Status="Successful"
StartingUserID="META:Scheduler" StartDateTime="2012-10-25T00:59:59-04:00"
Name="2012-10-25T00_59_59-04_00" Directory="Backups/2012-10-25T00_59_59-04_00"
Comment="" Reorg="N" Size="6015741"/>
```

2 Submit the request again, this time specifying the backup's Name value in the <METADATASERVERBACKUPMANIFEST/> XML element as follows:

```
<METADATASERVERBACKUPMANIFEST BackupName="2012-10-25T00_59_59-04_00"/>
```

Getting Information about the Server's Alert Email System

The same XML elements that are submitted in PROC METAOPERATE with ACTION=REFRESH to temporarily change the values of server alert email system options and configuration options can be submitted in PROC METADATA with METHOD=STATUS to get the current values of those options. For a listing of the XML elements, see "Using Alert Email XML Elements" on page 175.

In SAS 9.4M2, the SAS Metadata Server added a reminder system and defined a grace period for responding to alert email messages that report the condition `the journal commit task stopped running`.

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

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

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

The <Scheduler/> XML element is not case-sensitive.

For more information, see "Example 6: Get Information about the Server's Alert Email Notification Subsystem with PROC METADATA" on page 115.

Results: METADATA Procedure

Results: METADATA Procedure

The METADATA procedure produces output in the SAS log or in an XML file. If you do not specify the OUT= argument, the output is written to the SAS log. To send the output to an XML file, you must first submit a FILENAME statement to assign a fileref to the pathname. The file can be temporary or permanent.

In most cases, the output XML string is identical to the input XML string, with the addition of the requested values within the XML elements. XML output is mostly unformatted and difficult to read. To get a more readable representation, you can send the output to an XML file, and then open the XML file in an internet browser such as Internet Explorer. The browser inserts line breaks between the XML elements to make them more readable. For an example of a typical output versus an output that was routed to a file with the OUT= argument, see "Example 1: Get Information about Metadata Repositories" on page 101.

To use the output XML file (for example, to run reports), create an XML map, and then use an XML LIBNAME statement to read the XML file. The XML LIBNAME statement associates the XML map with the XML file so that it can be read by the XML engine as if it were a SAS data set. You can copy the contents to a SAS data set if you choose. Like the output XML file, this SAS data set can be temporary or permanent. For an example that creates a report and reads it with the XML engine, see "Example: Creating a Report with the METADATA Procedure and the XML Engine" on page 16. For more information about the XML engine and XML maps, see the *SAS XMLV2 and XML LIBNAME Engines: User's Guide*.

The VERBOSE= argument does not affect the XML output. It causes the input XML to be written to the SAS log.

Examples: METADATA Procedure

Example 1: Get Information about Metadata Repositories

Features: XML string in the IN= argument
default output vs. fileref in OUT= argument

Note: You must be an administrative user of the metadata server to perform this task.

Details

This example issues the IOMI GetRepositories method to list the metadata repositories that are registered on the SAS Metadata Server. This example also compares the procedure's default output, which is written to the SAS log, to the output returned when the OUT= argument is specified. The OUT= argument writes the output to an XML file.

GetRepositories Request That Returns the Default Output

The default behavior of the GetRepositories method is to return the Id=, Name=, Desc=, and DefaultNS= (namespace) attributes. This method call sets the OMI_ALL (1) flag to return a complete list of each repository's attributes. The default output of a PROC METADATA request is a continuous, unformatted string, which can be unreadable when one or more flags is specified.

```
proc metadata
   in="<GetRepositories>
   <Repositories/>
   <!-- OMI_ALL (1) flag -->
   <Flags>1</Flags>
   <Options/>
   </GetRepositories>";
run;
```

Example Code 11.1 *Log Output from GetRepositories Method*

```
<GetRepositories><Repositories><Repository Id="A0000001.A0000001"
Name="REPOSMGR" Desc="The Repository Manager" DefaultNS="REPOS"
RepositoryType="" RepositoryFormat="15" Access="OMS_FULL"CurrentAccess=
"OMS_FULL" PauseState="" Path="rposmgr" Engine="" Options=""
MetadataCreated="01Jan1960:00:00:00" MetadataUpdated="01Jan1960:00:00:00"/>
<Repository Id="A0000001.A5POKZP3" Name="scratch1" Desc="scratch1"
DefaultNS="SAS" RepositoryType="FOUNDATION" RepositoryFormat="15"
Access="OMS_FULL" CurrentAccess="OMS_FULL" PauseState="" Path="scratch1"
Engine="BASE" Options="" MetadataCreated="06Sep2012:15:35:15"
MetadataUpdated="06Sep2012:15:35:15"/><Repository Id="A0000001.A5A4F7P2"
Name="Custom1" Desc="TestRepository1" DefaultNS="SAS" RepositoryType="Custom"
RepositoryFormat="15" Access="OMS_FULL" CurrentAccess="OMS_FULL" PauseState=""
Path="Custom" Engine="" Options="" MetadataCreated="12Oct2012:14:57:08"
MetadataUpdated="12Oct2012:14:57:08"/><Repository Id="A0000001.A5QYJ9DO"
Name="Custom2" Desc="TestRepository2" DefaultNS="SAS" RepositoryType="Custom"
RepositoryFormat="15" Access="OMS_FULL" CurrentAccess="OMS_FULL" PauseState=""
Path="Custom2" Engine="" Options="" MetadataCreated="12Oct2012:14:58:10"
MetadataUpdated="12Oct2012:14:58:10"/><Repository Id="A0000001.A5PA4NBL"
Name="Custom3" Desc="TestRepository3" DefaultNS="SAS" RepositoryType="Custom"
RepositoryFormat="15" Access="OMS_FULL" CurrentAccess="OMS_FULL" PauseState=""
Path="Custom3" Engine="" Options="" MetadataCreated="12Oct2012:14:58:53"
MetadataUpdated="12Oct2012:14:58:53"/></Repositories><Flags>1</Flags><Options/>
</GetRepositories>
```

GetRepositories Request That Directs Output to an XML File

This PROC METADATA request submits the same GetRepositories method call as the previous request and specifies the OUT= argument to direct the output to an XML file. You must first submit a FILENAME statement, because the OUT= value

accepts a fileref only, not a pathname. When you open the output XML file in a browser, the browser displays formatted XML.

```
filename myoutput "C:\myxml\reports\getrepos.xml";

proc metadata
out=myoutput
in="<GetRepositories>
   <Repositories/>
   <!-- OMI_ALL (1) flag -->
   <Flags>1</Flags>
   <Options/>
   </GetRepositories>";
run;
```

Output 11.1 *Content of Output XML File When Opened in a Browser*

```
<?xml version="1.0"?>
- <GetRepositories>
  - <Repositories>
      <Repository MetadataUpdated="01Jan1960:00:00:00"
          MetadataCreated="01Jan1960:00:00:00" Options="" Engine=""
          Path="rposmgr" PauseState="" CurrentAccess="OMS_FULL"
          Access="OMS_FULL" RepositoryFormat="15" RepositoryType=""
          DefaultNS="REPOS" Desc="The Repository Manager"
          Name="REPOSMGR" Id="A0000001.A0000001"/>
      <Repository MetadataUpdated="06Sep2012:15:35:15"
          MetadataCreated="06Sep2012:15:35:15" Options="" Engine="BASE"
          Path="scratch1" PauseState="" CurrentAccess="OMS_FULL"
          Access="OMS_FULL" RepositoryFormat="15"
          RepositoryType="FOUNDATION" DefaultNS="SAS" Desc="scratch1"
          Name="scratch1" Id="A0000001.A5POKZP3"/>
      <Repository MetadataUpdated="12Oct2012:14:57:08"
          MetadataCreated="12Oct2012:14:57:08" Options="" Engine=""
          Path="Custom" PauseState="" CurrentAccess="OMS_FULL"
          Access="OMS_FULL" RepositoryFormat="15" RepositoryType="Custom"
          DefaultNS="SAS" Desc="TestRepository1" Name="Custom1"
          Id="A0000001.A5A4F7P2"/>
      <Repository MetadataUpdated="12Oct2012:14:58:10"
          MetadataCreated="12Oct2012:14:58:10" Options="" Engine=""
          Path="Custom2" PauseState="" CurrentAccess="OMS_FULL"
          Access="OMS_FULL" RepositoryFormat="15" RepositoryType="Custom"
          DefaultNS="SAS" Desc="TestRepository2" Name="Custom2"
          Id="A0000001.A5QYJ9DO"/>
      <Repository MetadataUpdated="12Oct2012:14:58:53"
          MetadataCreated="12Oct2012:14:58:53" Options="" Engine=""
          Path="Custom3" PauseState="" CurrentAccess="OMS_FULL"
          Access="OMS_FULL" RepositoryFormat="15" RepositoryType="Custom"
          DefaultNS="SAS" Desc="TestRepository3" Name="Custom3"
          Id="A0000001.A5PA4NBL"/>
    </Repositories>
    <Flags>1</Flags>
    <Options/>
</GetRepositories>
```

Example 2: Add an Encoding to The Output XML File

Features: HEADER= argument

Details

By default, PROC METADATA inserts the static header `<?xml version="1.0"?>` in the output XML file that is created when you specify the OUT= argument. The header does not specify an encoding. This example shows two ways that you can add an encoding value to the XML header.

Add the Session Encoding to the XML Header

To add the SAS session encoding to the XML header, specify the HEADER=FULL argument in the PROC METADATA request. When you do not specify HEADER=FULL, the default value is HEADER=SIMPLE, which omits an encoding value.

```
filename myoutput "u:\out2.xml";

proc metadata
   header=full
   out=myoutput
   in="<GetTypes>
      <Types/>
      <Ns>SAS</Ns>
      <Flags/>
      <Options/>
      </GetTypes>";
run;
```

Output 11.2 *Header Created with HEADER=FULL Argument*

```
<?xml version="1.0" encoding="windows-1252" ?>
```

Specify a Custom Encoding for the XML Header

To specify a custom encoding, specify the desired encoding value in the FILENAME statement and specify the HEADER=FULL argument in the PROC METADATA request.

```
filename myoutput "u:\out3.xml" encoding=ascii;

proc metadata
   header=full
   out=myoutput
   in="<GetTypes>
      <Types/>
      <Ns>SAS</Ns>
      <Flags/>
      <Options/>
      </GetTypes>";
run;
```

Output 11.3 *Header Created with ENCODING= Option and HEADER=FULL*

```
<?xml version="1.0" encoding="us-ascii" ?>
```

Example 3: Request the Metadata for One Object

Features:

XML string in the IN= argument

SAS Open Metadata Interface GetMetadata method

OMI_FULL_OBJECT and OMI_SUCCINCT flags

OUT= argument

Details

This method call returns the full set of metadata available for a PhysicalTable metadata object whose object identifier is A5TJRDIT.B2000005.

Program

```
filename myoutput 'C:\myoutput\output.xml';

proc metadata
 in='<GetMetadata>
```

```
      <Metadata>
        <PhysicalTable Id="A5TJRDIT.B2000005"/>
      </Metadata>

     <Ns>SAS</Ns>

      <!-- OMI_FULL_OBJECT (2) + OMI_SUCCINCT (2048) -->
       <Flags>2050</Flags>

     <Options/>
    </GetMetadata>'
  out=myoutput;
run;
```

Program Description

Submit an XML string that contains a GetMetadata method call in the PROC METADATA IN= argument. The GetMetadata method is used to retrieve the values of specified properties for a specified metadata object. The GetMetadata method has four parameters, which are submitted within the XML subelements <Metadata/>, <Ns/>, <Flags/>, and <Options/>. The XML string in this example requests an object of the Physical Table metadata type that has the object identifier A5TJRDIT.B2000005. This information is specified in the GetMetadata method's <METADATA/> subelement.

```
filename myoutput 'C:\myoutput\output.xml';

proc metadata
 in='<GetMetadata>
      <Metadata>
        <PhysicalTable Id="A5TJRDIT.B2000005"/>
      </Metadata>
```

Specify the namespace in the <Ns/> subelement. "SAS" is the valid value.

```
     <Ns>SAS</Ns>
```

Specify the object properties that you want to retrieve. An object's properties include attributes and associations to other objects. By default, the GetMetadata method returns an object's Id= and Name= attributes. You can request specific attributes by specifying their names in the XML string submitted in the <Metadata/> subelement. Or you can specify one or more OMI flags. This example specifies the flags OMI_FULL_OBJECT and OMI_SUCCINCT. OMI_FULL_OBJECT (2) gets the requested values of the specified object, and the properties of both an object's direct and nested associations. The OMI_SUCCINCT (2048) flag specifies to return only information about properties for which values are stored. OMI flags are set by specifying their numeric value in the <Flags/> subelement. To combine GetMetadata flags, you add their numeric values together, and specify the total in the <Flags/> subelement.

```
      <!-- OMI_FULL_OBJECT (2) + OMI_SUCCINCT (2048) -->
       <Flags>2050</Flags>
```

Complete the PROC METADATA request. No options are associated with the specified flags, so the <Options/> subelement is specified as a terminated element. The PROC METADATA OUT= argument is specified to write the output from the request to a file.

```
     <Options/>
```

```
    </GetMetadata>'
  out=myoutput;
run;
```

Output 11.4 *Contents of the results1.xml File When Opened in a Browser*

```
- <GetMetadata>
  - <Metadata>
    - <PhysicalTable Id="A5TJRDIT.B2000005">
      - <Columns>
        - <Column Id="A5TJRDIT.B700000G" Name="State">
          - <Table>
              <PhysicalTable ObjRef="A5TJRDIT.B2000005" Name="EDUCATION"
                PublicType="Table" UsageVersion="1000000" />
            </Table>
          </Column>
        - <Column Id="A5TJRDIT.B700000H" Name="Code">
          - <Table>
              <PhysicalTable ObjRef="A5TJRDIT.B2000005" Name="EDUCATION"
                PublicType="Table" UsageVersion="1000000" />
            </Table>
          </Column>
        - <Column Id="A5TJRDIT.B700000I" Name="DropoutRate" Desc="Dropout
          Percentage - 1989">
          - <Table>
              <PhysicalTable ObjRef="A5TJRDIT.B2000005" Name="EDUCATION"
                PublicType="Table" UsageVersion="1000000" />
            </Table>
          </Column>
        - <Column Id="A5TJRDIT.B700000J" Name="Expenditures" Desc="Expenditure Per
          Pupil - 1989">
          - <Table>
              <PhysicalTable ObjRef="A5TJRDIT.B2000005" Name="EDUCATION"
                PublicType="Table" UsageVersion="1000000" />
            </Table>
          </Column>
        - <Column Id="A5TJRDIT.B700000K" Name="MathScore" Desc="8th Grade Math
          Exam - 1990">
          - <Table>
              <PhysicalTable ObjRef="A5TJRDIT.B2000005" Name="EDUCATION"
                PublicType="Table" UsageVersion="1000000" />
            </Table>
          </Column>
        - <Column Id="A5TJRDIT.B700000L" Name="Region">
          - <Table>
              <PhysicalTable ObjRef="A5TJRDIT.B2000005" Name="EDUCATION"
                PublicType="Table" UsageVersion="1000000" />
            </Table>
          </Column>
        </Columns>
      - <TablePackage>
          <SASLibrary ObjRef="A5TJRDIT.AZ000005" Name="MyTest" PublicType="Library"
            UsageVersion="1000000" />
        </TablePackage>
      - <Trees>
          <Tree ObjRef="A5TJRDIT.AJ000002" Name="Shared Data" Desc="Folder for
            shared libraries, tables, cubes, and information maps." PublicType="Folder"
            UsageVersion="1000000" />
        </Trees>
      </PhysicalTable>
    </Metadata>
    <Ns>SAS</Ns>
    <Flags>2050</Flags>
    <Options />
  </GetMetadata>
```

Example 4: Request the Metadata for One Type of Object

Features: fileref in the IN= argument
SAS Open Metadata Interface GetMetadataObjects method
Use of OMI_XML_SELECT flag

Details

The following example lists information about the SAS Information Maps for relational databases that are registered in a metadata repository. The SAS Open Metadata Interface method call is stored in a temporary input file to simplify quoting requirements for a search string within the XML string and submitted to the server by using a fileref. The results are directed to an output XML file so that the content can be viewed in a browser.

Program

```
filename myinput temp lrecl=256;
filename myoutput "C:\results2.xml" lrecl=256;


data _null_;
   file myinput;
   input;
   put _infile_ ' ';
   datalines;
<GetMetadataObjects>
 <Reposid>$METAREPOSITORY</Reposid>
 <Type>Transformation</Type>
 <Objects/>
 <NS>SAS</NS>
 <-- OMI_XMLSELECT (128) -->
 <Flags>128</Flags>
 <Options>
  <XMLSelect search="*[@PublicType='InformationMap.Relational']"/>
 </Options>
</GetMetadataObjects>
;;
run;

proc metadata
   in=myinput
   out=myoutput;
run;
```

Program Description

Assign input and output filerefs.

```
filename myinput temp lrecl=256;
filename myoutput "C:\results2.xml" lrecl=256;
```

Use a null DATA step to create the temporary XML input file. The PUT statement specifies to write the lines following the DATALINES statement to the location indicated in the FILE statement.

```
data _null_;
   file myinput;
   input;
   put _infile_ ' ';
   datalines;
```

Under the DATALINES statement, specify the XML elements for a GetMetadataObjects method call. The GetMetadataObjects method retrieves information about all objects of a specified metadata type from a specified metadata repository. A GetMetadataObjects method call has XML subelements <Reposid/>, <Type/>, <Objects/>, <Ns/>, <Flags/>, and <Options/>. The <Reposid/> subelement identifies the repository to look in, by repository ID. This example specifies a macro variable, so that different repository ID values can be substituted in the request. A SAS Information Map is represented in a metadata repository with a PrimaryType object of the Transformation metadata type. This type is specified in the <Type> subelement. Several other objects in the type dictionary use the Transformation metadata type as their primary metadata type, including SAS reports. In addition, SAS supports two types of information maps: information maps for relational tables and information maps for cubes. This method call sets the GetMetadataObjects OMI_XMLSELECT flag (128) to filter the request to retrieve only Transformation objects describing information maps for relational tables. This numeric value is specified in the <Flags/> subelement. An information map for a relational table has a TypeName= value of "InformationMap.Relational” in its type definition. This value is specified in the <XMLSelect search=" "/> element, which is submitted in the <Options/> subelement. In all, the query specifies to return Transformation objects that have the value “InformationMap.Relational” in the PublicType= attribute.

```
<GetMetadataObjects>
 <Reposid>$METAREPOSITORY</Reposid>
 <Type>Transformation</Type>
 <Objects/>
 <NS>SAS</NS>
 <-- OMI_XMLSELECT (128) -->
 <Flags>128</Flags>
 <Options>
  <XMLSelect search="*[@PublicType='InformationMap.Relational']"/>
 </Options>
</GetMetadataObjects>
;;
run;
```

Submit the iput and output filerefs to PROC METADATA. The GetMetadataObjects method returns output in the <Objects/> subelement.

```
proc metadata
   in=myinput
   out=myoutput;
```

```
run;
```

Output 11.5 *Contents of the results2.xml File When Opened in a Browser*

Example 5: Get Server Backup Information with PROC METADATA

Features: METHOD= argument
XML element in the IN= argument
IServer Status method backup XML elements

Details

The SAS Metadata Server performs unassisted, scheduled server backups. The following code samples show how to get information about server backups with PROC METADATA.

Note: The recommended interface for managing server backups and performing recoveries is the Server Backup node of SAS Management Console. However, PROC METADATA can also be used to get information about server backups.

Get the Server Backup Location, Retention Policy, and Backup Schedule

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerBackupConfiguration/> XML element. The <MetadataServerBackupConfiguration/> XML element is valid in the inMeta parameter of the SAS Open Metadata Interface IServer Status method.

```
proc metadata
   method=status
   in='<MetadataServerBackupConfiguration/>';
run;
```

Example Code 11.2 *Log Output from Metadata Server Backup Configuration Request*

```
<MetadataServerBackupConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="MetadataServerBackupConfiguration.xsd">
1 <MetadataServer GUID="C678E41B-FC7E-4D90-9C4C-52EAC67427CB"
2 ClusterGUID="B818CCAE-5DC3-4B78-A12C-7F1F34971F70"/>
3 <Schedule Event="Backup" Weekday1="0200" Weekday2="0200" Weekday3="0200"
Weekday4="0200" Weekday5="0200" Weekday6="0200" Weekday7="0200"/></Schedule>
4 <BackupConfiguration BackupLocation="Backups" DaysToRetainBackups="7"
RunScheduledBackups="Y"/></MetadataServerBackupConfiguration>
```

1 The MetadataServer GUID is the server's unique metadata identifier.

2 The ClusterGUID is the cluster's unique metadata identifier. In a clustered server configuration, each backup records the GUID of the server that took the backup, and also records the ClusterGUID. In a clustered environment, a backup will not be recovered unless the ClusterGUIDs in the backup and the target server node match. In a single SAS Metadata Server configuration, there will be no ClusterGUID recorded. A backup should not be used to restore a server unless the MetadataServer GUIDs match.

3 The <Schedule> element contains the backup schedule.

4 The <BackupConfiguration> element contains the backup location (relative to the MetadataServer subdirectory of the server's configuration directory), the retention policy, and an attribute indicating whether automated backups are turned on.

Get the Backup Schedule for a Specific Day

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <Schedule/> XML element to get the backup schedule. Within the <Schedule/> XML element, specify Backup in the Event= attribute, and the attribute for the weekday about which you are inquiring with an empty string. The WeekDay1= attribute is Sunday. This example requests Tuesday's backup schedule.

```
proc metadata
   method=status
   in='<Schedule Event="Backup" WeekDay3=""/>';
run;
```

Get the Backup Retention Policy Only

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <BackupConfiguration/> XML element to get information about the configuration. Within the <BackupConfiguration/> XML element, specify the DaysToRetainBackups= attribute with an empty string.

```
proc metadata
   method=status
   in='<BackupConfiguration DaysToRetainBackups=""/>';
run;
```

Get the Backup History

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerBackupHistory/> XML element to get information about the backup history.

```
proc metadata
   method=status
   in='<MetadataServerBackupHistory/>';
run;
```

Get Information about the Last Backup from the History

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerBackupHistory/> XML element to get information about the backup history. Within the <MetadataServerBackupHistory/> XML element, specify the logical path for the backup location in the XPath= attribute and a query. The MetadataServerBackupManifest.xml file contains a record of the repositories and files copied in a backup. [POSITION()=LAST()] specifies the record's location.

```
proc metadata
   method=status
   in='<MetadataServerBackupHistory
XPath="MetadataServerBackupManifest/Backups/
Backup[POSITION()=LAST()]"/>';
run;
```

Get the Backup Manifest of the Last Backup

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerBackupManifest/> XML element to get the backup manifest. This element returns information about the last backup by default. Note that the steps for getting this information in a clustered server configuration are

different than for a single server configuration. For more information, see “Getting Information about Server Backups ” on page 99.

```
proc metadata
   method=status
   in='<MetadataServerBackupManifest/>';
run;
```

Get the Backup Manifest of an Earlier Backup

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerBackupManifest/> XML element to get a backup manifest. Within the <MetadataServerBackupManifest/> XML element, specify the backup name in the BackupName= attribute. The server assumes the backup in the BackupName= attribute is in the configured backup location.

```
proc metadata
   method=status
   in='<MetadataServerBackupManifest
BackupName="2010-12-13T00_59_59-05_00"/>';
run;
```

Get the Backup Manifest of a Backup That Is Not in the Configured Backup Location

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerBackupManifest/> XML element to get a backup manifest. Within the <MetadataServerBackupManifest/> XML element, specify the backup name in the BackupPath= attribute. Specify the absolute pathname of the backup in the BackupPath= attribute.

```
proc metadata
   method=status
   in='<MetadataServerBackupManifest BackupPath="C:/
2010-12-08T12_44_21-05_00"/>';
run;
```

List the Contents of the MetadataServerRecoveryManifest.xml File

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <MetadataServerRecoveryManifest/> XML element. The server maintains a record of the last recovery only.

```
proc metadata
   method=status
```

```
   in='<MetadataServerRecoveryManifest/>';
run;
```

Check the Health of the Backup Scheduler Thread

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <Scheduler/> XML element to get information about the scheduler. Within the <Scheduler/> XML element, specify the Ping= attribute with an empty string. Possible return values are Alive, TimeOut, Down, or Unconfigured.

```
proc metadata
   method=status
   in='<Scheduler Ping=""/>';
run;
```

Example 6: Get Information about the Server's Alert Email Notification Subsystem with PROC METADATA

Features:

METHOD=STATUS argument

XML element in the IN= argument

IServer Status method alert email XML elements

Details

The examples in this topic show how to get information about the server's alert email notification system with PROC METADATA.

Note: You must submit email server configuration options that are defined in the omaconfig.xml file to the Status method in the case in which they are defined in the omaconfig.xml file. Server configuration options have the form <OMA *ATTRIBUTE_NAME*="*value*"/> in the omaconfig.xml file. That is, all text in the XML elements except the value is specified as uppercase. Other XML elements are not as restrictive.

List the Configured Alert Email Recipients

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <OMA ALERTEMAIL=" "/> XML element with an empty string. This XML

element returns the email addresses to which the SAS Metadata Server sends an email message in the event of a metadata server backup error, a metadata server recovery error, or an error that prevents the repository data sets from being updated from the journal.

```
proc metadata
   method=status
   in='<OMA ALERTEMAIL=""/>';
run;
```

Example Code 11.3 *Log Output from the Alert Email Recipient Request*

```
<OMA ALERTEMAIL="John.Doe@us.company.com"/>
```

Determine the Alert Email Host

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <OMA EMAILHOST=" "/> XML element with an empty string. This XML element returns the network address of the enterprise's SMTP server (for example, mailhost.company.com).

```
proc metadata
   method=status
   in='<OMA EMAILHOST=""/>';
run;
```

Example Code 11.4 *Log Output from the Email Host Request*

```
<OMA EMAILHOST="mailhost.unx.sas.com"/>
```

Determine the Authentication Protocol and Port Number

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <OMA EMAILAUTHPROTOCOL=" "/> and <OMA EMAILPORT=" "/> XML elements with empty strings. <OMA EMAILAUTHPROTOCOL=" "/> returns the authentication protocol for SMTP that is sent by the SAS Metadata Server. Valid values are LOGIN or NONE. <OMA EMAILPORT=" "/> returns the port number that is used by the SMTP server that is configured in the EMAILHOST attribute.

```
proc metadata
   method=status
```

```
   in='<OMA EMAILAUTHPROTOCOL=" "/><OMA EMAILPORT=" "/>';
run;
```

Example Code 11.5 *Log Output from the Email Port and Authentication Protocol Request*

```
<OMA EMAILPORT="25"/><OMA EMAILAUTHPROTOCOL="LOGIN"/>
```

Determine the Frequency of Alert Email Reminders and the Termination Grace Period

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <OMA ALERT_CONDITION_FREQUENCY=" "/> and <OMA ALERT_CONDITION_GRACE_PERIOD=" "/> XMLelements with an empty string. <OMA ALERT_CONDITION_FREQUENCY=" "/> returns the amount of time that will elapse before the initial and subsequent alert email reminders about the alert condition are sent. The time value is returned in seconds. The default value is 21,600 seconds (six hours). The <OMA ALERT_CONDITION_GRACE_PERIOD=" "/> XML element returns the amount of time that the alert condition is allowed to persist before the SAS Metadata Server shuts itself down. The time value is returned in seconds. The default value is 259,200 seconds (three days).

```
proc metadata
   method=status
   in='<OMA ALERT_CONDITION_FREQUENCY=" "/>
   <OMA ALERT_CONDITION_GRACE_PERIOD=" "/>';
run;
```

Example Code 11.6 *Log Output from the Alert Condition Reporting Frequency and Grace Period Request*

```
<OMA ALERT_CONDITION_FREQUENCY="21600"/>
<OMA ALERT_CONDITION_GRACE_PERIOD="259200"/>';
```

Check for Alert Email Conditions

Specify the METHOD= argument with the STATUS value. In the IN= argument, specify the <Scheduler/> XML element and <AlertConditions/> XML subelement to determine whether an alert condition exists on the specified SAS Metadata Server. If an alert condition exists, the <AlertConditions/> subelement returns an

<AlertCondition/> XML element and an <ExpirationTime/> XML element. The <AlertCondition/> element includes the error and a datetime value representing the time at which the error occurred. The <ExpirationTime/> element includes the server's scheduled termination time. Specify the NOREDIRECT argument to process the request on the connected server. NOREDIRECT is ignored if it is specified in a single metadata server configuration.

```
proc metadata
   metaserver="computer.company.com"
   metaport=8564
   metauser="myid"
   metapass="mypassword"
   method=status
in='<Scheduler><AlertConditions/></Scheduler>'
noredirect;
run;
```

Example Code 11.7 *Log Output When No Alert Condition Is Found*

```
<Scheduler><AlertConditions/></Scheduler>
```

Example Code 11.8 *Log Output When Alert Condition Is Found*

```
<Scheduler><AlertConditions><AlertCondition>At 15May2014:18:19:34 the journal
commit task stopped running.</AlertCondition><ExpirationTime>18May2014:18:33:00
</ExpirationTime></AlertConditions></Scheduler>
```

Example 7: Get Information about the Server Cluster with PROC METADATA

Features:

METHOD=STATUS argument
METACONNECT=NONE system option
NOREDIRECT argument
OPTIONS="<CLUSTER/>" argument
IServer Status method cluster XML elements

Details

The examples in this topic show how to get information about a metadata server cluster.

Note: The following examples assume that the default server connection profile has been activated in the SAS session with the METAPROFILE option (the default configuration). When this profile is active, the SAS session re-routes server connections to another server node in the cluster when the node specified in the metadata system options cannot be found. For more information about the default server connection profile, see Specifying a Stored Connection Profile.

Note: In the following examples, all server nodes are installed on one computer. As a result, all nodes have the same host name and are assigned different port numbers. In a typical clustered metadata server configuration, the server nodes are installed on different computers; the nodes use the same port number and have different host names.

Requirement: You must submit server configuration options that are defined in the omaconfig.xml file to the Status method in the case in which they are defined in the omaconfig.xml file. Server configuration options have the form <OMA ATTRIBUTE_NAME=”value”/> in the omaconfig.xml file. That is, all text in the XML elements except the value is specified as uppercase. Other XML elements are not as restrictive.

Determine the Maximum Number of Server Nodes Configured in the omaconfig.xml File

This request can be submitted from any node in the cluster. The attribute name is uppercase because the <OMA MAXIMUM_CLUSTER_NODES=""/> XML element is an omaconfig.xml configuration option.

```
proc metadata
   method=status
   in='<OMA MAXIMUM_CLUSTER_NODES=""/>';
run;
```

Example Code 11.9 *Log Output from the Maximum Cluster Nodes Request*

```
<OMA MAXIMUM_CLUSTER_NODES="8" />
```

Determine the Actual Number of Server Nodes Defined

This request can be submitted from any node in the cluster. The <CLUSTER/> XML element and its attributes are not case-sensitive. This XML element requests properties for the cluster’s Defined_Nodes= attribute.

```
proc metadata
   method=status
   in='<Cluster Defined_Nodes=" " />';
run;
```

Example Code 11.10 *Log Output from the Cluster Defined Nodes Request*

```
<Cluster Defined_Nodes="4" />
```

List the Active Servers in the Cluster

This request must be submitted from the master node. The <Cluster/> XML element in the OPTIONS= argument directs the request to the master node. The <Cluster/> XML element in the IN= argument specifies the LIST= attribute. The LIST= attribute returns information about available server nodes. If the OPTIONS= argument was not specified, the results would include information about two nodes only: the slave node that processed the request and the master node.

```
proc metadata
   method=status
   in='<Cluster List=" "/>'
   options='<Cluster/>'
run;
```

The following output has been reformatted for readability. In the output, the master node is identified by the attribute Self="Y". The output also includes the Name, Host, and Port values of each server node. The Name value indicates the order in which the nodes were defined to the cluster.

Example Code 11.11 *Log Output for a Cluster List Request That Went to the Master Node*

```
<Cluster List="4">
<ClusterNode Self="Y" Name="Node_1" Host="a123" Port="3181" Flags="Master
+FirstNode"/>
<ClusterNode Self="N" Name="Node_2" Host="a123.us.company.com" Port="3182"
Flags="Init+Slave"/>
<ClusterNode Self="N" Name="Node_4" Host="a123.us.company.com" Port="3184"
Flags="Init+Slave"/>
<ClusterNode Self="N" Name="Node_3" Host="a123.us.company.com" Port="3183"
Flags="Init+Slave"/>
</Cluster>
```

Get the Current State of the Cluster

This request must be submitted from the master node. The <Cluster/> XML element in the OPTIONS= argument directs the request to the master node. The <Cluster/>

XML element in the IN= argument requests values for the Defined_Nodes=, Current_Nodes=, Has_First_Node=, and Has_Quorum= cluster attributes. If the OPTIONS= argument was not specified, the results would include information about two nodes only: the slave node that processed the request and the master node.

```
proc metadata
   method=status
   in='<ClusterState/>
      <Cluster Defined_Nodes="" Current_Nodes="" Has_First_Node=""
    Has_Quorum=""/>'
   options='<Cluster/>';
run;
```

Example Code 11.12 *Log Output from Cluster State and Cluster Attribute Requests*

```
<ClusterState>QUORUM</ClusterState>
<Cluster Defined_Nodes="4" Current_Nodes="4" Has_First_Node="Yes" Has_Quorum="Yes"/>
```

Determine the Cluster ID of the Metadata Server Cluster

This request can be submitted to any server node in the cluster.

```
proc metadata
   method=status
   in='<Cluster ClusterGUID=" "/>';
run;
```

Example Code 11.13 *Log Output from a ClusterGUID Request*

```
<Cluster ClusterGUID="33A9C93D-1A94-4DBE-B712-B09B55D57937"/>
```

Direct a Request to a Specific Server Node That Is Not the Master Node

This request specifies the server node that it wants to query using PROC METADATA metadata server connection options. The NOREDIRECT procedure option prevents the load balancer from rerouting the request to a different server node. This request checks for error conditions on a specific server node. You might want to submit the request to all server nodes periodically.

```
proc metadata
metaserver="computer.company.com"
metaport=8564
```

```
metauser="myid"
metapass="mypassword"
method=status
in='<SynchCheck><Results/></SynchCheck>
    <Scheduler><AlertConditions/></Scheduler>
    <OMA ALERTEMAIL=" "/>
    <OMA ALERT_CONDITION_FREQUENCY=" "/>
    <OMA ALERT_CONDITION_GRACE_PERIOD=" "/>
    <OMA SERVERSTARTPATH=" "/>';
noredirect;
run;
```

The following output has been reformatted for readability. <SynchCheck><Results/></SynchCheck> is an IServer Status method XML element introduced in SAS 9.4M3. It returns the results of a cluster synchronization check feature that can be invoked in the SAS Management Console Metadata Manager Analyze/Repair wizard or with the sas-analyze-metadata batch tool. The results indicate that two repositories were checked: the SAS Metadata Server Repository Manager and the Foundation repository. Discrepancies are reported in <Container/> subelements within the <Results/> XML element. The absence of <Container/> subelements indicates that no discrepancies were found. The <Scheduler/> XML element lets you know whether any alert conditions are being managed by the scheduler. This server node does not have any alert conditions in the queue. The next few elements show the server node has the default alert email reminder schedule and grace period. The <OMA SERVERSTARTPATH=" "/> XML element shows the directory from which the server node could be restarted if necessary. For more information about this feature, see the documentation for the Analyze/Repair wizard and the sas-analyze-metadata batch tool in *SAS Intelligence Platform: System Administration Guide*.

Example Code 11.14 *Log Output from the Error Check On the Server Node at Port 8564*

```
<SynchCheck>
<Results>
<Repository Id="A0000001" Name="REPOSMGR"/>
<Repository Id="A5DST1OY" Name="Foundation"/>
</Results></SynchCheck>
<OMA ALERTEMAIL="john.doe@company.com"/>
<OMA ALERT_CONDITION_FREQUENCY="21600"/>
<OMA ALERT_CONDITION_GRACE_PERIOD="259200"/>
<OMA SERVERSTARTPATH="C:\SAS\FoundationServers\Lev4\SASMeta\MetadataServer"/>
```

Chapter 12

METALIB Procedure

Overview: METALIB Procedure

What Does the METALIB Procedure Do?

The METALIB procedure creates, updates, and deletes metadata for data sources in a SAS library. In this documentation, a data source is referred to as a table, whether it is a data table or a view.

When you run PROC METALIB, you must specify a SAS library for which a metadata object is already defined in the SAS Metadata Repository. A SAS library is represented in a SAS Metadata Repository by a SASLibrary object. You can create a SASLibrary object by using the New Library wizard in the SAS Management Console Data Library Manager or in SAS Data Integration Studio. For important information to create a SASLibrary object, see “Considerations When Creating SASLibrary Objects” on page 138.

The METALIB procedure performs the following tasks by default:

- creates metadata for any table in the library that does not have metadata.
- updates existing metadata about the tables' columns, keys, indexes, and integrity constraints to match the current tables in the library.

With optional statements, PROC METALIB can perform the following additional tasks:

- Delete metadata that has no corresponding table in the library.
- Suppress the metadata add action, the metadata update action, or both.
- Add a prefix to the name of new table objects.
- Specify where new metadata is stored in SAS folders.
- Select or exclude specific tables from processing.
- Perform an impact analysis to see whether any Transformation or Job object is associated with the tables. (Information maps are modeled with Transformation objects.)
- Limit the update of table objects that would affect Job or Transformation objects.
- Generate a report of changes that the procedure made to metadata.
- Generate a report of needed metadata changes without making the changes.
- In the generated report, include a list of tables that match the metadata.

For more information, see “How PROC METALIB Works” on page 136.

Syntax: METALIB Procedure

Restriction: This procedure is not supported in SAS Viya.

Requirements: The SAS Metadata Server must be running.

The specified SAS library must already have a SASLibrary object in the SAS Metadata Server. For important information to create a SASLibrary object, see "Considerations When Creating SASLibrary Objects" on page 138.

PROC METALIB assigns libraries in the current SAS session. The library in the SASLibrary object must be accessible to the SAS session from which the procedure is submitted for the procedure to work.

If the data source is ADABAS, you must set the META_ADABAS environment variable to 1.

A user must have WriteMetadata permission to the SAS Metadata Repository and WriteMemberMetadata permission to the target SAS folder to create metadata. In addition, the user must have ReadMetadata permission to update metadata.

PROC METALIB;
OMR <=> (*library-identifier* <*server-connection-arguments*>);
<DBAUTH (DBUSER=*userid* DBPASSWORD=*password*);>
<EXCLUDE <=> (*table-specification(s)*);>
| <SELECT (*table-specification(s)* <READ=*read-password*>;>
<FOLDER= "/*pathname*";> | <FOLDERID= "*identifier.identifier*";>
<IMPACT_LIMIT = *n*;>
<NOEXEC;>
<PREFIX <=> *text*;>
<REPORT <<=> (*report-arguments*)>;>
<UPDATE_RULE <=> (<DELETE> <NOADD> <NOUPDATE>);>

Statement	Task	Example
PROC METALIB	Create and update metadata in the SAS Metadata Repository to match the data source	Ex. 1
OMR	Specify the target library and connection parameters for the SAS Metadata Server	Ex. 1
DBAUTH	Specify database credentials to authenticate to the database.	
EXCLUDE or SELECT	Exclude or select a table or a list of tables for processing	Ex. 3
FOLDER= or FOLDERID=	Specify where new metadata is stored in SAS folders	Ex. 6

Statement	Task	Example
IMPACT_LIMIT	Specify the maximum number of Job or Transformation objects that can be affected by updates to table objects	Ex. 4
NOEXEC	Suppress the metadata changes from being made	Ex. 4
PREFIX	Specify a text string to add to the beginning of the names of new table objects	Ex. 5
REPORT	Create a report that summarizes metadata changes	Ex. 2
UPDATE_RULE	Override the default add, delete, and update behavior	Ex. 2

PROC METALIB Statement

Creates and updates metadata in the SAS Metadata Repository to match the data source.

Syntax

PROC METALIB;

Details

The PROC METALIB statement invokes the METALIB procedure. Secondary statements identify the target SAS library and the specific actions that you want to perform. For processing details, see “How PROC METALIB Works” on page 136.

OMR Statement

Specifies the target SAS library and optional connection parameters for the SAS Metadata Server.

Syntax

OMR <=> (*library-identifier* <*server-connection-arguments*>);

Required Argument

library-identifier

specifies a SASLibrary object, which defines a SAS library, from the SAS Metadata Repository. The SASLibrary object can be identified using any of the following forms:

LIBID=<">*identifier*<">

specifies the 8-character metadata identifier of the SASLibrary object that represents the library. The 8-character metadata identifier is the second half of the 17-character identifier. For more information, see "What Is a Metadata Identifier?" on page 13 and "Obtaining Metadata Names and Identifiers" on page 13.

LIBRARY=<">*name*<">

specifies the value in the SASLibrary object's Name attribute.

LIBURI="*URI-format*"

specifies a URI, which is a standard from SAS Open Metadata Architecture. For more information, see "What Is a URI?" on page 14.The following URI formats are supported:

LIBURI="*identifier.identifier*"

specifies the full 17-character metadata identifier, which references both the repository and the object. This syntax is equivalent to specifying both LIBID= and REPID=. An example is **`liburi="A58LN5R2.A9000001"`**.

LIBURI="SASLibrary/*identifier.identifier*"

specifies the SASLibrary object type, followed by the full 17-character metadata identifier. This syntax is equivalent to specifying both LIBID= and REPID=. An example is **`liburi="SASLibrary/A58LN5R2.A9000001"`**.

LIBURI="SASLibrary?@*attribute*='*value*'"

specifies the SASLibrary object type, followed by a search string. Examples are **`liburi="SASLibrary?@libref='mylib'"`** and **`liburi="SASLibrary?@engine='base'"`**.

Requirement The URI must resolve to a single metadata object. When using an attribute qualifier such as @engine='base', if more than one Base library is defined in metadata, PROC METALIB returns `WARNING: Multiple metadata objects found.`

Requirement You must enclose the LIBURI= value in quotation marks.

Note SAS Data Integration Studio can process work tables that exist temporarily in the Work library. The metadata type is WorkTable. Usually, work tables are not assigned to a library and have no library metadata, but they do have table and column metadata. A work table that results from a generated transformation can be dynamic in nature. In other words, its structure might be modified by the transformation. PROC METALIB can update the metadata to match the work table. If there is no library assignment, submit a blank library specification, and identify the work table with the SELECT statement. Here is an example with a blank library specification: `proc metalib; omr (libid="" repid="A5O7HLNB"); select ("A5O7HLNB.A9000001"); run;`

Optional Arguments

The following server connection arguments establish communication with the SAS Metadata Server. If you omit these arguments, then the values of the system options are used, or the values can be obtained interactively. For more information, see “Connection Options ” on page 38.

PASSWORD="*password*"
is the password for the authorized user ID on the SAS Metadata Server. If you do not specify PASSWORD=, the value of the METAPASS= system option is used. For more information, see “METAPASS= System Option” on page 49. The maximum length is 256 characters.

Alias METAPASS= or PW=

PORT="*number*"
is the TCP port that the SAS Metadata Server listens to for connections. This port number was used to start the metadata server. If you do not specify PORT=, the value of the METAPORT= system option is used. For more information, see “METAPORT= System Option” on page 50. The range of allowed port numbers is 1 to 65535. The metadata server is configured with a default port number of 8561.

Alias METAPORT=

REPID=<">*identifier*<"> | REPNAME=<">*name*<">
specifies the repository that contains the SASLibrary object. If you specify both REPID= and REPNAME=, REPID= takes precedence over REPNAME=. If you do not specify REPID= or REPNAME=, the value of the METAREPOSITORY= system option is used. For more information, see “METAREPOSITORY= System Option” on page 53. The default for the METAREPOSITORY= system option is FOUNDATION.

REPID=<">*identifier*<">
specifies an 8-character identifier. This identifier is the first half of the SASLibrary object's 17-character identifier, and is the second half of the repository's identifier. For more information, see “What Is a Metadata Identifier?” on page 13 and “Obtaining Metadata Names and Identifiers” on page 13.

REPNAME=<">*name*<">
specifies the value in the repository's Name= attribute. The maximum length is 256 characters.

Alias METAREPOSITORY=

SERVER="*host-name*"
is the host name or network IP address of the computer that hosts the SAS Metadata Server. The value LOCALHOST can be used if the SAS session is connecting to a server on the same computer. If you do not specify SERVER=, the value of the METASERVER= system option is used. For more information, see “METASERVER= System Option” on page 54. The maximum length is 256 characters.

Alias HOST= or IPADDR= or METASERVER=

USER="*authorized-user-ID*"
is an authorized user ID on the SAS Metadata Server. An authorized user ID has ReadMetadata and WriteMetadata permission to the specified SASLibrary. It has

WriteMemberMetadata permission to the SAS folders that are affected by the update. SAS folders that can be affected by the update include the library's folder and the table's folder, if the table is in a different folder from the library. For more information, see *SAS Intelligence Platform: Security Administration Guide*. If you do not specify USER=, the value of the METAUSER= system option is used. For more information, see "METAUSER= System Option" on page 57. The maximum length is 256 characters.

Alias ID= or METAUSER= or USERID=

DBAUTH Statement

Specifies database credentials to authenticate to the database.

Notes: The behavior of this statement has changed. Beginning in SAS 9.4M6, the credentials in the DBAUTH statement override any other predefined authentication type. In SAS releases prior to SAS 9.4M6, the authentication credentials specified in the DBAUTH statement are overridden in favor of other predefined authentication types.

The DBAUTH statement was introduced in SAS 9.4M3.

Syntax

DBAUTH (DBUSER=*userid* DBPASSWORD=*password*);

Required Arguments

DBUSER=<">*user-name*<">
specifies the name of a database user account that can access the library on the database.

Note A user name value is changed to uppercase text if it is not enclosed in quotation marks. If a value is case sensitive or contains special characters, it has to be enclosed in either single or double quotation marks.

DBPASSWORD=<">*password*<">
specifies the password associated with the specified database user account.

Note A password value is changed to uppercase text if it is not enclosed in quotation marks. If a value is case sensitive or contains special characters, it has to be enclosed in either single or double quotation marks.

Tip We recommend that you encode the password before submitting it in the DBAUTH statement. You can encode the password with PROC PWENCODE. For more information about PROC PWENCODE, see *Base SAS Procedures Guide*.

Details

Normally, the METALIB procedure uses credentials that are stored in the server definition that is associated with the specified library to connect to a data source. The use of metadata-based credentials is desirable because it saves users from having to know the database user ID and password. It enables administrators to define additional security beyond what is enforced by the database. Metadata-based credentials continue to be the recommended mode of data source authentication.

The DBAUTH statement was initially provided to enable batch programs to supply the credentials for libraries that have an authentication type of Prompt set in their server definitions. Normally, when the authentication type is Prompt, the SAS Metadata Server displays a logon dialog box to prompt users for the credentials. The DBAUTH statement suppresses the logon dialog box and provides the credentials programmatically instead. If any stored credentials were found in the metadata repository, the DBAUTH statement had no effect.

Beginning in SAS 9.4M6, the DBAUTH statement behaves the same way as the metadata engine's DBUSER= and DBPASSWORD= options. The database authentication credentials in the DBAUTH statement override any predefined authentication types, including credentials that are stored in an authentication domain.

Here is an example of how the DBAUTH statement is specified:

```
proc metalib;
 omr(library="oralib_prompt");
 select(dept);
 dbauth(dbuser=scott dbpassword="{SAS002}F77E0C345A42C6A753443DCE");
 run;
```

EXCLUDE or SELECT Statement

Excludes or selects a table, or a list of tables, for processing.

Requirement: Use either EXCLUDE or SELECT, not both. Use one form of table specification (that is, either *table-name* or *table-identifier*).

Interaction: When you specify a SELECT or EXCLUDE statement, if any table in the selected list has a foreign key or a unique key that is referenced by a foreign key, be sure to include the related foreign key and unique key tables in the selected list. Otherwise, the foreign key definition will not be added or updated in the repository.

Syntax

EXCLUDE<=>(*table-specification(s)*)
| **SELECT**<=>(*table-specification(s)* <READ=*read-password*>);

Required Argument

(*table-specification*)

<">table-name<"> <<">table-name-n<">>
is the SAS name of one or more physical data sources in the location that is referenced by the specified SASLibrary object.

If the underlying engine associated with the SASLibrary object supports case-sensitive identifiers and any of the listed table names contains mixed-case elements or special characters, you must enclose each table name in quotation marks. Otherwise, SAS treats the table name as uppercase. In the following example, all of the values must be enclosed in quotation marks if you want the casing of the table names preserved. If the values were not enclosed in quotation marks, they would be uppercase as TAB1, TAB2, TAB3, and TABLE4.

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

<">reposid.tableid<">
is the full 17-character metadata identifier of a PhysicalTable object. The identifier is valid for SELECT, but not for EXCLUDE. For more information, see "What Is a Metadata Identifier?" on page 13 and "Obtaining Metadata Names and Identifiers" on page 13. Quotation marks are optional.

Note: SAS Data Integration Studio can process work tables that exist temporarily in the Work library. See the note about a blank library specification at "OMR Statement" on page 126.

Optional Argument

read-password
is the READ password, if any, that was previously assigned to the table. For information about file protection, see *SAS Language Reference: Concepts*. The following example specifies a READ password for tab1:

```
select ("tab1" read=mypwd "tab2" "tab3" "Table4");
```

FOLDER= or FOLDERID= Statement

Specifies where new metadata is stored in SAS folders.

See: "Example 6: Specifying a Folder for the Metadata" on page 150

Syntax

FOLDER = "/*pathname*" | **FOLDERID** = "*identifier.identifier*";

Required Arguments

FOLDER= "/*pathname*"
is the pathname to an existing folder in the SAS folder tree. If the specified folder does not exist, or if the name of the folder is misspelled, PROC METALIB returns an error. The pathname begins with a forward slash and is relative to the branch of the folder tree in which the folder resides. Here is an example:

```
folder="/User Folders/MyUserID/My Folder/Test";
```

Restriction The pathname cannot have more than nine nested subfolders and cannot exceed 512 characters in length.

Requirement The metadata server interprets a parenthetical phrase that appears at the end of the path specification to be the name of a valid metadata object type. Include a slash (/) at the end of the path specification when it includes a value enclosed in parentheses. For example:

```
folder="/SharedData/Schemas/par(en)/";
```

FOLDERID= "*identifier.identifier*"
is the full 17-character metadata identifier of the Tree object that represents the folder. Using FOLDERID= is not recommended if you can use FOLDER=. The FOLDER= syntax is preferable because it shows the location of the folder in SAS Management Console.

Details

PROC METALIB creates and updates table objects in the folder indicated by the specified SASLibrary object unless you specify a different folder with the FOLDER= or FOLDERID= statement. The default location for a SASLibrary object is the Shared Data folder. When you specify FOLDER= or FOLDERID=, you add or update the table object in the specified SAS folder instead. Column, ForeignKey, Index, KeyAssociation, and UniqueKey objects are added or updated in the same folder as the specified PhysicalTable object. The SASLibrary object remains in its original folder.

If a table is defined in more than one folder, updating the table object in all of the folders is recommended. And, you must submit a PROC METALIB step for each folder. Using the SELECT= statement is recommended to ensure that you update the correct table. If a table is defined in more than one folder, then you will see multiple table objects in the Data Library Manager on the **Plug-ins** tab of SAS Management Console. The multiple table objects will have the same name, but they will be in different SAS folder locations. Every table object has a unique metadata identifier.

IMPACT_LIMIT Statement

Specifies the maximum number of Job or Transformation objects that can be affected by an update to a table object.

See: “Example 4: Performing an Impact Analysis” on page 145

Syntax

IMPACT_LIMIT=*n*;

Required Argument

n

maximum number (an integer) of Job or Transformation objects that can be affected by an update to a table object. For each table that is analyzed, if the specified number is exceeded, the table's metadata is not added, updated, or deleted.

Details

The IMPACT_LIMIT statement is optional. An impact analysis is not performed unless IMPACT_LIMIT and REPORT are specified.

The recommended usage is as follows:

1. Specify IMPACT_LIMIT=0 with REPORT to determine what tables have associated Job or Transformation objects.
2. Specify IMPACT_LIMIT=0 with REPORT (TYPE=DETAIL) to identify which type of object is associated: Job or Transformation.
3. Specify IMPACT_LIMIT with an integer that specifies the number of Job or Transformation objects found. Any updates to table objects will be made.

IMPACT_LIMIT identifies potential impact only. It does not verify that a Job or Transformation object *was* affected, only that it *could be* affected.

IMPACT_LIMIT identifies only the Job or Transformation objects that can be directly affected. These objects might contain other objects that could be affected down the line by the changes, but those objects are not analyzed. If you would like to perform a more thorough impact analysis, you can use SAS Data Integration Studio.

For more information about Job and Transformation objects, see the online Help in SAS Data Integration Studio.

NOEXEC Statement

Suppress the metadata changes from being made.

Syntax

NOEXEC;

Details

If you specify NOEXEC and the REPORT statement, you can generate a report of changes that your request would make to metadata before you commit to making the changes. The SAS log contains warnings about any tables that have metadata, but no longer exist in the library.

PREFIX Statement

Specifies a text string to add to the beginning of the name of new table objects.

See: "Example 5: Adding a Prefix to New Metadata Names" on page 149

Syntax

PREFIX <=> *text*;

Required Argument

<">text<">
is the text string to add. The text string is prepended to the value in the Name attribute of the current table objects being created. By default, the SAS table name is stored in a table object's Name attribute. Defining a prefix modifies the value in the Name attribute. Together, the length of the SAS table name and the prefix cannot exceed 60 characters. If the text string includes special characters or to retain the casing in the string as it is entered, enclose the text string in quotation marks. Otherwise, the text is converted to uppercase.

REPORT Statement

Creates a report that summarizes metadata changes in the Output window.

Default: TYPE=SUMMARY report

See: "Example 4: Performing an Impact Analysis" on page 145

Syntax

REPORT <<=> (*report-arguments*)>;

Optional Arguments

TYPE=DETAIL | SUMMARY

DETAIL
specifies that the report includes all of the information generated by TYPE=SUMMARY. In addition, the report includes the list of Job and Transformation objects that are related to the tables that are being processed.The "IMPACT_LIMIT Statement" statement must also be specified to include the list of Job and Transformation objects.

SUMMARY
specifies that the report includes information about any metadata changes that were (or will be) made to the table that is being processed.

When specified with IMPACT_LIMIT, the following occurs:

- Only tables for which Job or Transformation objects are associated are listed. (This is also known as an impact analysis.)
- No changes are made unless IMPACT_LIMIT is greater than zero.

MATCHING
specifies that the report includes a list of tables whose metadata matches their data sources (that is, they require no metadata changes). By default, the report does not include the list of these matching tables, but it does include the number of matching tables.

Details

The REPORT statement is optional. If it is omitted from the PROC METALIB request, PROC METALIB writes summary information to the SAS log. Specifying REPORT without any report arguments causes the output to be written to the Output window. For more information, see Results: METALIB Procedure on page 139 .

UPDATE_RULE Statement

Overrides one or more of the default add, update, and delete actions.

Requirement: An error is returned if you specify NOADD and NOUPDATE and omit DELETE. The procedure must have an action to perform if both the add and update actions are suppressed.

Syntax

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

Required Arguments

DELETE
specifies to delete a table object in the repository if a corresponding data source is not found. PROC METALIB ignores table objects that have no corresponding data sources by default.

NOADD
specifies not to add table objects to the repository. By default, PROC METALIB creates a new table object for any data source in the library whose SAS name does not match the name of a table object in the current repository.

NOUPDATE
specifies not to update existing table objects in the current repository to match the corresponding data sources. PROC METALIB updates existing table objects to match the corresponding data sources by default.

Usage: METALIB Procedure

How PROC METALIB Works

When submitted without options, PROC METALIB compares the SAS tables in the SAS library indicated by the specified SASLibrary object to the table objects in the SAS folder indicated by the SASLibrary object. If you did not specify a SAS folder when defining the SASLibrary object, the Shared Data folder is used. You can request that the procedure compare the SAS tables to the objects in a different SAS folder by using the FOLDER statement.

The procedure compares all tables in the SASLibrary to all table objects in the folder by default. You can limit processing to specific tables with the SELECT statement. You can exclude tables from processing with the EXCLUDE statement.

The procedure performs update processing and add processing only by default. It performs update processing before it performs add processing. That is, the procedure compares and updates the table objects that are already in the SAS folder to the tables in the SAS library. Then, it attempts to create new table objects for SAS tables that do not have objects in the folder.

The procedure uses the value in the table objects TableName attribute to match the SAS tables to table objects for update processing. The TableName attribute stores the SAS table name of the table that the object describes.

The procedure uses the value in the table objects Name attribute for add processing. The value in a table object's Name attribute is the same as the value in the TableName attribute by default (the SAS table name), unless you specify a prefix with the PREFIX statement when the object is created. The PREFIX statement adds a text string at the beginning of the table name stored in the Name attribute. Because add processing looks for a matching SAS table name in the Name attribute, this is a way to guarantee that a new table object is created for an object the next time that you submit the METALIB procedure. The PREFIX

statement gives you the flexibility to create multiple table objects for a given SAS table in a SAS folder. Use of the TableName attribute for update processing ensures that all table objects that describe a given SAS table in a folder are updated.

You can request that the procedure delete table objects for which a corresponding SAS table no longer exists by using the UPDATE_RULE statement and specifying the DELETE option. When the UPDATE_RULE is DELETE, this delete process precedes the default update and add processes.

You can suppress one of the default update or add processes at any time by specifying NOUPDATE or NOADD in the UPDATE_RULE statement. Do not specify both options at the same time unless you also specify DELETE. Otherwise, the procedure has nothing to do.

PROC METALIB writes summary information to the SAS log by default. You can request that the information be written to the Output window by specifying the REPORT statement without options. The summary information notes which tables were updated, added, and deleted, and the number of table objects that did not require a metadata update. A MATCHING option includes the names of the table objects that did not require metadata changes in the summary report. A DETAIL option prints information about Job or Transformation objects that can be affected by an update to a table object when the IMPACT_LIMIT statement is specified with the REPORT statement. If you specify the NOEXEC statement and the REPORT statement, you can generate a report of changes that your request makes to metadata before you commit to making the changes. The SAS log contains warnings about any tables that have metadata but no longer exist in the library.

What Metadata Is Updated?

A SAS table object consists of objects of the following SAS Metadata Model metadata types: PhysicalTable, Column, Index, UniqueKey, ForeignKey, UniqueKey, and KeyAssociation. Each is updated, as appropriate to describe any given SAS table, with these exceptions:

- PROC METALIB does not create Index, UniqueKey, and ForeignKey metadata objects for external databases that are accessed with the SAS/SHARE server.
- PROC METALIB does not create metadata for indexes on which expressions are defined. When an expression is defined on an index, the index is ignored. Table metadata is created without metadata for the index.

PROC METALIB populates the PhysicalTable ID, Name, Desc, IsDBMSView, MemberType, PublicType, TableName, and MetadataCreated attributes when it adds a table object. Additional attributes might be populated for a table object if the table object was created or updated by another mechanism. For example, a table object created by the SAS Import wizard might populate additional attributes. PROC METALIB preserves the additional attributes populated or updated by other mechanisms in its update processing.

Beginning with SAS 9.4M2, the procedure does not presume that a table object will always be updated with the SASLibrary object that was used to create it. When invoked on an existing table object, the procedure checks the table object's library ownership. The library that owns a table object is identified in the PhysicalTable object's TablePackage association. If the SASLibrary object that is being used to update the table object is different from the SASLibrary object that was used to create the table object, the TablePackage association is updated with information about the new library. This approach provides flexibility for exporting and importing

data between different servers. In this way, a Base SAS table that was exported to Oracle can be updated to reflect that the source library is now an Oracle library.

PROC METALIB imports column attributes to the table object, except the DBSASTYPE data set option setting, to match the columns in the SAS table. You can manually adjust a column's attributes using SAS Management Console or SAS Data Integration Studio, and the metadata LIBNAME engine uses the modified attributes. However, if you do this, you must manually adjust the column's attributes after each run of PROC METALIB or exclude tables whose attributes you have manually modified from update processing. Otherwise, the updates are lost in the update process.

For more information about the SAS Metadata Model metadata types that make up a table object, see their descriptions in the SAS Metadata Model: Reference.

Considerations When Creating SASLibrary Objects

Here are some important considerations when creating SASLibrary objects:

- When you create a SASLibrary object in SAS Management Console or SAS Data Integration Studio, choose the resource template that is specific to the type of data source library that you are creating. For example, use the **SAS BASE Library** template, or the template for the specific database library. Do not use the **Pre-assigned Library** template. Library definitions created with the Pre-assigned Library template cannot be assigned using metadata.
- To pre-assign the SAS library described in the resource template, select the **This library is pre-assigned** check box on the **Advanced** tab of the resource template, and then select the pre-assignment type **By Native Engine**. PROC METALIB might fail if the library is pre-assigned with the **By Metadata Engine** pre-assignment type. The METALIB procedure gains access to table data from the assigned library. When the pre-assignment type is **By Metadata Engine,** the procedure can access only tables that are already described by metadata, and only as the tables are described. The existing metadata cannot be updated, and no new table objects can be added. For PROC METALIB to create and update metadata, it must have access to the physical tables in the library.

See *SAS Intelligence Platform: Data Administration Guide* for detailed instructions on how to create a SASLibrary object.

SASLibrary Objects and Folders

PROC METALIB creates table objects in the folder indicated by the specified SASLibrary object unless you specify a different folder with the FOLDER= or FOLDERID= statement. If a folder is not specified in the SASLibrary object, the default location is the `Shared Data` folder.

If you choose to create table objects in more than one folder, be aware that you need to run PROC METALIB on each folder to update the metadata. PROC METALIB maintains a separate set of table objects for each folder.

Table object names within a folder must be unique. A folder is not limited to table objects from a single library. You can register tables from different libraries in the same folder as long as the table names are unique.

To ensure that a table name is unique within a folder or across multiple folders, you can use the PREFIX statement. The PREFIX statement enables you to add a string value to the beginning of table object name to make the metadata name unique. If you rename a table object in SAS Management Console, the name is not preserved the next time you run PROC METALIB.

For more information, see "Example 6: Specifying a Folder for the Metadata" on page 150. Also see "FOLDER= or FOLDERID= Statement" on page 131.

Results: METALIB Procedure

Introduction

By default, regardless of whether you specify the REPORT statement, the METALIB procedure writes a summary to the SAS log of changes that were made to the metadata. Here is an example:

```
NOTE: A total of 10 tables were analyzed for library "mylib".
NOTE: Metadata for 2 tables was updated.
NOTE: Metadata for 0 tables was added.
NOTE: Metadata for 7 tables matched the data sources.
NOTE: 1 other tables were not processed due to error or UPDATE_RULE.
```

If you specify the REPORT statement, a detailed report is written to the SAS Output window. The report provides the same summary as the SAS log, and also lists the changes to tables and their Column, ForeignKey, Index, KeyAssociation, and UniqueKey objects.

Some procedure arguments add information to the report.

- If you specify UPDATE_RULE=(DELETE), the report lists the number of table objects that were deleted from metadata.
- If you specify the SELECT or EXCLUDE statement, the report lists the number of tables that were not found in either source (data source or metadata).
- If you specify MATCHING in the REPORT statement, the report lists the tables that match the metadata.
- If you specify TYPE=DETAIL in the REPORT statement, and you specify the IMPACT_LIMIT statement, the report lists the number of tables that were not processed because of large impact. It also lists Job and Transformation objects that are directly related to the table that is being processed.

If you specify the NOEXEC statement, the procedure does not make any of the changes to the metadata. The SAS log and Output window summarize the metadata changes that would have been applied if NOEXEC had not been specified.

For information about REPORT statement syntax, see "REPORT Statement" on page 134.

Output Format

The default report destination is HTML.

Details in the Report

The METALIB procedure updates the attribute values of the table object and the attribute values of associated objects to match the data in the specified SAS library. The procedure then produces a report. Most of the report is self-explanatory. Here is more information about two of the columns in the report:

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

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

Change
is a system-generated description of the change that was made. The description can be a single word, such as “Added” or “Deleted”, or it can be an attribute name (which indicates that the attribute's value was modified). It can be a “Column” or a “Column Order” message, followed by the name of the column that was affected by the change. PROC METALIB changes a table's Columns association to make the metadata column order match the data source column order. Affected columns are listed separately in the report. The column order in the report indicates the new metadata column order.

Examples: METALIB Procedure

Example 1: Creating Metadata for a Data Source

Features: OMR statement with LIBID= argument
REPORT statement

Details

Note: To read or write a metadata definition, you must first establish a connection to the SAS Metadata Server. All of the examples in this section assume that a server connection was previously established by using metadata system options.

This example creates metadata that describes the physical data sources in a new SAS library in a SAS Metadata Repository. The SAS library must already exist and contain SAS data. You must have already created a metadata definition for the SAS library in the SAS Management Console Data Library Manager.

Program

```
proc metalib;
     omr (libid="AZ00000A");

   report;
 run;
```

Program Description

Identify the SAS library for which you want to create metadata in the OMR statement. PROC METALIB creates new metadata and updates any existing metadata for the SAS library identified in the OMR statement. In this request, the SASLibrary object is identified in the LIBID= argument by its 8-character metadata identifier.

```
proc metalib;
     omr (libid="AZ00000A");
```

Specify the REPORT statement. The REPORT statement without options creates a default summary report of the tasks that were performed by PROC METALIB.

```
   report;
 run;
```

The summary report shows that PROC METALIB added metadata definitions for seven tables.

Output 12.1 *Summary of New Table Metadata Created by PROC METALIB*

The SAS System

The METALIB Procedure

Summary Report for Library AZ00000A
Repository A0000001.A5JTOPDN
15DEC2010

Metadata Summary Statistics	
Total tables analyzed	7
Tables Updated	0
Tables Added	7
Tables matching data source	0
Tables not processed	0

Tables Added		
Metadata Name	**Metadata ID**	**SAS Name**
AUTO	A5JTOPDN.B400000R	AUTO
CENSUS	A5JTOPDN.B400000S	CENSUS
CENSUS1990	A5JTOPDN.B400000T	CENSUS1990
EDUCATION	A5JTOPDN.B400000U	EDUCATION
ENERGY	A5JTOPDN.B400000V	ENERGY
GROC	A5JTOPDN.B400000W	GROC
SALES	A5JTOPDN.B400000X	SALES

Example 2: Synchronizing Metadata with the Data Source

Features: OMR statement
UPDATE_RULE statement with DELETE argument
REPORT statement with MATCHING argument

Details

This example adds, updates, and deletes existing metadata in a SAS library in the SAS Metadata Repository to match the current data sources in the SAS library.

Program

```
proc metalib;
   omr (libid="AZ00000A");

   update_rule=(delete);
   report(matching);
run;
```

Program Description

Identify the SAS library in the OMR statement. This example specifies the same library that was in the previous example.

```
proc metalib;
   omr (libid="AZ00000A");
```

Specify the UPDATE_RULE statement with the DELETE argument. The DELETE argument specifies to delete any table object that does not correspond to a table in the SAS library. The default actions of add and update are also performed.

```
   update_rule=(delete);
```

Specify the REPORT statement with the MATCHING argument. The MATCHING argument causes the report to include a list of tables whose metadata matches the data source. If you do not specify the MATCHING argument when synchronizing existing metadata, and if there has been no change to the data source that would result in adding, updating, or deleting metadata, the REPORT statement returns only summary statistics.

```
   report(matching);
run;
```

The report shows that one table was updated, two tables were added, and six tables matched the data source.

Output 12.2 *Summary of Metadata Updated by PROC METALIB with UPDATE_RULE=(DELETE)*

The SAS System

The METALIB Procedure

Summary Report for Library AZ000006
Repository A0000001.A5TJRDIT
02FEB2011

Metadata Summary Statistics	
Total tables analyzed	9
Tables Updated	1
Tables Deleted	0
Tables Added	2
Tables matching data source	6
Tables not processed	0

Tables Updated

Table			Updates				
Metadata Name	Metadata ID	SAS Name	Metadata Name	Metadata ID	SAS Name	Metadata Type	Change
EDUCATION	A5TJRDIT.B20000T7	EDUCATION	State	A5TJRDIT.B70000XG	State	Column	IsDiscrete
			State	A5TJRDIT.BI000008	State	Index	Added
			EDUCATION.ic_id	A5TJRDIT.BH000008	ic_id	UniqueKey	Added

Tables Added

Metadata Name	Metadata ID	SAS Name
GROCSALES	A5TJRDIT.B20000TB	GROCSALES
STUDENTS	A5TJRDIT.B20000TC	STUDENTS

Tables matching data source

Metadata Name	Metadata ID	SAS Name
SALES	A5TJRDIT.B20000TA	SALES
GROC	A5TJRDIT.B20000T9	GROC
ENERGY	A5TJRDIT.B20000T8	ENERGY
CENSUS1990	A5TJRDIT.B20000T6	CENSUS1990
CENSUS	A5TJRDIT.B20000T5	CENSUS
AUTO	A5TJRDIT.B20000T4	AUTO

Example 3: Selecting Tables for Processing

Features: OMR statement with LIBURI= argument
SELECT statement

Details

This example adds or updates metadata for a specific table. The table can be identified by specifying its table name or metadata identifier.

Select a Table by Name

To select a table by name, specify the value in the SASTableName= attribute of its table object in the SELECT statement. This table has the value “mytable” in the SASTableName= attribute. Because the UPDATE_RULE statement is omitted, the default is to update or add the specified metadata. Therefore, if a table with this SAS table name does not exist, a new table object is created.

```
proc metalib;
   omr (liburi="SASLibrary?@name='MyTestLibrary'");
   select (mytable);
run;
```

Select a Table with Its Metadata Identifier

This syntax is preferred because metadata identifiers are unique. The first part of the two-part metadata identifier (A7892350) identifies the repository that contains the table object. The second part (B00265DX) identifies the table object in the repository.

```
proc metalib;
   omr (liburi="SASLibrary?@name='MyTestLibrary'");
  select (A7892350.B00265DX);
run;
```

Example 4: Performing an Impact Analysis

Features: IMPACT_LIMIT statement
REPORT statement
REPORT(TYPE=DETAIL) statement

Details

The IMPACT_LIMIT statement can be used with the REPORT statement in PROC METALIB to obtain information about Job and Transformation objects that are potentially affected by updates to table objects. The statements can be used to determine what tables have associated Job or Transformation objects, and which type of object is associated. The tables are not updated unless the IMPACT_LIMIT statement specifies an integer matching the outstanding number of objects that are found.

Determine Which Tables Have Associated Objects

Specify the library that you want to examine in the OMR statement, specify the IMPACT_LIMIT= statement with a value of 0 (zero), and specify the REPORT statement. Because the impact limit is set to zero, any impact on a Job or Transformation object results in an impact limit exceeded entry in the output. Library AZ000009 has two tables that are potentially associated with a Job or Transformation object: EMPINFO and SALARY.

```
proc metalib;
   omr (libid=AZ000009);
   impact_limit=0;
   report;
run;
```

Output 12.3 *Summary Report of Tables That Have Associated Job or Transformation Objects*

The SAS System

The METALIB Procedure

Summary Report for Library AZ000009
Repository A0000001.A5TJRDIT
01FEB2011

Metadata Summary Statistics	
Total tables analyzed	10
Tables not processed due to large impact	2
Tables Updated	0
Tables Added	0
Tables matching data source	8
Tables not processed	0

Tables Exceeding Impact Limit of 0			
Metadata Name	**Metadata ID**	**SAS Name**	**Total Impact**
EMPINFO	A5TJRDIT.B200000J	EMPINFO	1
SALARY	A5TJRDIT.B200000R	SALARY	1

Determine the Type of the Associated Objects

Specify the library that you want to examine in the OMR statement, specify the IMPACT_LIMIT= statement with a value of zero, and specify REPORT (TYPE=DETAIL). The detail report specifies the object's metadata type and potential role within the table. The output shows the objects are of type Transformation and are part of an information map.

```
proc metalib;
   OMR=(libid=AZ000009);
   impact_limit=0;
   report (type=detail);
run;
```

Output 12.4 *Detailed Report of Objects Associated with Each Table*

The SAS System

The METALIB Procedure

Detail Report for Library AZ000009
Repository A0000001.A5TJRDIT
01FEB2011

Metadata Summary Statistics	
Total tables analyzed	10
Tables not processed due to large impact	2
Tables Updated	0
Tables Added	0
Tables matching data source	8
Tables not processed	0

Tables Exceeding Impact Limit of 0			
Metadata Name	**Metadata ID**	**SAS Name**	**Total Impact**
EMPINFO	A5TJRDIT.B200000J	EMPINFO	1
SALARY	A5TJRDIT.B200000R	SALARY	1

Potential-Impact Analysis						
Table			**Impact**			
Metadata Name	**Metadata ID**	**Action**	**Metadata Name**	**Metadata ID**	**Metadata Type**	**Transformation Role**
EMPINFO	A5TJRDIT.B200000J	Limit Exceeded	Employee Statistics Sample	A5TJRDIT.B100001C	Transformation	InformationMap
SALARY	A5TJRDIT.B200000R	Limit Exceeded	Employee Statistics Sample	A5TJRDIT.B100001C	Transformation	InformationMap

Update the Affected Tables

Specify the IMPACT_LIMIT= statement with an integer value to modify the tables. The integer “2” is specified, because based on the previous example, two table objects are affected. Also specify the REPORT statement.

```
proc metalib;
   OMR=(libid=AZ000009);
   impact_limit=2;
   report;
run;
```

The report shows that one table was updated and nine tables matched the data source.

Output 12.5 *Report of Changes When IMPACT_LIMIT=2 Is Specified*

The SAS System

The METALIB Procedure

Summary Report for Library AZ000009
Repository A0000001.A5TJRDIT
02FEB2011

Metadata Summary Statistics	
Total tables analyzed	10
Tables not processed due to large impact	0
Tables Updated	1
Tables Added	0
Tables matching data source	9
Tables not processed	0

Tables Updated

Table			Updates				
Metadata Name	Metadata ID	SAS Name	Metadata Name	Metadata ID	SAS Name	Metadata Type	Change
EMPLOYEE	A5TJRDIT.B200000K	EMPLOYEE	EMPNO	A5TJRDIT.B700002R	EMPNO	Column	IsDiscrete
			EMPNO	A5TJRDIT.BI000006	EMPNO	Index	Added
			EMPLOYEE.ic_id	A5TJRDIT.BH000006	ic_id	UniqueKey	Added

Example 5: Adding a Prefix to New Metadata Names

Features: PREFIX= statement

Details

To add a prefix to the name of a new metadata object during an update, specify the PREFIX statement.

In this example, the user runs an update on December 15, and wants to add that date to any new metadata object. A new table exists in the SAS library. In the data

source, the table has the SASTableName= value of **ABBA**. In the metadata, the table object is named **December15ABBA**.

Program

Submit the PREFIX statement with PROC METALIB. If any new metadata object is defined, the metadata name (the Name= attribute) begins with the specified prefix.

```
proc metalib;
    omr (library="MyTestLibrary");
       select (abba);
   prefix="December15";
   report;
run;
```

Output 12.6 *Prefix on Metadata Name Value*

The SAS System

The METALIB Procedure

Summary Report for Library MyTestLibrary
Repository Foundation
15DEC2010

Metadata Summary Statistics	
Total tables analyzed	1
Tables Updated	0
Tables Added	1
Tables matching data source	0
Tables not found	0
Tables not processed	0

Tables Added		
Metadata Name	**Metadata ID**	**SAS Name**
December15ABBA	A5JTOPDN.B4000011	ABBA

Example 6: Specifying a Folder for the Metadata

Features: FOLDER= statement

Details

These examples illustrate how PROC METALIB works with folders.

Create Metadata in the Registered Library Location

If a SAS library named baselib is defined in /My Folder/test, the following PROC METALIB statement registers the Class table in /My Folder/test. The FOLDER= statement is not necessary because the library is already located in /My Folder/test.

```
proc metalib;
   omr(library=baselib);
   select (class);
run;
```

Create Metadata in a Different Shared Folder

To create metadata for the Class table in a different folder, use the FOLDER= or FOLDERID= statement. The FOLDER= statement specifies a different folder location than the one in which the library is located. The folder must exist. PROC METALIB does not create it for you. There will be two table objects for the Class table in SAS library baselib. One definition is associated with the /My Folder/test location, and one definition is associated with the /Shared Data/Export location.

```
proc metalib;
   omr(library=baselib);
   folder="/Shared Data/Export";
   select (class);
run;
```

Create Metadata in a Personal Folder

If a library named oraclelib is defined in /Shared Data, the following request creates table objects in /My Folder/Oracle. This request assumes the /My Folder/Oracle folder already exists. PROC METALIB returns an error if a folder of the specified name cannot be found in the specified location.

```
proc metalib;
   omr(library=oraclelib);
   folder="/User Folders/MyUserId/My Folder/Oracle";
run;
```

Chapter 13

METAOPERATE Procedure

Overview: METAOPERATE Procedure

What Does the METAOPERATE Procedure Do?

The METAOPERATE procedure enables you to perform administrative tasks in batch mode that are associated with the SAS Metadata Server. PROC METAOPERATE performs the following tasks:

- delete, empty, or unregister a SAS Metadata Repository
- pause the metadata server to temporarily change it to a more restrictive state, and then resume it to the online state
- refresh the metadata server to do the following:
  - recover memory
  - reload authorization inheritance rules
  - enable or disable Application Response Measurement (ARM) logging
  - temporarily change the SAS Metadata Server's alert email system options and configuration options
  - execute an ad hoc server backup
  - change the metadata server's backup configuration
  - change the metadata server's backup schedule
  - recover the SAS Metadata Server from an earlier backup, and perform roll-forward recovery from the metadata server journal
  - terminate the recovery if you need to regain control of the metadata server during the recovery process
  - rebuild or restart the scheduler that executes the server backups
- stop or get the status of the metadata server

Beginning in SAS 9.4, the SAS Intelligence Platform supports single SAS Metadata Server configurations and clustered SAS Metadata Server configurations. PROC METAOPERATE has been modified to operate uniformly on all servers in a cluster by default when a clustered server configuration is detected. The only exceptions are when a recovery is performed and when the REORG option is selected to regain unused disk space in repository data sets during a metadata backup. Optional arguments are available to enable administrators to control client activity on specific servers within the cluster. Users of the single server configuration are unaffected by cluster-related defaults. For more information, see "Metadata Server Configurations and PROC METAOPERATE" on page 168.

The METADATA procedure performs some of the same tasks as PROC METAOPERATE. For more information, see "Comparison of the METADATA Procedure and the METAOPERATE Procedure" on page 84.

Syntax: METAOPERATE Procedure

Restriction: This procedure is not supported in SAS Viya.

PROC METAOPERATE <*server-connection-arguments*>
ACTION=PAUSE | REFRESH | RESUME | DELETE | EMPTY | STATUS | STOP | UNREGISTER
<NOAUTOPAUSE>
<NOCLUSTER>
<NOREDIRECT>
<OPTIONS="*XML-string*">
<OUT=*SAS-data-set*>;

Statement	Task
PROC METAOPERATE	Perform administrative tasks associated with a single SAS Metadata Server or metadata server cluster

PROC METAOPERATE Statement

Performs administrative tasks associated with a single SAS Metadata Server or metadata server cluster.

Syntax

PROC METAOPERATE ACTION=*value* <*options*>;

Summary of Optional Arguments

NOAUTOPAUSE
NOCLUSTER
NOREDIRECT
OPTIONS="*XML-string*"
OUT=*SAS-data-set*

Required Argument

ACTION=*value*

Value specifies the action that you want to perform.

DELETE

removes the specified repository, and removes the repository's registration from the repository manager. The repository is specified in the REPOSITORY= server connection argument or the METAREPOSITORY= system option. To invoke this action, the user must have appropriate permission to the repository, the repository must be registered in SAS Management Console as online, and the metadata server cannot be paused offline. For more information about the permissions needed to delete a repository, see *SAS Intelligence Platform: System Administration Guide*. The "NOAUTOPAUSE" argument is required.

In a clustered server configuration, the DELETE action is executed uniformly on all servers in the cluster.

EMPTY

removes the metadata records from the specified repository, but does not remove the repository's registration from the repository manager. The repository is specified in the REPOSITORY= server connection argument or the METAREPOSITORY= system option. To invoke this action, the user must have access permission to the repository, the repository must be registered in SAS Management Console as online, and the metadata server cannot be paused offline. The "NOAUTOPAUSE" argument is required. For more information, see "Example 5: Delete All Metadata Records from a Repository" on page 181.

In a clustered server configuration, the EMPTY action is executed uniformly on all servers in the cluster.

PAUSE

The Pause action has two uses:

- The primary use is to downgrade the availability of the SAS Metadata Server configuration to a specified state. The state is specified in an XML "<SERVER STATE="ADMIN | OFFLINE | READONLY"" element in the OPTIONS= argument. For information about the supported state changes, see "How PAUSE, RESUME, and REFRESH Affect Repositories" on page 170. If the <SERVER STATE="*value*"/> XML element is omitted, the server is paused to an offline state. In an offline state, the server process continues to exist, but it does not accept client requests, except for some administrative requests. You might find it more beneficial to downgrade the server to an ADMIN or a READONLY state.

  An XML "<PAUSECOMMENT>text</PAUSECOMMENT>" element can also be specified in the OPTIONS= argument. <PAUSECOMMENT> enables you to submit free-form text with the PAUSE action to post a reason for the service interruption. This text is inserted into a holding area and is displayed to all SAS Metadata Server callers until the server is resumed and the text is cleared from the holding area with the RESUME action. For more information, see "Example 2: Pause and Resume the SAS Metadata Server" on page 178.

- The second reason to use ACTION=PAUSE is to regain control of the metadata server in the event that the backup recovery process stops responding. Specify the "<FORCE/>" XML element in the OPTIONS= argument with <SERVER STATE="ADMIN"/> and ACTION=PAUSE to

regain control of the server. The server is paused to an offline state if you omit <SERVER STATE="ADMIN"/>.

In a clustered metadata server configuration, the Pause action is executed uniformly on all servers in the cluster.

Before using the <FORCE/> option with Pause, see “Recovery in a Clustered Server Configuration” on page 173.

REFRESH

affects the metadata server configuration differently depending on the XML elements that you specify in the OPTIONS= argument. Here are the choices:

- If you specify REFRESH without an XML element or if you specify the <SERVER/> XML element, the REFRESH action pauses and resumes the metadata server configuration (in a single step). Do not specify the STATE= parameter in the <SERVER/> XML element. The REFRESH action recovers memory on the entire metadata server configuration, and reloads authorization inheritance rules. For more information, see “Example 4: Recover Memory on the Metadata Server” on page 180. After the refresh, all repositories return to their registered access mode. For more information, see “How PAUSE, RESUME, and REFRESH Affect Repositories” on page 170.
- With the “<ARM parameter-name="value"/>” XML element specified, the REFRESH action enables or disables ARM logging, and specifies a pathname for the ARM log. For more information, see “Example 3: Enable ARM Logging” on page 179.
- With the “<OMA ALERTEMAILTEST="text"/>” XML element specified, the REFRESH action sends a test alert email message to the address configured in the <OMA ALERTEMAIL="*email-address*"/> option in the omaconfig.xml configuration file. If the test message is not received, you can specify one or more of the following XML elements. These XML elements can be used to change the alert email settings until an alert email test is successful:“<OMA ALERTEMAIL="email-address(es)"/>”, “<OMA EMAILAUTHPROTOCOL="NONE | LOGIN"”, “<OMA EMAILHOST="server-network-address"/>”, “<OMA EMAILID="server-email-address"/>”, “<OMA EMAILPW="password"/>”, and “<OMA EMAILPORT="port-number"/>”. For more information, see “Using Alert Email XML Elements” on page 175.
- With the “<OMA JOURNALPATH="filename"/>” XML element specified, the REFRESH action specifies a new filename for metadata server journaling.

  Note: This option is valid only when <OMA JOURNALTYPE="SINGLE"/> is specified in the server’s omaconfig.xml configuration file. Use of the <OMA JOURNALTYPE=”SINGLE”/> option is discouraged because it disables recovery roll-forward processing.

- When used with backup-related XML elements, REFRESH enables you to execute ad hoc backups and modify the default SAS Metadata Server backup configuration and backup schedule. It also recovers the metadata server from a previous backup, and rebuilds or restarts the backup scheduler thread. The following options are available:“<BACKUP *attributes*/>” on page 161, “<BACKUPCONFIGURATION *attribute(s)*/> ” on page 161, “<RECOVER BACKUPNAME="*name*" | BACKUPPATH="*pathname*" *options*/>” on page 163, “<SCHEDULE

EVENT="Backup" WEEKDAY*n*="*time*<R>"/>" on page 165, and "<SCHEDULER/>" on page 165. For more information, see "Using Backup and Recover Options" on page 171, "Recovery in a Clustered Server Configuration" on page 173, and "Example 7: Execute Backup and Recover Options in PROC METAOPERATE" on page 183.

For information about how Refresh options are processed in a clustered metadata server configuration, see "Metadata Server Configurations and PROC METAOPERATE" on page 168.

RESUME

returns all servers in the metadata server configuration to the online state and metadata repositories to their registered state before the Pause. For more information, see "How PAUSE, RESUME, and REFRESH Affect Repositories" on page 170 "Example 2: Pause and Resume the SAS Metadata Server" on page 178.

Any text that was specified in the "<PAUSECOMMENT>text</PAUSECOMMENT>" XML element by the PAUSE action is cleared.

In a clustered metadata server configuration, the Resume action is executed uniformly on all servers in the cluster. For more information about SAS Metadata Server configurations, see "Metadata Server Configurations and PROC METAOPERATE" on page 168.

You can specify the "<FORCE/>" XML element in the OPTIONS= argument when using ACTION=RESUME. <FORCE/> has one use: to regain control of the metadata server configuration during the backup recovery process in the event that the recovery process stops responding . However, when <FORCE/> is used with ACTION=RESUME, the server is returned to an online state, when it might be safer to return to an administrator-only state. The <SERVER/> XML element is required when <FORCE/> is used.

Before using the <FORCE/> option with RESUME, see "Using Backup and Recover Options" on page 171 and "Recovery in a Clustered Server Configuration" on page 173.

STATUS

returns information about the access state and software properties of the SAS Metadata Server. The properties include the SAS software version, release number, host operating environment, SAS Metadata Model version number, and the user ID that started the metadata server. The access state indicates whether the metadata server is paused or running. For an example of how the Status action is specified, see "Example 1: Get the SAS Metadata Server's Status with PROC METAOPERATE" on page 176.

In a clustered server configuration, status queries are routed to a slave server that is chosen by the cluster's load balancer. To get status information for a specific server in the cluster, specify that server node in the connection options and specify the NOREDIRECT argument with ACTION=STATUS.

> **TIP** The PROC METAOPERATE STATUS action gets basic information about the metadata server's availability. You can obtain more detailed information by using PROC METADATA with METHOD=STATUS. For more information, see Chapter 11, "METADATA Procedure," on page 87.

STOP
halts client activity on the metadata server configuration and terminates the metadata server process(es). In complex environments, the metadata server shutdown can take a few minutes. Therefore, PROC METAOPERATE might finish executing before the metadata server configuration finishes its shutdown. Metadata in repositories is unavailable until the metadata server configuration is restarted. You cannot restart a metadata server configuration with PROC METAOPERATE.

In a clustered metadata server configuration, the Stop action is executed uniformly on all servers in the cluster. To stop a specific server in the cluster, specify that server in the connection options and specify the NOCLUSTER argument with ACTION=STOP. For more information, see “Metadata Server Configurations and PROC METAOPERATE” on page 168.

UNREGISTER
removes the repository's registration from the repository manager, but does not remove the metadata records from the repository, and does not remove the repository from disk. The repository is specified in the REPOSITORY= server connection argument or the METAREPOSITORY= system option. To invoke this action, the user must have access permission to the repository, the repository must be registered in SAS Management Console as online, and the metadata server cannot be paused offline. The “NOAUTOPAUSE” argument is required.

In a clustered server configuration, it is recommended that you change a repository to an offline state to make it unavailable instead of using UNREGISTER.

Requirement You must have the appropriate SAS Administrator role on the metadata server to execute all actions except STATUS.

Tips Specifying more than one XML element in a PROC METAOPERATE statement might cause unwanted results. Use more than one XML element only when specified in the documentation.

If you use PROC METAOPERATE to delete, empty, or unregister a project repository, you must first make sure that no metadata is checked out to that project repository. See SAS Data Integration Studio documentation for information about unlocking any checked out objects. Or, you can use SAS Management Console to delete, empty, or unregister a project repository. SAS Management Console unlocks any checked-out objects before it performs the action.

Optional Arguments

NOAUTOPAUSE
NOAUTOPAUSE is required for the DELETE, EMPTY, and UNREGISTER actions. It is required when the REFRESH action is specified with the “<BACKUP attributes/>”, “<BACKUPCONFIGURATION attribute(s)/> ”, “<RECOVER BACKUPNAME="name" | BACKUPPATH="pathname" options/>”, “<SCHEDULE EVENT="Backup" WEEKDAYn="time<R>"/>”, and “<SCHEDULER/>” XML elements in the OPTIONS= parameter. It is recommended with the “<OMA ALERTEMAILTEST="text"/>” option. NOAUTOPAUSE omits the automatic pause and resume of the metadata server

when PROC METAOPERATE passes an action to the metadata server. Without NOAUTOPAUSE, all repositories experience the implicit pause and resume whenever an action is passed to the server. This can be unwanted because it makes all repositories temporarily unavailable.

NOCLUSTER
NOCLUSTER is an option that is used with the STOP action in a clustered metadata server configuration. In a single metadata server configuration, NOCLUSTER is ignored. NOCLUSTER turns off the default behavior that causes the STOP action to be executed on all servers in the cluster. It causes the STOP action to be executed on the connected metadata server only. For more information, see "Metadata Server Configurations and PROC METAOPERATE" on page 168. When NOCLUSTER is specified, NOREDIRECT is executed behind the scenes.

NOREDIRECT
NOREDIRECT is an option that is used with the STATUS action in a clustered metadata server configuration. In a single metadata server configuration, NOREDIRECT is ignored. NOREDIRECT temporarily overrides the cluster's load balancer so that the status request can be executed only on the metadata server in the connection options. Use NOREDIRECT with ACTION=STATUS when you want to get status information about a specific slave server or the master server. For more information, see "Metadata Server Configurations and PROC METAOPERATE" on page 168.

OPTIONS="*XML-string*"
specifies a quoted string that contains one or more XML elements. Some of the XML elements specify additional parameters for the actions. The OPTIONS= argument is required for some actions.

Note: To ensure that the XML string is parsed correctly by the metadata server, you must indicate that quotation marks within the XML element are characters. You can nest single and double quotation marks, or double and double-double quotation marks as follows: `options='<ARM ARMSUBSYS="(ARM_OMA)" ARMLOC="myfileref"/>' options="<ARM ARMSUBSYS=""(ARM_OMA)"" ARMLOC=""myfileref""/>"`

The XML elements include the following:

<ARM *parameter-name*="*value*"/>
is one or more <ARM/> XML elements that specify system options to enable or disable ARM logging. REFRESH is the most appropriate action to specify the <ARM/> XML element, but PAUSE and RESUME actions can specify it. If the metadata server is refreshed or stopped and restarted, ARM parameters return to the values in the configuration file. For more information, see "Example 3: Enable ARM Logging" on page 179 . Also see information about ARM logging in *SAS Intelligence Platform: System Administration Guide* and the ARMSUBSYS= and ARMLOC= system options in *SAS Interface to Application Response Measurement (ARM): Reference*. An <ARM/> element can include the following parameters:

ARMSUBSYS="(| OMA)"
enables and disables ARM logging.

ARMLOC="*fileref* | *filename*"
specifies a location to which to write the ARM log. If ARM logging is already enabled, specifying ARMLOC= writes the ARM log to a new location. Relative and absolute pathnames are read as different locations.

<BACKUP *attributes*/>
supported with the REFRESH action, invokes an ad hoc backup of the metadata server to the location specified in the server's backup configuration. The backup is named with a modified date-and-time stamp in ISO 8601 format. For more information, see "Using Backup and Recover Options" on page 171.

Optional attributes are the following:

COMMENT="*text*"
accepts a user-specified text string of unlimited length to describe the reason for the ad hoc backup. This comment is recorded as part of the backup history. The backup history is visible in the Server Backup node of the SAS Management Console Metadata Manager, or it can be requested with PROC METADATA. For more information, see Chapter 11, "METADATA Procedure," on page 87.

REORG="Y | N"
specifies whether repository data sets should be rebuilt to release unused disk space before the backup. The default value is N (No).

CAUTION
The REORG option is not recommended for ad hoc backups because it interrupts the operation of the metadata server. For more information, see "<SCHEDULE EVENT="Backup" WEEKDAY*n*="*time*<R>"/>" on page 165.

Note: The REORG option is ignored in a running clustered metadata server configuration. The REORG option must be run on a metadata server that is running in stand-alone mode. For more information, see "Using Backup and Recover Options" on page 171.

<BACKUPCONFIGURATION *attribute(s)*/>
supported with the REFRESH action, modifies the value of the specified backup configuration attribute. Backup configuration attributes include:

BackupLocation="*directory*"
specifies the directory in which to write the metadata server backups. In a single SAS Metadata Server configuration, the default location is a `Backups` subdirectory of the `SASMeta/MetadataServer` directory. To create the directory in a location other than `MetadataServer` or on a different drive, specify an absolute pathname that is meaningful to the computer that hosts the metadata server. If the specified directory does not exist, the metadata server creates it for you. In a clustered SAS Metadata Server configuration, the backup location is a shared location that is configured at installation.

RunScheduledBackups="Y | N"
controls the backup scheduler. A value of "Y" enables scheduled backups. A value of "N" disables them.

DaysToRetainBackups="*number*"
specifies the number of days to keep backups before they are deleted from the backup location. The default value is "7". To never remove any backups, specify "0" in this attribute. A value of "0" is not advisable except as a temporary setting.

<FORCE/>
supported with the PAUSE or RESUME actions, regains control of the metadata server during the recovery process in the event that the recovery process stops responding. When used with RESUME, <FORCE/> returns the server to an online state, where it is available to clients. When used with PAUSE, you have the option to specify "<SERVER STATE="ADMIN | OFFLINE | READONLY"" to return the server to the ADMIN state. In the ADMIN state, you can examine the server for problems before making it available to clients.

<OMA ALERTEMAIL="*email-address(es)*"/>
supported with the REFRESH action, temporarily modifies the email addresses to which the SAS Metadata Server sends an alert email message. This option is permanently configured in the omaconfig.xml file, but it can be temporarily modified for the duration of the server session with the REFRESH action. The configured email recipients can be viewed on the **General** tab of the active server's Properties window in SAS Management Console. Or, configured email recipients can be listed by issuing the omaconfig.xml option in PROC METADATA with METHOD=STATUS. To specify more than one email address, enclose each address in single quotation marks, place a blank space between each address, and enclose the list in parentheses: for example, "('Bill@mycompany.com' 'Susan@mycompany.com')". For more information, see "Using Alert Email XML Elements" on page 175. The NOAUTOPAUSE argument is recommended with this action.

<OMA ALERTEMAILTEST="*text*"/>
supported with the REFRESH action, sends a test alert email message to the address(es) configured in the <OMA ALERTEMAIL="*email-address*"/> option. This option is permanently configured in the omaconfig.xml configuration file, but can be temporarily modified for the duration of the server session with the REFRESH action. The test email messages are sent to both the permanent addresses in the omaconfig.xml file and any temporary addresses specified with the REFRESH action. For more information, see "Using Alert Email XML Elements" on page 175.

<OMA EMAILAUTHPROTOCOL="NONE | LOGIN"
supported with the REFRESH action, temporarily changes the authentication protocol for SMTP email that is sent by the SAS Metadata Server. When you specify the value "LOGIN", you also need to specify EMAILID and EMAILPW. The value "NONE" specifies that no authentication protocol is used. This option is permanently configured in the sasv9.cfg file, but it can be temporarily modified for the duration of the server session with the REFRESH action. For more information, see "Using Alert Email XML Elements" on page 175.

<OMA EMAILHOST="*server-network-address*"/>
supported with the REFRESH action, temporarily changes the network address of the enterprise's SMTP server (for example, mailhost.company.com). This option is permanently configured in the sasv9.cfg file, but it can be temporarily modified for the duration of the server session with the REFRESH action. For more information, see "Using Alert Email XML Elements" on page 175.

<OMA EMAILID="*server-email-address*"/>
supported with the REFRESH action, temporarily changes the email address for the FROM field of alert email messages that are sent by the SAS Metadata Server. The email address can be entered in either of the following forms: "server-name<user-account@domain>" or "<user-account@domain>". This option is permanently configured in the sasv9.cfg file, but it can be

temporarily modified for the duration of the server session with the REFRESH action. For more information, see “Using Alert Email XML Elements” on page 175.

<OMA EMAILPW="*password*"/>
supported with the REFRESH action, specifies the logon password to be used with the email address that you specified in the EMAILID option. The password should be encoded with PROC PWENCODE. For more information about PROC PWENCODE, see the *Base SAS Procedures Guide*. You should use SAS002 as the minimum encryption level. SAS002 is the standard SAS proprietary encryption level that is available with PROC PWENCODE. To specify a higher encryption level, you must have SAS/SECURE software enabled.

<OMA EMAILPORT="*port-number*"/>
supported with the REFRESH action, temporarily changes the port number that is used by the SMTP server that you specified in the EMAILHOST option. This option is permanently configured in the sasv9.cfg file, but it can be temporarily modified for the duration of the server session with the REFRESH action. For more information, see “Using Alert Email XML Elements” on page 175.

<OMA JOURNALPATH="*filename*"/>
supported with the REFRESH action, stops writing journal entries to the metadata server journal file in the current location, and resumes writing journal entries in a new journal file in the specified physical location. This option is valid only when the metadata server is configured with <OMA JOURNALTYPE="SINGLE"/> in the omaconfig.xml configuration file. The default configuration setting is <OMA JOURNALTYPE="ROLL_FORWARD"/>, which supports roll-forward recovery of the server.

<PAUSECOMMENT>*text*</PAUSECOMMENT>
supported with the PAUSE action, enables you to submit free-form text (for example, the reason for the pause) into a holding area that will be displayed to all SAS Metadata Server callers. The text is cleared from the holding area when the server is resumed with the RESUME action. Quotation marks are optional around the text.For more information, see “Example 2: Pause and Resume the SAS Metadata Server” on page 178.

<RECOVER BACKUPNAME="*name*" | BACKUPPATH="*pathname*" *options*/>
supported with the REFRESH action, recovers the metadata server from the backup specified in the BACKUPNAME="*name*" or BACKUPPATH="*pathname*" attribute with the specified options.

BACKUPNAME="*name*"
specifies the name of a backup. Server backups are named with a modified date-and-time stamp. For information about backup names, see “Using Backup and Recover Options” on page 171. The server looks for backups in the backup location specified in the current configuration. To use a backup from a different directory, either use the BACKUPPATH= attribute instead of BACKUPNAME=, or specify the BACKUPLOCATION= attribute with the BACKUPNAME= attribute. The default backup location is the `Backups` subdirectory of the `SASMeta/MetadataServer` configuration directory.

BACKUPPATH="*pathname*"
specifies the absolute pathname to the backup. This option is useful when the backup is located in a different directory or drive from the backup location specified in the current configuration.

Optional recovery attributes are as follows:

BACKUPLOCATION="*directory*"
specifies the name of the directory that contains the backup specified in the BACKUPNAME= attribute, if the directory differs from the backup directory specified in the current configuration. The name that you specify is considered to be relative to the `SASMeta/MetadataServer` directory.

COMMENT="*text*"
specifies a user-defined text string to record an explanation for the recovery. The text string is displayed in the backup history.

PAUSECOMMENT="*text*"
specifies a user-defined text string that will be displayed as a recovery notification to clients.

INCLUDEALLCONFIGFILES=" Y | N"
specifies whether to replace configuration files in the server directory with the configuration files that are in the directory when the backup occurs. The default value is N (No).

CAUTION
When INCLUDEALLCONFIGFILES="Y", the server.pid file, any recent changes to the omaconfig.xml file, and files that were added to the configuration directory after the last backup will be lost. The recovery overwrites all files in the `SASMeta/MetadataServer` directory, except the backup history, backup configuration, and manifests. It replaces them with the configuration files in the backup. This option should never be set in a clustered server configuration. The server.pid file from the backup might have a different PID number than the current server.

ROLLFORWARD="blank | _ALL_ | *datetime*"
specifies whether the metadata server journal should be used to apply changes that were made to the server after the backup was taken, and whether to recover all changes from the journal or only changes up to a specified point in time.

Omitting this attribute, or specifying it with a blank value specifies not to recover changes from the journal.

ALL
recovers all changes from the journal.

datetime
recovers changes from the journal up to a specified point in time. The metadata server log displays changes in server local time. The ROLLFORWARD= attribute requires input in GMT time. See "Using Backup and Recover Options" on page 171 for information to convert server local time values to GMT time.

Note: The <RECOVER/> option is available only on a SAS Metadata Server that is running in stand-alone mode. The option is ignored when it is encountered in a running clustered server configuration. For more

information, see “Recovery in a Clustered Server Configuration” on page 173.

<SCHEDULE EVENT="Backup" WEEKDAY*n*="*time*<R>"/>
supported with the REFRESH action, modifies the server backup schedule.

EVENT="Backup"
specifies the event to be scheduled. The valid value is "Backup". The Event= attribute is required.

WEEKDAY*n*="*time*”
specifies the backup schedule. The metadata server supports daily backups, specified in a weekly schedule where the attribute WeekDay1= is Sunday, the attribute WeekDay7= is Saturday, and appropriately numbered WeekDay*n*= attributes represent the other days of the week. Backup times are specified in four-digit values based on a 24-hour clock: for example, 0100 is 1 a.m. and 1300 is 1 p.m. The <SCHEDULE EVENT="Backup"/> option accepts input in server local time and applies values in server local time. The default backup schedule specifies backups Monday through Saturday at 1 a.m. (Deployment backups occur at 1 a.m. on Sundays.) To change the schedule, specify the appropriate WeekDay*n*= attribute with the backup time. The new time overwrites the old time. To schedule more than one backup in a day, separate the time values with semicolons: for example, “0100; 1300”. To remove all backups from a day, specify an empty string.

R
specifies to perform a REORG with a backup. REORG releases unused disk space from metadata repository data sets before a backup. The operation is necessary for repository maintenance, but it is time-intensive and causes the metadata server to be paused. It should not be performed often.

In a single SAS Metadata Server configuration, the default backup schedule performs REORG on Monday mornings (WeekDay2="0100R"). The REORG option is ignored in a clustered metadata server configuration. To run a backup with REORG in a clustered metadata server configuration, the administrator must stop the cluster and execute the backup on a server that is running in stand-alone mode. For more information, see “Changing a Clustered Server to Stand-Alone Mode” on page 174.

<SCHEDULER/>
supported with the REFRESH action, rebuilds or restarts the backup scheduler, depending on the XML subelement that is specified in the <SCHEDULER/> element. The backup scheduler runs continuously from the time the metadata server is started, and buffers the schedule in 48-hour increments. The <SCHEDULER/> XML element restores the scheduler in the event the scheduler is inoperative and backups are not taking place on the specified schedule. You can issue a STATUS method request through PROC METADATA to check the health of the scheduler. For more information, see “Example 5: Get Server Backup Information with PROC METADATA” on page 111. The supported subelements are:

<REBUILD/>
forces the scheduler to rebuild its in-memory linked list of events.

<RESTART/>
causes the current scheduler thread to stop, and then starts a new one.

<SERVER STATE="ADMIN | OFFLINE | READONLY"
supported with the PAUSE and RESUME actions, the XML element has the following uses:

- With PAUSE, it is optional and specifies an access state to apply to the metadata server. If you do not specify the <SERVER STATE="ADMIN | OFFLINE | READONLY"/> XML element, or if you specify <SERVER/> without a STATE= parameter, the default behavior is to pause the metadata server to an offline state. When the server is paused to an offline state, all repositories are also paused to an offline state.

  STATE= enables you to specify these alternative values:

ADMIN
allows only users with administrative status to read and write metadata on the metadata server.

READONLY
allows Read-Only access to all users.

- With RESUME, the XML element specifies that the action applies to the metadata server. It is specified without the STATE= parameter (SERVER/) when the <FORCE/> option is used.

OUT=*SAS-data-set*
names the output data set. This argument is used with the STATUS action. Other actions do not create output.

Server Connection Arguments

Server connection arguments establish communication with the metadata server. If you omit these arguments, then the values of the metadata system options are used or the values can be obtained interactively. Server connections made with server connection arguments are not redirected to another server in the cluster when the default server connection profile is used. If the server specified in the server connection arguments is not available, then the connection will fail. If connection redirection is important for the request, use metadata system options to establish communication with the server instead. For more information, see “Connection Options ” on page 38.

PASSWORD="*password*"
is the password for the authenticated user ID on the metadata server. If you do not specify PASSWORD=, the value of the METAPASS= system option is used. For more information, see “METAPASS= System Option” on page 49. The maximum length is 512 characters.

Alias METAPASS= or PW=

PORT=*number*
is the TCP port that the metadata server listens to for connections. This port number started the metadata server. If you do not specify PORT=, the value of the METAPORT= system option is used. For more information, see “METAPORT= System Option” on page 50. The range of allowed port numbers is 1 to 65535. The metadata server is configured with a default port number of 8561.

Alias METAPORT=

Requirement Do not enclose the value in quotation marks.

PROTOCOL=BRIDGE
is the network protocol for connecting to the metadata server. If you do not specify PROTOCOL=, the value of the METAPROTOCOL= system option is used. For more information, see “METAPROTOCOL= System Option” on page 52. In this release, the only supported value is BRIDGE, which specifies the SAS Bridge protocol. This is the server default, so there is no need to specify this argument.

Alias METAPROTOCOL=

Requirement Do not enclose the value in quotation marks.

REPOSITORY="*name*"
is the name of an existing repository. This value is the repository's Name= parameter. The REPOSITORY= argument is required when the action is UNREGISTER, DELETE, or EMPTY. For other actions, if you do not specify REPOSITORY=, the value of the METAREPOSITORY= system option is used. For more information, see “METAREPOSITORY= System Option” on page 53. The default for the METAREPOSITORY= system option is FOUNDATION. The maximum length is 32,000 characters.

Alias METAREPOSITORY= or REPOS=

SERVER="*host-name*"
is the host name or network IP address of the computer that hosts the metadata server. The value LOCALHOST can be used if the SAS session is connecting to the metadata server on the same computer. If you do not specify SERVER=, the value of the METASERVER= system option is used. For more information, see “METASERVER= System Option” on page 54. The maximum length is 256 characters.

Alias HOST= or IPADDR= or METASERVER=

USER="*authenticated-user-ID*"
is an authenticated user ID on the metadata server. The metadata server supports several authentication providers. For more information about controlling user access to the metadata server, see the *SAS Intelligence Platform: Security Administration Guide*. If you do not specify USER=, the value of the METAUSER= system option is used. For more information, see “METAUSER= System Option” on page 57. The maximum length is 256 characters.

Alias ID= or METAUSER= or USERID=

Usage: METAOPERATE Procedure

How PROC METAOPERATE Works

The administrator of the metadata server configuration can perform four types of maintenance with PROC METAOPERATE.

- Control the availability of the metadata server by calling methods in the IServer server interface of SAS Open Metadata Architecture. Use PAUSE, REFRESH, RESUME, STATUS, and STOP.
- Destroy or otherwise permanently remove a repository by calling methods in the IOMI server interface of SAS Open Metadata Architecture. Use DELETE, EMPTY, and UNREGISTER.
- Manage server backups. Use REFRESH.
- Test and temporarily reconfigure the SAS Metadata Server's alert email notification subsystem. Use REFRESH.

Metadata Server Configurations and PROC METAOPERATE

Beginning in SAS 9.4, the SAS Intelligence Platform supports single SAS Metadata Server configurations and clustered SAS Metadata Server configurations.

- In a single metadata server configuration, all metadata requests are received and processed by a single SAS Metadata Server. If this metadata server ceases to function, metadata processing is unavailable until the metadata server can be recovered.
- In a clustered metadata server configuration, three or more identical metadata servers are linked in such a way that provides metadata redundancy. All of the metadata servers are available for processing requests. If one server becomes unavailable, its load is transferred to another server.

To ensure consistency among the servers in a clustered metadata server configuration, PROC METAOPERATE behaves as follows:

- PROC METAOPERATE cannot be used to configure a metadata server cluster or to permanently add or remove servers from the clustered metadata server configuration. A metadata server cluster is configured with the SAS Deployment Wizard at installation. Any additional servers must be added to the cluster with the SAS Deployment Wizard. Servers are removed from the cluster using the SAS Deployment Manager.

- PROC METAOPERATE controls the availability of the cluster of servers as a whole. The server control actions (for example, PAUSE, RESUME, and STOP) pause, resume, and stop all servers in the cluster by default.
- A NOCLUSTER argument is supported with the STOP action to enable an administrator to stop an individual server in the cluster if it needs maintenance. When NOCLUSTER is specified, the STOP action is executed only on the server specified in the connection options. A server cannot be started or restarted with PROC METAOPERATE. For information about how to start a server, see the *SAS Intelligence Platform: System Administration Guide*.

  For an example of how the NOCLUSTER argument is specified, see "Example 8: Stop One Server in a Metadata Server Cluster" on page 185.

  The cluster remains available as long as it has the quorum. The quorum requires that at least half of the servers that are defined in the cluster be available to continue operating. If exactly half of the servers are available, then the server defined as Node 1 must be among those that are operating. When a quorum is lost, the cluster makes metadata unavailable to clients until enough servers are recovered and the quorum is regained.

CAUTION
Before stopping a server node with NOCLUSTER, make sure you have enough remaining nodes to maintain a quorum. When too many servers become unavailable in the cluster, the quorum could be lost.

- In a cluster, a load balancer controls server connections. The STATUS action gets information about whatever server received the request by default. The server node that you specify in the connection options is unlikely to be the node that processes the request. You can direct the STATUS action to a specific server node by specifying the NOREDIRECT argument. When NOREDIRECT is specified, the request is processed by the server identified in the connection options.

TIP The PROC METAOPERATE STATUS action returns basic information about a metadata server's availability. It does not get any information about the server cluster or any given server node's role in the cluster. You can obtain cluster information by using PROC METADATA with METHOD=STATUS. For more information, see "Metadata Server Configurations and PROC METADATA" on page 96.

Here are some other things to know about how PROC METAOPERATE actions and options operate in a clustered metadata server configuration:

- ACTION=REFRESH options that recover memory on the server and change alert email settings or the alert email server configuration are executed uniformly on all servers in the cluster. Modifications to settings are for the duration of the cluster session. To make the changes permanent, you must modify all of the servers' sasv9.cfg and omaconfig.xml files.
- Repository control actions (for example, EMPTY, DELETE, and UNREGISTER) execute uniformly on all servers in the cluster.
- The server nodes in the cluster have a master-slave relationship. ACTION=REFRESH options that request server backups back up the master server. The backups are used by the master server to recover the server nodes as needed, and they can be used by administrators to return the master server

to a point in time before a transaction occurred that corrupted repository metadata. For more information, see “Recovery in a Clustered Server Configuration” on page 173.

- ACTION=REFRESH options that modify the server backup configuration and backup schedule store the information in a central location. As a result, the information is available to all of the server nodes for processing queries. Any server node can take over the backup task should it become the master server. For information about these options, see “Using Backup and Recover Options” on page 171.
- ACTION=REFRESH options that enable or disable ARM logging are executed on the master server.
- The ACTION=REFRESH <RECOVER/> option is available only on a server that is running in stand-alone mode. For more information, see “Recovery in a Clustered Server Configuration” on page 173.
- Backups that specify the REORG option must be executed on a SAS Metadata Server that is running in stand-alone mode.

For more information about administering a clustered metadata server configuration, see the *SAS Intelligence Platform: System Administration Guide*.

Users of the single metadata server configuration are not affected by cluster-related defaults. If cluster-related arguments are specified in a single metadata server configuration, they are ignored or processed as appropriate for the single metadata server configuration.

How PAUSE, RESUME, and REFRESH Affect Repositories

The PAUSE and RESUME actions enable you to temporarily downgrade the availability of all repositories on a metadata server by specifying a state change for the server. PAUSE invokes the more restrictive server state. RESUME returns the server to an ONLINE state.

Important things to know about PAUSE are:

- You cannot change the state of a specific repository with PAUSE. PAUSE affects all repositories on the server.
- You can never increase the availability of a repository with PAUSE. PAUSE changes only a repository’s availability if the specified server state is more restrictive than the repository’s registered access mode.
- A repository’s registered access mode cannot be permanently modified with PROC METAOPERATE; the downgrade is temporary. You must use SAS Management Console to permanently modify a repository’s registered access mode. As an alternative, you can modify the value of a repository’s Access= attribute by issuing an UpdateMetadata request with PROC METADATA.

An important thing to know about RESUME is:

- A repository’s registered access mode is its persisted state. When RESUME returns the server to an ONLINE state, each repository is returned to its registered access mode.

A metadata repository's availability during a PAUSE is reported in its PauseState attribute. The PauseState= value is computed from both the metadata server state and the repository's registered access mode.

The following grid shows how a repository's PauseState= value is computed from the repository's access mode (the rows) and the metadata server's state (the columns). For example, a repository with a registered Read-Only access mode and an Admin server state has an admin(readonly) pause state.

Table 13.1 *How the Server State Affects the Repository State*

Registered Access Mode	Online Server State	Admin Server State	Read-Only Server State	Offline Server State
online	online	admin	read-only	offline
read-only	read-only	admin(readonly)	read-only	offline
administration	admin	admin	admin(readonly)	offline
offline	offline	offline	offline	offline

Notice in the grid that when you use PROC METAOPERATE to pause the metadata server to an OFFLINE state (which is the default), the repositories are set to an OFFLINE state. This is because OFFLINE is the most restrictive state.

The REFRESH action is equivalent to a PAUSE to OFFLINE that is immediately followed by a RESUME. You can REFRESH a specific repository or the server as a whole. For more information about the tasks that require PAUSE, REFRESH, or RESUME actions, see *SAS Intelligence Platform: System Administration Guide*.

Using Backup and Recover Options

A SAS Metadata Server has the ability to back up and recover itself. Backups are initiated by a dedicated scheduler thread that is started when the metadata server is started. Backups are executed in a dedicated backup thread that is started as needed, so that backups do not interrupt the regular operation of the metadata server. When a server recovery is requested, the recovery process is executed in the backup thread.

The SAS Metadata Server is configured with a default backup configuration and backup schedule by the SAS Deployment Wizard. The default backup schedule performs backups Monday through Saturday at 1 a.m. (Deployment backups occur at 1 a.m. on Sundays.) Backups are retained for seven days, and run unassisted unless you modify the default backup schedule. In a single SAS Metadata Server configuration, the Monday morning backup includes a REORG process that releases unused disk space from repository data sets. The REORG process temporarily pauses the server to a read-only state. Therefore, it needs to run when server activity is low. The REORG option is ignored in the clustered server configuration. In order to run a backup with REORG in a clustered server configuration, an administrator must stop the cluster and execute the backup on a server that is running in stand-alone mode. For more information, see "Changing a Clustered Server to Stand-Alone Mode" on page 174.

You can modify the default backup configuration and backup schedule in SAS Management Console by opening the Server Backup node in the SAS Management Console Metadata Manager (this is the recommended method). Or you can modify the backup configuration and backup schedule by using PROC METAOPERATE. You can also perform ad hoc backups and request a recovery using both tools.

PROC METAOPERATE supports two options that are not supported in SAS Management Console:

- an option to rebuild or restart the scheduler thread in case backups are not occurring as scheduled. See ACTION="REFRESH" on page 157 and "<SCHEDULER/>" on page 165.
- an option to interrupt the recovery process in the event that it stops responding. See ACTION="PAUSE" on page 156, ACTION="RESUME" on page 158, and "<FORCE/>" on page 162.

For more information about performing metadata server backups, including integrating server backups with other SAS backups, see the *SAS Intelligence Platform: System Administration Guide*.

The recommended tool for performing metadata server recoveries is SAS Management Console. The recovery feature enables you to restore the metadata server using a specified backup and supports a ROLL_FORWARD= attribute that enables you to request roll-forward recovery to a specified datetime value from the metadata server journal file.

The metadata server log records datetime values in server local time. The ROLL_FORWARD= attribute of the PROC METAOPERATE <RECOVER BACKUPNAME="*name*" | BACKUPPATH="*pathname*"/> option requires input in GMT time. Backup names contain information that you can use to convert server local time values to GMT time values. Backups are named with a date-and-time stamp in modified ISO 8601 format. The ISO 8601 format is a server local datetime value that includes the GMT offset at the end of the string. The server modifies the format in that it changes colons between time values to underscores. For example, consider the backup name:

```
2010-09-20T00_59_59-04_00
```

The numbers preceding the T are the date: September 20, 2010. The numbers immediately following the T are the server local time (00_59_59). The -04_00 at the end of the time is the GMT offset. In this case, the backup was made just before 1 a.m. server local time. The minus offset indicates that the time value is four hours less than GMT. A time zone that is greater than GMT has a plus offset (+07_00). Use the GMT offset in your backup names to determine how you need to adjust the input value from the server log.

> **TIP** To avoid having to make the time conversion, you can use SAS Management Console to perform recoveries. The SAS Management Console Server Backup Recovery window enables you to specify the roll-forward value in server local or GMT time.

The following table summarizes the backup-related options that are supported by PROC METAOPERATE:

Table 13.2 *Summary of Backup and Recover Tasks and Options*

Task	ACTION=	OPTION=
change the default backup location, retention policy, or turn off scheduled backups	REFRESH	<BACKUPCONFIGURATION *attribute(s)*/>
modify the backup schedule	REFRESH	<SCHEDULE EVENT="BACKUP" WEEKDAY*n*="*timevalue*"/>
invoke an ad hoc backup	REFRESH	<BACKUP *options*/>
recover the server from a backup if a metadata update causes applications to stop functioning (but the server is still functioning)	REFRESH	Single-server configuration: <RECOVER BACKUPNAME="*name*" | BACKUPPATH="*pathname*" *options*/> Clustered server configuration: See "Recovery in a Clustered Server Configuration" on page 173.
restart the scheduler thread	REFRESH	<SCHEDULER/>
regain control of the metadata server during the recovery process in the event that the recover process stops responding	PAUSE or RESUME	<FORCE/>

For usage examples, see "Example 7: Execute Backup and Recover Options in PROC METAOPERATE" on page 183.

For information about how to recover a metadata server or cluster node that is unresponsive, see "What to Do If the SAS Metadata Server Is Unresponsive" and "Use Individual Scripts to Operate Servers" in the *SAS Intelligence Platform: System Administration Guide*. A metadata server or cluster node is unresponsive if it cannot be started or does not respond to a PROC METADATA METHOD=STATUS request.

The server's backup history can be monitored in the Server Backup node of the SAS Management Console Metadata Manager or by using PROC METADATA. For more information, see Chapter 11, "METADATA Procedure," on page 87.

Recovery in a Clustered Server Configuration

In a clustered server configuration, the master server manages restore and recovery. Any server node can leave and rejoin the cluster at any point in time.

When it rejoins, the master server examines it and either forces it to recover from the most recent backup, or feeds it the necessary updates from the master's journal file, without administrative intervention. A slave server is rejected if its cluster GUID does not match that of the master server.

Should the cluster lose quorum, all server nodes are paused to an offline state until the administrator can return enough server nodes to service to regain quorum. These nodes can be existing server nodes that have been repaired, or new nodes. An administrator can add new server nodes by using the SAS Deployment Wizard. The master server automatically brings new and recommissioned server nodes up-to-date, whether they are missing a few transactions or need a complete set of metadata added. The cluster will be in a STARTING state until enough server nodes have been updated to meet the quorum requirements. As soon as the cluster reaches quorum, the available server nodes are changed to an online state. Any server nodes added after quorum is reached are updated by the master while the cluster operates.

The only time an administrator needs to restore a cluster is if repository metadata becomes corrupted. For example, if a large-scale metadata update added metadata to repositories that caused applications to stop functioning. To recover a cluster in the case of corrupted metadata, an administrator must stop the cluster, recover one server in stand-alone mode, and then re-start this one server and other server nodes in normal (clustered) mode. The first server will become master and will recover the slave nodes as they join the cluster. For more information, see "Changing a Clustered Server to Stand-Alone Mode" on page 174.

Changing a Clustered Server to Stand-Alone Mode

In a clustered metadata server configuration, tasks that replace or restructure repository data sets must be performed when the cluster is stopped and on one server that is running in stand-alone mode. Examples of tasks that must be run in stand-alone mode are as follows:

- restoring SAS metadata repositories to an earlier version of metadata
- removing unused disk space from repository data sets by running a backup with the REORG option

To change a cluster to run in stand-alone mode:

1. Stop the cluster by running PROC METAOPERATE with ACTION=STOP or by using a script. By default, PROC METAOPERATE stops all servers in the cluster..
2. Start one server with the `-startNoCluster` option. For information about starting a server, see the *SAS Intelligence Platform: System Administration Guide*.
3. Use SAS Management Console or PROC METAOPERATE to perform the task.

CAUTION

When recovering a server that was started with –startNoCluster, do NOT check the check box to also restore configuration files. Configuration files contain host-specific information. Restoring another server's configuration files can cause problems. Backing up and restoring configuration files is still supported in a

single SAS Metadata Server configuration, but it is not supported in a clustered SAS Metadata Server configuration.

4 After the task is successfully completed, stop the server.

5 Restart the server in normal mode.

6 Start the remaining servers in normal mode.

The first server becomes the master server and recovers the slave nodes as they join the cluster.

Using Alert Email XML Elements

The REFRESH action supports optional XML elements that can be used to test the SAS Metadata Server's alert email notification subsystem. The alert email notification subsystem sends alert email messages when any of the following occur:

- an alert email test
- a metadata server backup error
- a metadata server recovery error
- an error prevents repository data sets from being updated from the journal
- a slave server fails to initialize or synchronize and shuts itself down.

XML elements are available to send a test alert email message to configured recipients, to temporarily change the addressees, and to temporarily modify the alert email server configuration until an alert email can be sent successfully. The modifications are active for the duration of the server session. To permanently modify the alert email settings, you must stop the SAS Metadata Server and modify the options in the appropriate configuration files. Alert email addressees are permanently configured in the omaconfig.xml file. The alert email server is permanently configured in the sasv9.cfg file

The following table summarizes alert email testing tasks that can be performed with PROC METAOPERATE ACTION=REFRESH.

Task	OPTION=
Send a test alert email message	<OMA ALERTEMAILTEST="*text*"/>
Specify a different alert email recipient	<OMA ALERTEMAIL="*email-address*"/>
Change the email authentication protocol	<OMA EMAILAUTHPROTOCOL="LOGIN | NONE"/>
Change the email server	<OMA EMAILHOST="*server-network-address*"/>
Change the email port	<OMA EMAILPORT="*port-number*"/>

Task	OPTION=
Change the email address in the From field	<OMA EMAILID="*server-email-address*"/>
Change the password associated with the From user address	<OMA EMAILPW="*password*"/>

For usage examples, see “Example 6: Test the Alert Email Notification Subsystem with PROC METAOPERATE” on page 181.

To get the current value of the alert email settings, use PROC METADATA. When you specify METHOD=STATUS, submitting the above-mentioned XML elements in the IN= argument returns current values for each option. See “Example 6: Get Information about the Server’s Alert Email Notification Subsystem with PROC METADATA” on page 115.

Examples: METAOPERATE Procedure

Example 1: Get the SAS Metadata Server’s Status with PROC METAOPERATE

Features: ACTION=STATUS argument
OUT= argument

Details

These examples use ACTION=STATUS to request basic information about the configuration and availability of the SAS Metadata Server. They also show how the default log output compares to the output that can be obtained with the OUT= argument. The OUT= argument returns the same information in a SAS data set.

To issue more detailed queries about the metadata server's configuration and availability, use Chapter 11, “METADATA Procedure,” on page 87.

ACTION=STATUS Argument with No Other Arguments

By default, the output goes to the SAS log.

```
proc metaoperate
   action=status;
run;
```

Example Code 13.1 *Status Information That Is Written to the SAS Log*

```
NOTE: Server a123.us.company.com SAS Version is 9.4.
NOTE: Server a123.us.company.com SAS Long Version is 9.04.01M3P10022014.
NOTE: Server a123.us.company.com Operating System is X64_7PRO.
NOTE: Server a123.us.company.com Operating System Family is WIN.
NOTE: Server a123.us.company.com Operating System Version is Service Pack 1.
NOTE: Server a123.us.company.com Client is myuserid.
NOTE: Server a123.us.company.com Metadata Model is Version 16.01.
NOTE: Server a123.us.company.com is RUNNING on 04Nov2014:15:44:40.
```

ACTION=STATUS with the OUT= Argument

The OUT= argument directs output to a SAS data set. This data set is created in the WORK library. PROC PRINT is used to print the contents of the WORK.STATOUT data set.

```
proc metaoperate
   action=status
   out=statout;
run;

proc print data=work.statout;
run;
```

Output 13.1 *PROC PRINT of Data Set WORK.STATOUT*

The SAS System

Obs	ATTRIBUTE	VALUE
1	Server	a123.us.company.com
2	Version	9.4
3	Sysvlong	9.04.01M3P10022014
4	OS	X64_7PRO
5	OS Family	WIN
6	OS Version	Service Pack 1
7	Client	myuserid
8	Query Time	04Nov2014:14:54:50
9	Status	Running

Example 2: Pause and Resume the SAS Metadata Server

Features: ACTION=PAUSE argument

OPTIONS= argument with <SERVER STATE=”ADMIN”/> and <PAUSECOMMENT>text</PAUSECOMMENT>

ACTION=RESUME argument

Note: You must be an administrative user of the metadata server to perform these actions.

Details

In a clustered server configuration, the PAUSE action always pauses all server nodes in the cluster. The RESUME action executes on all servers in the cluster.

Pause the Metadata Server

The option <SERVER STATE="ADMIN"/> specifies to downgrade the server to an ADMIN state. If you omit the server <SERVER STATE=“ "/> XML element, the server is changed to an offline state. <PAUSECOMMENT> enables you to specify a reason the metadata server is paused. If any user requests the status of the metadata server, the <PAUSECOMMENT> text is included in the information that is printed to the log. The server remains in the downgraded state until you issue a Resume action.

```
proc metaoperate
   action=pause
   options="<Server State='ADMIN'/>
          <PauseComment>The server will resume at 2a.m.</
PauseComment>";
run;
```

Resume the Metadata Server

The RESUME action restores a paused metadata server to the online state. The RESUME action cannot restart a stopped metadata server.

```
proc metaoperate
     action=resume;
run;
```

Example 3: Enable ARM Logging

Features:	ACTION=REFRESH argument OPTIONS= argument with <ARM/>
Note:	You must be an administrative user of the metadata server to perform this action.

Details

ARM logging is enabled by specifying ACTION=REFRESH and specifying ARM system options in an <ARM/> XML element in the OPTIONS= argument. ARMSUBSYS=(ARM_OMA) specifies to start ARM logging for the metadata server. The ARMLOC= system option specifies the location and filename for the log file. In a clustered server configuration, ARM logging is performed on the master server.

Program

```
proc metaoperate
   action=refresh
   options="<ARM ARMSUBSYS=""(ARM_OMA)"" ARMLOC=""logs/
armfile.log""/>";
run;
```

Example 4: Recover Memory on the Metadata Server

Features: ACTION=REFRESH argument
OPTIONS= argument with <Server/> only

Note: You must be an administrative user of the metadata server to perform this action.

Details

The Refresh action can be used to recover memory by closing and reopening all repositories on the server, or by closing and reopening a specific repository. In a clustered server configuration, the Refresh action always executes on all servers in the cluster.

Close and Reopen All Repositories

Specify the Refresh action and specify the <Server/> XML element in the OPTIONS= argument. Closed containers will be reopened as needed in response to subsequent metadata queries and updates.

```
proc metaoperate
     action=refresh
     options='<Server/>';
run;
```

Close and Reopen a Specific Repository

Specify the Refresh action and specify the <Repository/> XML element with the repository's ID= value.

```
proc metaoperate
     action=refresh
     options='<Repository Id="A5H9YT45"/>';
run;
```

Example 5: Delete All Metadata Records from a Repository

Features: ACTION=EMPTY argument
NOAUTOPAUSE argument

Note: You must be an administrative user of the metadata server to perform this action.

Details

Metadata records are deleted from a metadata repository by specifying ACTION=EMPTY and specifying the repository name in the REPOSITORY= procedure option. The Empty action deletes metadata records from a specified repository, but it does not remove the repository's registration from the repository manager. The Empty action is useful for clearing a repository that will be repopulated. In a clustered server configuration, the Empty action always executes on all servers in the cluster.

Note: The NOAUTOPAUSE argument is required with the Empty action.The NOAUTOPAUSE argument suppresses the automatic pause and resume of the metadata server that occurs when PROC METAOPERATE passes an action to the metadata server. The automatic pause and resume is unwanted because it makes all repositories temporarily unavailable.

Program

```
proc metaoperate
   action=empty
   repository="MyRepos"
   noautopause;
run;
```

Example 6: Test the Alert Email Notification Subsystem with PROC METAOPERATE

Features: ACTION=REFRESH argument

OPTIONS= argument with <OMA ALERTEMAILTEST="*text*"/>, <OMA ALERTEMAIL="*email-address*"/>, <OMA EMAILHOST="*server-network-address*"/>, <OMA EMAILPORT="*port-number*"/>, <OMA EMAILAUTHPROTOCOL="*value*"/>

NOAUTOPAUSE argument

Note: You must be an administrative user of the metadata server to perform this action.

Details

The REFRESH action supports options that enable you to test the configuration of the metadata server's alert email subsystem. Any modifications that you make to both the recipients and the alert email server's configuration are active for the duration of the server session. You must update the sasconfig.cfg and omaconfig.xml files to permanently modify the recipients or email configuration.

Note: To permanently modify the recipients or configuration in a clustered server configuration, you must update the sasconfig.cfg and omaconfig.xml files of all server nodes in the cluster.

Note: The NOAUTOPAUSE argument is recommended with ACTION=REFRESH.

Send a Test Message To the Configured Alert Email Recipients

The alert email recipient(s) that are configured in the omaconfig.xml file are displayed on the **General** tab of the active server's Properties window. When you submit the <OMA ALERTEMAILTEST= "*message*"/> XML element, if the addressees do not receive an email message with the specified text from the metadata server, there is a problem with the email address(es) or the alert email configuration.

```
proc metaoperate
   action=refresh
   options='<OMA ALERTEMAILTEST="Please disregard. This is only a
test."/>'
   noautopause;
run;
```

Output 13.2 *Sample Content of a Test Alert Email Message*

```
This is a test of the Refresh method alert email test options.

This email was sent by the Metadata Server running on host a123.us.company.com
using port 8561.
The server directory is 'C:\SAS\BIServer\Lev1\SASMeta\MetadataServer' and the
current log will
typically be found in the 'Logs' directory.
```

Send a Test Message to a Different Email Address

A test message to an email address that is not in the omaconfig.xml is sent by submitting the <OMA ALERTEMAIL= ” ”/> XML element. If the message is not received, there is likely a problem with the specified email address or with the alert email configuration.

```
proc metaoperate
   action=refresh
   options='<OMA ALERTEMAIL="John.Doe@company.com"/>
<OMA ALERTEMAILTEST="This is a test of the alert email system."/>'
noautopause;
run;
```

Submit System Options That Temporarily Change the Alert Email Server Configuration

Submit email omaconfig.xml elements with ACTION=REFRESH to test email hosts, ports, and protocols. Continue testing until a working combination is found.

```
proc metaoperate
   action=refresh
   options='<OMA EMAILHOST="mailserver2.us.company.com"/>
<OMA EMAILPORT="35"/>
<OMA EMAILAUTHPROTOCOL="login"/>
<OMA ALERTEMAIL="MyUserID@us.company.com"/>
<OMA ALERTEMAILTEST="alert email test using mailserver 2 on port
35."/>'
   noautopause;
run;
```

Example 7: Execute Backup and Recover Options in PROC METAOPERATE

Features:	ACTION=REFRESH argument OPTIONS= argument with <BACKUP/>, <BACKUPCONFIGURATION/>, <SCHEDULE/>, and <RECOVER/>
Note:	You must be an administrative user of the metadata server to perform these actions.

Details

In a clustered metadata server configuration, backups run on the master server. All of the backup configuration files are maintained in a central location, regardless of the server specified in the server connection parameters (not shown here).

Execute an Ad Hoc Backup of the SAS Metadata Server

Specify the Refresh action and specify the <Backup/> XML element in the OPTIONS= procedure option. Use the Comment= attribute of the <Backup/> XML element to specify a reason for backup.

```
proc metaoperate
  action=REFRESH
  options=
    "<Backup Comment='Test of backup system.'/>"
    noautopause;
  run;
```

Modify the Backup Configuration

Specify the Refresh action and specify the <BackupConfiguration/> XML element in the OPTIONS= procedure option. Within the <BackupConfiguration/> XML element, specify the configuration attributes that you want to modify. This request changes the backup retention policy from the default value of 7 days to 4 days.

```
proc metaoperate
  action=REFRESH
  options=
    "<BackupConfiguration DaysToRetainBackups='4'/>"
   noautopause;
  run;
```

Modify the Default Backup Schedule

Specify the Refresh action and specify the <Schedule/> XML element in the OPTIONS= procedure option with the Event= attribute. This example continues nightly backups. However, they will occur at 2 a.m. instead of 1 a.m.

```
proc metaoperate
  action=REFRESH
  options=
    "<Schedule Event='Backup' WeekDay2='0200R' WeekDay3='0200'
 WeekDay4='0200' WeekDay5='0200' WeekDay6='0200' WeekDay7='0200'/>"
```

```
 noautopause;
run;
```

Recover the Metadata Server

Specify the Refresh action and specify the <Recover/> XML element in the OPTIONS= procedure option. Within the <Recover/> XML element, specify the BackupPath= attribute and pathname for the backup that you want to restore. Note that the <Recover/> option is not supported in a running clustered server configuration. In a clustered configuration, the option is available only on a server that is started with the **-startNoCluster** option.For more information, see "Recovery in a Clustered Server Configuration" on page 173.

```
proc metaoperate
   action=REFRESH
   options=
   "<Recover BackupPath='Backups/2010-09-16T16_14_36-05_00'
IncludeAllConfigFiles='N' RollForward='16Sep2010:16:22:30'
PauseComment='The metadata server is being recovered.'
Comment='Recovery from 2010-09-16T16_14_36-05_00'/>"
   noautopause;
  run;
```

Terminate a Hung Recovery Process

Specify the Pause action and submit the <Server/> and <Force/> XML elements in the OPTIONS= procedure option. When you terminate a recovery, put the metadata server in an ADMIN state.

```
proc metaoperate
   action=PAUSE
   options="<Server State='ADMIN'/><Force/>";
run;
```

Example 8: Stop One Server in a Metadata Server Cluster

Features:	ACTION=STOP argument METACONNECT=NONE system option NOCLUSTER argument
Note:	You must be an administrative user of the metadata server to perform this action.

Details

When a clustered metadata server configuration is detected, the STOP action (and all other PROC METAOPERATE actions) are executed uniformly on all of the servers in the cluster by default. This example shows two ways to stop a specific server only.

Note: The following examples assume that the default server connection profile has been activated in the SAS session with the METAPROFILE option. When this profile is active, the SAS session re-routes server connections to another server node in the cluster if the node specified in METASERVER= and METAPORT= system options cannot be found. For more information about the default server connection profile, see “Specifying a Stored Connection Profile” on page 40.

Stop a Specific Server Node Using Connection Options

Specify server connection properties for the node that you want to stop as procedure arguments and specify ACTION=STOP with the NOCLUSTER procedure option. SAS does not enforce METAPROFILE processing when metadata server connection values are specified as procedure options. If the node specified in SERVER= and PORT= cannot be found, the request will fail. NOCLUSTER instructs the cluster’s load balancer to execute the STOP action only on the node identified in SERVER= and PORT=.

```
proc metaoperate
   server="a123.us.company.com"
   port=3182
   user="sasadm@saspw"
   password="adminpw"
   action=stop
   nocluster;
run;
```

Stop a Specific Server Node Using System Options

When the server that you want to stop is connected via system options, the METACONNECT=NONE system option is needed to turn off METAPROFILE processing. Otherwise, SAS attempts to connect to another server from the cluster if the server specified in METASERVER= and METAPORT= cannot be found. NOCLUSTER prevents the reconnection from occurring. Set the METACONNECT= system option again with a blank value following the request. The

METACONNECT= system option is needed to reactivate METAPROFILE processing for subsequent metadata requests.

```
options metaserver="a123.us.company.com"
   metaport=3182
   metauser="sasadm@saspw"
   metapass="adminpw"
   metaconnect=none;

proc metaoperate
   action=stop
   nocluster;
run;

options metaconnect=" ";
```

PART 5

DATA Step Functions

14

Introduction to DATA Step Functions for Metadata

Overview of DATA Step Functions for Metadata

The metadata DATA step functions provide a programming-based interface to create and maintain metadata in the SAS Metadata Server. Alternatively, you can perform metadata tasks by using a product like SAS Management Console. However, with DATA step functions, you can write a SAS program and submit it in batch. You can store information in a data set, create your own customized reports, or use information in an existing data set to update metadata. The DATA step provides broad flexibility with IF-THEN/ELSE conditional logic, DO loops, and more.

This book documents two categories of DATA step functions:

- DATA step functions for reading and writing metadata
- DATA step functions for metadata security administration

Before you can use the metadata DATA step functions, you must issue the metadata system options to establish a connection with the metadata server. For more information, see “Connection Options ” on page 38.

For help with forming your DATA step, see the following references:

- For information about metadata objects, see the SAS Metadata Model: Reference.

- For information about administering metadata, see the *SAS Intelligence Platform: System Administration Guide*.
- For information about using functions in a DATA step, see *SAS Functions and CALL Routines: Reference*.
- For information about DATA step concepts, see *SAS Language Reference: Concepts*.

Best Practices

A product like SAS Management Console or SAS Data Integration Studio is recommended over the DATA step functions for the creation and routine maintenance of metadata. For more information, see Chapter 2, “Using Language Elements That Read and Write Metadata,” on page 7. If you must modify the metadata, run a full backup of the repositories. For more information, see the *SAS Intelligence Platform: System Administration Guide*.

When the metadata server returns multiple objects, they are returned in the same order as they are stored in the metadata server. Therefore, if order is important, your program must examine the objects before it acts on them.

A best programming practice is to define all variables (for example, with a LENGTH or FORMAT statement) in the DATA step before you call any functions. Metadata functions that get values (“METADATA_APPPROP Function” on page 218, “METADATA_GETNATR Function” on page 227, “METADATA_GETNPRP Function” on page 231, “METADATA_GETPROP Function” on page 234, “METADATA_GETURI Function” on page 235, “METADATA_PATHOBJ Function” on page 242, and “METADATA_RESOLVE Function” on page 245) store retrieved metadata values in a SAS character variable. There are a series of return code values that indicate either normal completion (0) or some error (-1, -2, or -3). When the data to be retrieved is longer than the declared length of the character variable to receive it, the data is truncated, and the return code is still 0. Be careful to specify an appropriate length for the character variable to be sure you get all of the data. The maximum length for a character variable is 32,767.

For performance reasons, metadata objects are cached by URI. If an object name includes special characters, you should encode the object name with the URLENCODE function before using the metadata DATA step functions. When the URI is created, the metadata DATA step logic decodes reserved URL characters, which can change the object name. In the following example, the URLENCODE function encodes the table name “J#1% 2” before calling the METADATA_DELOBJ function:

```
data _null_;
   tabname= urlencode( 'J#1%  2' );
   put tabname=;

   rc=metadata_delobj("omsobj:PhysicalTable?PhysicalTable[@SASTableName='" ||
                      strip(tabname) || "'][TablePackage/SASLibrary[@Name?'BASE_PC']]");
   put rc= ;

run;
```

Note: If your metadata object name contains embedded blanks, make sure that the object name in the DATA step has the same number of blanks.

To refresh the metadata object with the most recent data from the metadata server, purge the URI with the “METADATA_PURGE Function” on page 244.

For best performance, always resolve your URI into an ID instance. For example, if you make several function calls on the object “OMSOBJ:LogicalServer?@Name='foo '”, first use the “METADATA_RESOLVE Function” on page 245 or “METADATA_GETNOBJ Function” on page 229 to convert the object to “OMSOBJ:LogicalServer\A57DQR88.AU000003”. URIs in the ID instance form can fully exploit object caching and usually require only one read from the metadata server.

Array Parameters

Several of the DATA step functions use two-dimensional arrays for input or output. The arrays enable applications to move information in and out of the metadata server with fewer calls. However, the DATA step is not two-dimensional, so the following conventions enable you to handle these multiple-row arrays:

- For functions that return arrays, the function asks the metadata server to return only one row (or a specific row) of an output array. The output array is generally kept in an object cache that lasts only as long as the DATA step. The key to the cache is the *uri* argument, and the key to the row is the *n* argument. When you submit the function, it checks whether information from the output array already exists in the cache and, if so, returns the information from the cache. If the information does not exist in the cache, the function calls the metadata server to fill the cache. You can use the *n* argument to iterate through the rows of the array; see how *n* is used in “Examples: DATA Step Functions for Reading Metadata” on page 198 and “Examples: DATA Step Functions for Metadata Security Administration” on page 256.
- The functions that input arrays are similar to the functions that return arrays, but the array is not kept in an object cache. Rather than iterating with an *n* argument, you specify the multiple values in a comma-delimited list. In some functions, you submit two values that must be in parallel. In other words, for a *name*, *value* pair, if you specify three *name* arguments, then you must specify three *value* arguments.

For more information about DO loops and array processing in a DATA step, see *SAS Language Reference: Concepts*.

15

Understanding DATA Step Functions for Reading and Writing Metadata

What Are the DATA Step Functions for Reading and Writing Metadata?

These DATA step functions enable an administrator to set or return information about attributes, associations, and properties from metadata objects.

Table 15.1 Metadata DATA Step Functions for Reading and Writing Metadata

Name	Description
"METADATA_APPPROP Function" on page 218	Returns the value of the specified property from the specified SoftwareComponent or DeployedComponent object.
"METADATA_DELASSN Function" (p. 220)	Deletes all objects that make up the specified association
"METADATA_DELOBJ Function" (p. 222)	Deletes the first object that matches the specified URI
"METADATA_GETATTR Function" (p. 223)	Returns the value of the specified attribute for specified object
"METADATA_GETNASL Function" (p. 225)	Returns the *n*th association of the specified object
"METADATA_GETNASN Function" (p. 226)	Returns the *n*th associated object of the specified association
"METADATA_GETNATR Function" (p. 227)	Returns the *n*th attribute on the object specified by the URI
"METADATA_GETNOBJ Function" (p. 229)	Returns the *n*th object that matches the specified URI
"METADATA_GETNPRP Function" (p. 231)	Returns the *n*th property of the specified object
"METADATA_GETNTYP Function" (p. 233)	Returns the *n*th object type on the metadata server
"METADATA_GETPROP Function" (p. 234)	Returns the specified property of the specified object
"METADATA_GETURI Function" (p. 235)	Returns a URL for the application described by the specified SoftwareComponent object.
"METADATA_NEWOBJ Function" (p. 238)	Creates a new metadata object
"METADATA_PATHOBJ Function" (p. 242)	Returns the Id and Type attributes of the specified folder object
"METADATA_PAUSED Function" (p. 244)	Determines whether the metadata server is paused
"METADATA_PURGE Function" (p. 244)	Purges the specified URI

Name	Description
“METADATA_RESOLVE Function” (p. 245)	Resolves a URI into an object on the metadata server
“METADATA_SETASSN Function” (p. 247)	Modifies an association list for an object
“METADATA_SETATTR Function” (p. 249)	Sets the specified attribute for the specified object
“METADATA_SETPROP Function” (p. 250)	Sets the specified property for the specified object
“METADATA_VERSION Function” (p. 252)	Returns the metadata server model version number

Referencing a Metadata Object with a URI

When you use a metadata DATA step function for reading and writing metadata, you specify an object by using a URI, which is a concept from SAS Open Metadata Architecture. For more information, see “What Is a Metadata Identifier?” on page 13 and “What Is a URI?” on page 14. Here are examples for the DATA step functions for reading and writing metadata:

ID

```
omsobj: A57DQR88.AU000003
```

type/*ID*

```
omsobj: LogicalServer/A57DQR88.AU000003
```

type?@*attribute*='*value*'

```
omsobj: LogicalServer?@Name='SASApp - OLAP Server'
```

Notes:

- The OMSOBJ: prefix is not case sensitive.
- Escape characters are supported with the %*nn* URL escape syntax. For more information, see the URLENCODE function in *SAS Functions and CALL Routines: Reference*.

Comparison of DATA Step Functions to Metadata Procedures

The METADATA procedure can perform some of the same tasks as the DATA step functions for reading and writing metadata. Both language elements can query metadata for reports, or make changes to specified objects.

PROC METADATA can submit any method that is supported by the DoRequest method of the SAS Open Metadata Interface, including all methods of the IOMI class, and the Status method of the IServer class. PROC METADATA produces XML output. By using the XML LIBNAME engine and ODS, you can create reports.

In general, the DATA step functions perform the same tasks as the PROC METADATA methods. However, with the DATA step functions, you do not have to understand the XML hierarchy. You can create the same type of ODS reports as you can with PROC METADATA. Instead of writing the output to an XML data set, you use the DATA step to create an output SAS data set directly. See how these two examples create similar reports from their output: “Example: Creating a Report with the DATA Step” on page 21 and “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 16.

Examples: DATA Step Functions for Reading Metadata

Overview

This section describes how to use SAS metadata DATA step functions to identify and track metadata that describes data libraries and users. The examples show how to:

- list the SAS libraries that are defined in metadata
- list the servers that are used to access the libraries
- list the logins defined on the system and their associated user identities and authentication domains
- list user group assignments
- list the logins used by metadata identities

Because these examples do not create metadata, they can be run on a production metadata server. However, they must be executed by a user who has authorization to the metadata, such as the SAS Administrator.

Metadata Access Overview

There is no need to know the physical location of metadata to access it. All access to metadata is controlled by the SAS Metadata Server. To use the metadata server, a program must establish a connection to it. In a SAS program, you establish a connection by specifying metadata system options. For information about the metadata server connection system options, see “Overview of System Options for Metadata” on page 37.

Once a server connection is established, communicating with the metadata server involves defining variables for function arguments and issuing metadata DATA step functions. Use the FORMAT or LENGTH statement to define argument variables. Use the KEEP statement to specify which variables to include in the output data set. You should be familiar with the SAS Metadata Model metadata types that represent entities that you want to query, and the properties defined for each metadata type.

The SAS metadata DATA step functions use a Uniform Resource Identifier (URI) argument to access metadata objects. The preferred URI forms are:

```
"omsobj: type/ID"
```

or

```
"omsobj: type?@attribute='value'"
```

type is the name of the metadata type that represents the entity in the SAS Metadata Model. The `ID` value or *attribute*= ‘*value*’ pair is used as a filter to locate the objects that meet the criteria.

In the examples that follow, the following URI is used to request all objects of the specified type:

```
"omsobj:type?@Id contains '.'"
```

The URI can return multiple objects. The GETNOBJ DATA step function returns output in a two-dimensional array. Each call to GETNOBJ retrieves the URI representing the nth object returned from the query. In the examples, `n=1` specifies to get the first object in the array. `n+1` specifies to iterate through the content of the array.

Featured Functions

METADATA_GETNOBJ
: Returns the nth object that matches the specified URI. For more information, see “METADATA_GETNOBJ Function” on page 229.

METADATA_GETATTR
: Returns the value of the specified attribute for the specified object. For more information, see “METADATA_GETATTR Function” on page 223.

METADATA_GETNASN
: Returns the nth associated object of the specified association. For more information, see “METADATA_GETNASN Function” on page 226.

METADATA_RESOLVE
Resolves a URI into an object on the metadata server. For more information, see "METADATA_RESOLVE Function" on page 245.

Featured Metadata Types and Associations

- A SAS library is described in the SAS Metadata Model by the SASLibrary metadata type. A directory is described by the Directory metadata type. A database schema is described by the DatabaseSchema metadata type. A SASLibrary metadata object has a UsingPackages association to a Directory or DatabaseSchema metadata object.
- A group of SAS servers of different server types is described in the SAS Metadata Model by the ServerContext metadata type. A SASLibrary metadata object has a DeployedComponents association to a ServerContext metadata object.
- External logins are represented in the SAS Metadata Model by the Login metadata type. Users are represented by the Person metadata type. User groups are represented by the IdentityGroup metadata type. Person and IdentityGroup are subtypes of the Identity metadata type. An authentication domain is represented by the AuthenticationDomain metadata type. A Login object has an AssociatedIdentities association to objects of the Identity subtypes. A Login object has a Domains association to an AuthenticationDomain object.
- A Person metadata object has an IdentityGroups association to an IdentityGroup metadata object. A Person metadata object has a Logins association to a Login metadata object.
- Internal logins are represented by the InternalLogin metadata type. An InternalLogin has a ForIdentity association to an identity. An identity has a InternalLoginInfo association to an InternalLogin.

For more information about the metadata types and their associations, see SAS Metadata Model: Reference.

Example: Listing Libraries and Their Associated Directory or Database Schema

This program uses the SAS metadata DATA step functions to query the metadata repository, and return a list of all libraries and their associated directory or database schema. The results are returned to a SAS data set in the Work library, which is printed with PROC PRINT.

Note: When running the program, be sure to modify the META* system options to provide connection parameters for your metadata server. METAUSER should be a user who has ReadMetadata permission to the metadata objects being queried. METAREPOSITORY specifies to look in the Foundation repository. If you have more than one metadata repository, you might want to run this program on all of the repositories. The same is true for other examples in this section.

```
/*Connect to the metadata server.  */

options metaserver="myserver"
    metaport=8561
    metauser="sasadm@saspw"
    metapass="adminpw"
    metarepository="Foundation";

/* Begin the query. The DATA statement names the output data set.  */

data metadata_libraries;

/* The LENGTH statement defines variables for function arguments and
assigns the maximum length of each variable.  */

  length liburi upasnuri $256 name $128 type id $17 libref engine $8 path
mdschemaname schema $256;

/* The KEEP statement defines the variables to include in the
output data set.  */

  keep name libref engine path mdschemaname schema;

/* The CALL MISSING routine initializes the output variables to missing values.  */

  call missing(liburi,upasnuri,name,engine,libref);

  /* The METADATA_GETNOBJ function specifies to get the SASLibrary objects
in the repository. The argument nlibobj=1 specifies to get the first object that
matches the requested URI. liburi is an output variable. It will store the URI
of the returned SASLibrary object. */

  nlibobj=1;
  librc=metadata_getnobj("omsobj:SASLibrary?@Id contains '.'",nlibobj,liburi);

  /* The DO statement specifies a group of statements to be executed as a unit
for each object that is returned by METADATA_GETNOBJ. The METADATA_GETATTR function
is used to retrieve the values of the Name, Engine, and Libref attributes of
the SASLibrary object.  */

  do while (librc>0);

     /* Get Library attributes */
     rc=metadata_getattr(liburi,'Name',name);
     rc=metadata_getattr(liburi,'Engine',engine);
      rc=metadata_getattr(liburi,'Libref',libref);

      /* The METADATA_GETNASN function specifies to get objects associated to the
library via the UsingPackages association. The n argument specifies to return the
first associated object for that association type. upasnuri is an output variable.
It will store the URI of the associated metadata object, if one is found.  */

        n=1;
        uprc=metadata_getnasn(liburi,'UsingPackages',n,upasnuri);

        /* When a UsingPackages association is found, the METADATA_RESOLVE function
```

```
is called to resolve the URI to an object on the metadata server. The CALL MISSING
routine assigns missing values to output variables.  */

        if uprc > 0 then do;
           call missing(type,id,path,mdschemaname,schema);
           rc=metadata_resolve(upasnuri,type,id);

           /* If type='Directory', the METADATA_GETATTR function is used to get its
path and output the record */

           if type='Directory' then do;
              rc=metadata_getattr(upasnuri,'DirectoryName',path);
              output;
              end;

           /* If type='DatabaseSchema', the METADATA_GETATTR function is used to get
the name and schema, and output the record */

           else if type='DatabaseSchema' then do;
               rc=metadata_getattr(upasnuri,'Name',mdschemaname);
               rc=metadata_getattr(upasnuri,'SchemaName',schema);
              output;
              end;

        /* Check to see if there are any more Directory objects */

            n+1;
            uprc=metadata_getnasn(liburi,'UsingPackages',n,upasnuri);
          end; /* if uprc > 0 */

     /* Look for another library */

     nlibobj+1;
     librc=metadata_getnobj("omsobj:SASLibrary?@Id contains '.'",nlibobj,liburi);
  end; /* do while (librc>0) */
run;

/* Print the metadata_libraries data set */

proc print data=metadata_libraries; run;
```

The example creates output similar to the following:

Figure 15.1 *PROC PRINT of metadata_libraries Data Set*

The SAS System

Obs	name	libref	engine	path	mdschemaname	schema
1	Information Map Library	maplib	SASIOIME		Information Map Library	
2	Microsoft Excel Library	exelref	EXCEL		Microsoft Excel Library	
3	Oracle Library	oraref	ORACLE		Oracle Library	Austin
4	Sample Employee Data	sampdata	BASE	C:\Program Files\SAS\SASFoundation\9.3 \core\sample		
5	SASDemoTest	demotest	BASE	C:\mytest		
6	MyTest2	mytest2	BASE	C:\mytest		
7	MyTest	mytest	BASE	C:\mytest		
8	SASApp - wrstemp	wrstemp	BASE	C:\SAS\BIserver\Lev1\SASApp\Data\wrstemp		
9	SASApp - wrsdist	wrsdist	BASE	C:\SAS\BIserver\Lev1\SASApp\Data\wrsdist		
10	STP Samples	stpsamp	BASE	C:\Program Files\SAS\SASFoundation\9.3 \inttech\sample		
11	SASApp - SASDATA	SASDATA	BASE	Data		

Example: Listing Libraries and Their Server Contexts

This program uses the SAS metadata DATA step functions to return more detailed information about the libraries. The results are returned to a Libraries data set in the Work library. The requested data includes the library metadata ID, the library name, libref, engine, path on the file system (or if DBMS data, the DBMS path), and the server contexts to which the library is associated.

```
/*Connect to the metadata server with the metadata system options,
as shown in the previous example. */

data work.Libraries;

/* The LENGTH statement defines variables for function arguments and
assigns the maximum length for each variable.  */

  length LibId LibName $ 32 LibRef LibEngine $ 8 LibPath $ 256
ServerContext uri uri2 type $ 256 server $ 32;
```

```
/* The LABEL statement assigns descriptive labels to variables. */

  label
LibId = "Library Id"
LibName = "Library Name"
LibRef = "SAS Libref"
LibEngine = "Library Engine"
ServerContext = "Server Contexts"
LibPath = "Library Path"
;

/* The CALL MISSING routine initializes output variables to missing values.  */

  call missing(LibId,LibName,LibRef,LibEngine,LibPath,
       ServerContext,uri,uri2,type,server);
  n=1;
  n2=1;

  /* The METADATA_GETNOBJ function gets the first Library object. If none
are found, the program prints an informational message. */
  rc=metadata_getnobj("omsobj:SASLibrary?@Id contains '.'",n,uri);
  if rc<=0 then put "NOTE: rc=" rc
    "There are no Libraries defined in this repository"
    " or there was an error reading the repository.";

/* The DO statement specifies a group of statements to be executed as a unit
for the object that is returned by METADATA_GETNOBJ. The METADATA_GETATTR
function gets the values of the Id, Name, LibRef, and Engine attributes
of the SASLibrary object.  */

  do while(rc>0);
     objrc=metadata_getattr(uri,"Id",LibId);
     objrc=metadata_getattr(uri,"Name",LibName);
objrc=metadata_getattr(uri,"Libref",LibRef);
objrc=metadata_getattr(uri,"Engine",LibEngine);

     /* The METADATA_GETNASN function gets objects associated
via the DeployedComponents association. If none are found, the program
prints an informational message. */

     objrc=metadata_getnasn(uri,"DeployedComponents",n2,uri2);
     if objrc<=0 then
       do;
         put "NOTE: There is no DeployedComponents association for "
             LibName +(-1)", and therefore no server context.";
         ServerContext="";
       end;

 /* When an association is found, the METADATA_GETATTR function gets
the server name. */

       do while(objrc>0);
         objrc=metadata_getattr(uri2,"Name",server);
         if n2=1 then ServerContext=quote(trim(server));
         else ServerContext=trim(ServerContext)||" "||quote(trim(server));
```

```
/* Look for another ServerContext */
         n2+1;
         objrc=metadata_getnasn(uri,"DeployedComponents",n2,uri2);
       end; /*do while objrc*/

     n2=1;

     /* The METADATA_GETNASN function gets objects associated via the
UsingPackages association. The program prints a message if an
association is not found.*/

     objrc=metadata_getnasn(uri,"UsingPackages",n2,uri2);
     if objrc<=0 then
       do;
         put "NOTE: There is no UsingPackages association for "
             LibName +(-1)", and therefore no Path.";
         LibPath="";
       end;

/* When a UsingPackages association is found, the METADATA_RESOLVE function
is called to resolve the URI to an object on the metadata server. */

       do while(objrc>0);
         objrc=metadata_resolve(uri2,type,id);

    /*if type='Directory', the METADATA_GETATTR function is used to get its path */

    if type='Directory' then objrc=metadata_getattr(uri2,"DirectoryName",LibPath);

    /*if type='DatabaseSchema', the METADATA_GETATTR function is used to get
the name */

     else if type='DatabaseSchema' then objrc=metadata_getattr(uri2, "Name", LibPath);
         else LibPath="*unknown*";

        /* output the records */
         output;
         LibPath="";

        /* Look for other directories or database schemas */

         n2+1;
         objrc=metadata_getnasn(uri,"UsingPackages",n2,uri2);
       end; /*do while objrc*/

     ServerContext="";
     n+1;

        /* Look for other libraries */

     n2=1;
     rc=metadata_getnobj("omsobj:SASLibrary?@Id contains '.'",n,uri);

  end; /*do while rc*/

/* The KEEP statement defines the variables to include in the output data set. */
```

```
  keep
LibId
LibName
LibRef
LibEngine
ServerContext
LibPath;
run;

/* Write a basic listing of data */

proc print data=work.Libraries label;
  /* subset results if you wish
     where indexw(ServerContext,'"SASMain"') > 0; */
run;
```

The example creates output similar to the following:

Figure 15.2 *PROC PRINT of work.Libraries Data Set*

The SAS System

Obs	Library Id	Library Name	SAS Libref	Library Engine	Library Path	Server Contexts
1	A5TJRDIT.AZ0000S4	Information Map Library	maplib	SASIOIME	Information Map Library	"SASMeta" "SASApp"
2	A5TJRDIT.AZ0000S3	Microsoft Excel Library	exelref	EXCEL	Microsoft Excel Library	"SASMeta" "SASApp"
3	A5TJRDIT.AZ0000S2	Oracle Library	oraref	ORACLE	Oracle Library	"SASMeta" "SASApp"
4	A5TJRDIT.AZ000009	Sample Employee Data	sampdata	BASE	C:\Program Files\SAS\SASFoundation\9.3 \core\sample	"SASApp"
5	A5TJRDIT.AZ000007	SASDemoTest	demotest	BASE	C:\mytest	"SASApp"
6	A5TJRDIT.AZ000006	MyTest2	mytest2	BASE	C:\mytest	"SASMeta" "SASApp"
7	A5TJRDIT.AZ000005	MyTest	mytest	BASE	C:\mytest	"SASMeta" "SASApp"
8	A5TJRDIT.AZ000004	SASApp - wrstemp	wrstemp	BASE	C:\SAS\BIserver\Lev1 \SASApp\Data\wrstemp	"SASApp"
9	A5TJRDIT.AZ000003	SASApp - wrsdist	wrsdist	BASE	C:\SAS\BIserver\Lev1 \SASApp\Data\wrsdist	"SASApp"
10	A5TJRDIT.AZ000002	STP Samples	stpsamp	BASE	C:\Program Files\SAS\SASFoundation\9.3 \inttech\sample	"SASApp"
11	A5TJRDIT.AZ000001	SASApp - SASDATA	SASDATA	BASE	Data	"SASApp"

Example: Listing Logins and Their Associated Identities and Authentication Domains

This program uses the SAS metadata DATA step functions to query the metadata repository, and return a list of all logins and the users or groups to which they belong. It returns the authentication domains in which the logins are active. The results are returned to a Logins data set in the Work library.

Note: A typical user can see only logins that he or she owns, and the logins of groups of which he or she is a member. For this example to return meaningful information, it must be executed by an unrestricted user or by a user who has been assigned the User and Group Administrative role.

```
/*Connect to the metadata server using the metadata system options
shown in the first example.*/

data logins;

  /* The LENGTH statement defines variables for function arguments and assigns
the maximum length for each variable.  */

  length LoginObjId UserId IdentId AuthDomId $ 17
         IdentType $ 32
         Name DispName Desc uri uri2 uri3 AuthDomName $ 256;

/* The CALL MISSING routine initializes the output variables to missing values.  */

  call missing
(LoginObjId, UserId, IdentType, IdentId, Name, DispName, Desc, AuthDomId, AuthDomName);
  call missing(uri, uri2, uri3);
  n=1;

 /* The METADATA_GETNOBJ function specifies to get the Login objects
in the repository. The n argument specifies to get the first object that
matches the uri requested in the first argument. The uri argument is an output
variable. It will store the actual uri of the Login object that is returned.
The program prints an informational message if no objects are found. */

  objrc=metadata_getnobj("omsobj:Login?@Id contains '.'",n,uri);
  if objrc<=0 then put "NOTE: rc=" objrc
    "There are no Logins defined in this repository"
    " or there was an error reading the repository.";

/* The DO statement specifies a group of statements to be executed as a unit
for the Login object that is returned by METADATA_GETNOBJ. The METADATA_GETATTR
function gets the values of the object's Id and UserId attributes. */

  do while(objrc>0);
     arc=metadata_getattr(uri,"Id",LoginObjId);
     arc=metadata_getattr(uri,"UserId",UserId);
```

```
/* The METADATA_GETNASN function specifies to get objects associated
via the AssociatedIdentity association. The AssociatedIdentity association name
returns both Person and IdentityGroup objects, which are subtypes of the Identity
metadata type. The URIs of the associated objects are returned in the uri2 variable.
If no associations are found, the program prints an informational message. */

     n2=1;
     asnrc=metadata_getnasn(uri,"AssociatedIdentity",n2,uri2);
     if asnrc<=0 then put "NOTE: rc=" asnrc
       "There is no Person or Group associated with the " UserId "user ID.";

/* When an association is found, the METADATA_RESOLVE function is called to
resolve the URI to an object on the metadata server. */

     else do;
       arc=metadata_resolve(uri2,IdentType,IdentId);

    /* The METADATA_GETATTR function is used to get the values of each identity's
Name, DisplayName and Desc attributes. */

       arc=metadata_getattr(uri2,"Name",Name);
       arc=metadata_getattr(uri2,"DisplayName",DispName);
       arc=metadata_getattr(uri2,"Desc",Desc);
     end;

 /* The METADATA_GETNASN function specifies to get objects associated
via the Domain association. The URIs of the associated objects are returned in
the uri3 variable. If no associations are found, the program prints an
informational message. */

     n3=1;
     autrc=metadata_getnasn(uri,"Domain",n3,uri3);
     if autrc<=0 then put "NOTE: rc=" autrc
       "There is no Authentication Domain associated with the " UserId "user ID.";

        /* The METADATA_GETATTR function is used to get the values of each
AuthenticationDomain object's Id and Name attributes. */

     else do;
       arc=metadata_getattr(uri3,"Id",AuthDomId);
       arc=metadata_getattr(uri3,"Name",AuthDomName);
     end;

     output;

  /* The CALL MISSING routine reinitializes the variables back to missing values. */

  call missing(LoginObjId, UserId, IdentType, IdentId, Name, DispName, Desc, AuthDomId,
AuthDomName);

 /* Look for more Login objects */

 n+1;
  objrc=metadata_getnobj("omsobj:Login?@Id contains '.'",n,uri);
  end;
```

```
/* The KEEP statement specifies the variables to include in the output data set.  */

  keep LoginObjId UserId IdentType Name DispName Desc AuthDomId AuthDomName;
run;

/* The PROC PRINT statement prints the output data set. */
proc print data=logins;
   var LoginObjId UserId IdentType Name DispName Desc AuthDomId AuthDomName;
run;
```

The example creates output similar to the following:

Figure 15.3 *PROC PRINT of Logins Data Set*

The SAS System

Obs	LoginObjId	UserId	IdentType	Name	DispName	Desc	AuthDomId	AuthDomName
1	A5TJRDIT.AQ000004	omitest1	Person	testuser2	Advanced User	An advanced user account for testing purposes.	A5TJRDIT.AP000001	DefaultAuth
2	A5TJRDIT.AQ000003	omitest	Person	testuser1	General User	A general user account for testing purpose.	A5TJRDIT.AP000001	DefaultAuth
3	A5TJRDIT.AQ000002	vmw0116 \sassrv	IdentityGroup	SAS General Servers	SAS General Servers	Allows members to be used for launching stored process servers and pooled workspace servers.	A5TJRDIT.AP000001	DefaultAuth
4	A5TJRDIT.AQ000001	vmw0116 \sasdemo	Person	sasdemo	SAS Demo User		A5TJRDIT.AP000001	DefaultAuth

Example: Listing User Group Memberships

This program uses the SAS metadata DATA step functions to query the metadata repository, and return a list of all users and the user groups to which they belong. The results are returned to a Users_Grps data set in the Work library. The results are presented in a listing created with PROC REPORT.

Note: User groups are represented in the SAS Metadata Model by the IdentityGroup metadata type. The IdentityGroup metadata type is also used to represent roles. This example lists IdentityGroup objects of both types. If you want to exclude roles from the listing, use the METADATA_GETATTR function to get the value of each object's PublicType attribute. A traditional user group has PublicType= "UserGroup". A role has PublicType="Role". Then, use the values to distinguish between two types of IdentityGroup objects.

```
/*Connect to the metadata server using the metadata system options as
shown in the first example. */
```

```
data users_grps;

/* The LENGTH statement defines variables for function arguments and
assigns the maximum length of each variable.  */

  length uri name dispname group groupuri $256
id MDUpdate $20;

/* The CALL MISSING routine initializes output variables to missing values.*/

  n=1;
      call missing(uri, name, dispname, group, groupuri, id, MDUpdate);


  /* The METADATA_GETNOBJ function specifies to get the Person objects
in the repository. The n argument specifies to get the first Person object that is
returned. The uri argument will return the actual uri of the Person object that
is returned. The program prints an informational message if no Person objects
are found. */

      nobj=metadata_getnobj("omsobj:Person?@Id contains '.'",n,uri);
  if nobj=0 then put 'No Persons available.';

/* The DO statement specifies a group of statements to be executed as a unit
for the Person object that is returned by METADATA_GETNOBJ. The METADATA_GETATTR
function gets the values of the object's Name and DisplayName attributes. */

  else do while (nobj > 0);
     rc=metadata_getattr(uri, "Name", Name);
     rc=metadata_getattr(uri, "DisplayName", DispName);


/* The METADATA_GETNASN function gets objects associated via the IdentityGroups
association. The a argument specifies to return the first associated object for
that association type. The URI of the associated object is returned in the
groupuri variable.  */

   a=1;
     grpassn=metadata_getnasn(uri,"IdentityGroups",a,groupuri);

        /* If a person does not belong to any groups, set their group
          variable to 'No groups' and output their name. */

     if grpassn in (-3,-4) then do;
            group="No groups";
        output;
     end;

        /* If the person belongs to many groups, loop through the list
          and retrieve the Name and MetadataUpdated attributes of each group,
            outputting each on a separate record. */

     else do while (grpassn > 0);
        rc2=metadata_getattr(groupuri, "Name", group);
        rc=metadata_getattr(groupuri, "MetadataUpdated", MDUpdate);
        a+1;
```

```
        output;
        grpassn=metadata_getnasn(uri,"IdentityGroups",a,groupuri);
      end;

        /* Retrieve the next person's information */

      n+1;
      nobj=metadata_getnobj("omsobj:Person?@Id contains '.'",n,uri);
  end;

/* The KEEP statement specifies the variables to include in the output data set. */

  keep name dispname MDUpdate group;
run;

   /* Display the list of users and their groups */
proc report data=users_grps nowd headline headskip;
  columns name dispname group MDUpdate;
  define name / order 'User Name' format=$30.;
  define dispname / order 'Display Name' format=$30.;
  define group / order 'Group' format=$30.;
  define MDUpdate / display 'Updated' format=$20.;
  break after name / skip;
run;
```

The example creates output similar to the following:

Figure 15.4 *PROC REPORT of Users_Grps Data Set*

The SAS System

User Name	Display Name	Group	Updated
sasadm	SAS Administrator	META: Unrestricted Users Role	04Jan2011:16:29:20
		SASAdministrators	04Jan2011:16:29:20
sasdemo	SAS Demo User	No groups	04Jan2011:16:39:26
sastrust	SAS Trusted User	SAS General Servers	04Jan2011:16:31:44
		SAS System Services	04Jan2011:16:29:20
testuser1	General User	Web Report Studio: Report View	04Jan2011:16:40:11
testuser2	Advanced User	Job Execution: Job Administrat	04Jan2011:16:38:32
		Job Execution: Job Designer	04Jan2011:16:38:32
		Job Execution: Job Scheduler	04Jan2011:16:38:32
		Job Execution: Job Submitter	04Jan2011:16:38:32
		Web Report Studio: Advanced	04Jan2011:16:40:11
		Web Report Studio: Report Crea	04Jan2011:16:40:11
		Web Report Studio: Report View	04Jan2011:16:40:11
webanon	SAS Anonymous Web User	BI Web Services Users	04Jan2011:16:39:26

Example: Listing Users and Their Logins

This program uses the SAS metadata DATA step functions to query the metadata repository, and return a list of all Person objects and their associated logins. The SAS Metadata Server supports external user accounts and internal user accounts. Both types of user accounts are modeled with the metadata type Person. An internal user account differs from an external user account in that a user specifies the user name with the suffix @saspw to log on (for example, sasadm@saspw). This value is known only to the metadata server. External accounts require domain-qualified user IDs to log on. An external account can have multiple logins defined for it. The logins are controlled with authentication domains. This program requests a listing of the users that are defined on the metadata server, and notes whether they are internal or external accounts. It lists the logins and authentication domains for each external account. The results are returned to an Identities data set in the Work library. The example includes code to print the listing with PROC PRINT, or to export it to a Microsoft Excel spreadsheet with PROC EXPORT.

```
/*Connect to the metadata server using the metadata system options as
shown in the first example. */

data work.Identities;

/* The LENGTH statement defines the lengths of variables for function arguments. */
length IdentId IdentName DispName ExtLogin IntLogin DomainName $32
uri uri2 uri3 uri4 $256;

/* The LABEL statement assigns descriptive labels to variables. */
label
    IdentId    = "Identity Id"
    IdentName  = "Identity Name"
    DispName   = "Display Name"
    ExtLogin   = "External Login"
    IntLogin   = "Is Account Internal?"
    DomainName = "Authentication Domain";

/* The CALL MISSING statement initializes the output variables to missing values. */
call missing(IdentId, IdentName, DispName, ExtLogin, IntLogin, DomainName,
uri, uri2, uri3, uri4);
n=1;
n2=1;

/* The METADATA_GETNOBJ function specifies to get the Person objects in the repository.
The n argument specifies to get the first person object that is returned.
The uri argument will return the actual uri of the Person object. The program prints an
informational message if no objects are found. */

rc=metadata_getnobj("omsobj:Person?@Id contains '.'",n,uri);
if rc<=0 then put "NOTE: rc=" rc
"There are no identities defined in this repository"
" or there was an error reading the repository.";

/* The DO statement specifies a group of statements to be executed as a unit.
The METADATA_GETATTR function gets the values of the Person object's Id, Name,
```

```
and DisplayName attributes. */
do while(rc>0);
    objrc=metadata_getattr(uri,"Id",IdentId);
    objrc=metadata_getattr(uri,"Name",IdentName);
    objrc=metadata_getattr(uri,"DisplayName",DispName);

/* The METADATA_GETNASN function gets objects associated via the
InternalLoginInfo association. The InternalLoginInfo association returns
internal logins. The n2 argument specifies to return the first associated object
for that association name. The URI of the associated object is returned in
the uri2 variable. */

objrc=metadata_getnasn(uri,"InternalLoginInfo",n2,uri2);

/* If a Person does not have any internal logins, set their IntLogin
variable to 'No' Otherwise, set to 'Yes'. */
IntLogin="Yes";
DomainName="**None**";
if objrc<=0 then
do;
put "NOTE: There are no internal Logins defined for " IdentName +(-1)".";
IntLogin="No";
end;

/* The METADATA_GETNASN function gets objects associated via the Logins association.
The Logins association returns external logins. The n2 argument specifies to return
the first associated object for that association name. The URI of the associated
object is returned in the uri3 variable. */

objrc=metadata_getnasn(uri,"Logins",n2,uri3);

/* If a Person does not have any logins, set their ExtLogin
variable to '**None**' and output their name. */
if objrc<=0 then
do;
put "NOTE: There are no external Logins defined for " IdentName +(-1)".";
ExtLogin="**None**";
output;
end;

/* If a Person has many logins, loop through the list and retrieve the name of
each login. */
do while(objrc>0);
objrc=metadata_getattr(uri3,"UserID",ExtLogin);

/* If a Login is associated to an authentication domain, get the domain name. */
DomainName="**None**";
objrc2=metadata_getnasn(uri3,"Domain",1,uri4);
if objrc2 >0 then
do;
 objrc2=metadata_getattr(uri4,"Name",DomainName);
end;

/*Output the record. */
output;
```

```
n2+1;

/* Retrieve the next Login's information */
objrc=metadata_getnasn(uri,"Logins",n2,uri3);
end; /*do while objrc*/

/* Retrieve the next Person's information */
n+1;
n2=1;

rc=metadata_getnobj("omsobj:Person?@Id contains '.'",n,uri);
end; /*do while rc*/

/* The KEEP statement specifies the variables to include in the output data set. */
keep IdentId IdentName DispName ExtLogin IntLogin DomainName;
run;

/* The PROC PRINT statement writes a basic listing of the data. */
proc print data=work.Identities label;
run;

/* The PROC EXPORT statement can be used to write the data to an Excel spreadsheet. */
/* Change DATA= to the data set name you specified above. */
/* Change OUTFILE= to an appropriate path for your system. */

proc export data=work.Identities
    dbms=EXCEL2000
    outfile="C:\temp\Identities.xls"
    replace;
run;
```

The example creates output similar to the following:

Figure 15.5 *PROC PRINT of work.Identities Data Set*

The SAS System

Obs	Identity Id	Identity Name	Display Name	External Login	Is Account Internal?	Authentication Domain
1	A5TJRDIT.AN000006	testuser2	Advanced User	omitest1	No	DefaultAuth
2	A5TJRDIT.AN000005	testuser1	General User	omitest	No	DefaultAuth
3	A5TJRDIT.AN000004	webanon	SAS Anonymous Web User	**None**	Yes	**None**
4	A5TJRDIT.AN000003	sasdemo	SAS Demo User	vmw0116 \sasdemo	No	DefaultAuth
5	A5TJRDIT.AN000002	sastrust	SAS Trusted User	**None**	Yes	**None**
6	A5TJRDIT.AN000001	sasadm	SAS Administrator	**None**	Yes	**None**

16

DATA Step Functions for Reading and Writing Metadata

Dictionary

METADATA_APPPROP Function

Returns the value of the specified property from the specified SoftwareComponent or DeployedComponent object.

Syntax

value=**METADATA_APPPROP**(*softwareComponent*, *property*);

value=**METADATA_APPPROP**(*softwareComponent/deployedComponent*, *property*)

Required Arguments

***softwareComponent* (in)**
specifies the name of a SoftwareComponent object.

***softwareComponent/deployedComponent* (in)**
specifies the name of a DeployedComponent object. The name of the owning SoftwareComponent must precede the name of the DeployedComponent object, separated by a slash.

***property* (in)**
specifies the name of a Property object.

value (out)
returns the value of the specified Property object. The value is in character form.

Details

See “Best Practices” on page 192 for important information about submitting this function.

The function searches the repository specified in the METAREPOSITORY system option.

The METADATA_APPPROP function uses SYSRC and SYSMSG to return function status, error, and warning messages. Valid SYSRC values are:

0
Success completion.

XOMINOREPOSITORY
Unable to connect to the metadata repository.

XOMISCFAIL
No objects match the specified SoftwareComponent name.

XOMIPROPFAIL
No objects match the specified Property name.

Examples

Example 1: Getting a Property Value for a SoftwareComponent

```
data _null_;

    length val $50;
    softwareComp="BI Dashboard 4.3";
    prop="bid.WorkspaceServer";
    val="";

    val = metadata_appprop(softwareComp,prop);

    sysrc=sysrc();
    sysmsg=sysmsg();

    put sysrc=;
    put sysmsg;
    put val=;

run;
```

Output 16.1 *Value Returned for the SoftwareComponent's bid.WorkspaceServer Property*

```
sysrc = 0
 sysmsg is empty
 val = SASApp - Logical Pooled Workspace Server
```

Example 2: Getting a Property Value for a DeployedComponent

```
data _null_;

    length val $50;
    softwareComp="Web Infra Platfrm Soap Svcs 9.3/
SecurityTokenService";
    prop="TokenExpiry";
        val="";

    val = metadata_appprop(softwareComp,prop);

    sysrc=sysrc();
    sysmsg=sysmsg();
```

```
    put sysrc=;
    put sysmsg;
    put val=;

run;
```

Output 16.2 *Value Returned for the DeployedComponent's TokenExpiry Property*

```
sysrc = 0
 sysmsg is empty
 val = 900
```

Example 3: Using the Function in a Macro

```
%let val=%sysfunc(metadata_appprop(BI Dashboard 4.3,bid.DefaultAlertExpirationInterval));
%put &=sysrc;
%put &=sysmsg;
%put &=val;
```

Output 16.3 *Value Returned for the bid.DefaultAlertExpirationInterval Property*

```
sysrc = 0
 sysmsg is empty
 val = 86400000
```

See Also

Functions

- "METADATA_GETPROP Function" on page 234
- "METADATA_GETNPRP Function" on page 231

METADATA_DELASSN Function

Deletes all objects that make up the specified association.

Syntax

rc=**METADATA_DELASSN**(*uri*, *assn*);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***assn* (in)**
specifies an association name.

Return Values

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

Example

```
data _null_;
    length uri $256
           curi $256
           curi1 $256
           curi2 $256;

    rc=0;

    /* Create a PhysicalTable object. */

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

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

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

    put rc=;
    put curi=;

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

```
    put rc=;
    put curi1=;

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

    put rc=;
    put curi2=;

    /* Delete association between table and columns, remove Column objects. */
    rc=metadata_delassn(uri,"Columns");
    put rc=;

    /* Delete PhysicalTable object. */
    rc=metadata_delobj(uri);
    put rc=;

run;
```

See Also

Functions

- “METADATA_SETASSN Function” on page 247
- “METADATA_GETNASN Function” on page 226

METADATA_DELOBJ Function

Deletes the first object that matches the specified URI.

Syntax

rc = **METADATA_DELOBJ**(*uri*);

Required Argument

uri **(in)**
specifies a Uniform Resource Identifier.

Return Values

0
Successful completion

-1
Unable to connect to the metadata server

-2
The deletion was unsuccessful; see the SAS log for details

-3
No objects match the URI

Example

```
data _null_;

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

run;
```

See Also

Functions

- “METADATA_DELASSN Function” on page 220
- “METADATA_GETNOBJ Function” on page 229
- “METADATA_GETNTYP Function” on page 233
- “METADATA_NEWOBJ Function” on page 238

METADATA_GETATTR Function

Returns the value of the specified attribute for the specified object.

Syntax

rc = **METADATA_GETATTR**(*uri*, *attr*, *value*);

Required Arguments

uri **(in)**
specifies a Uniform Resource Identifier.

***attr* (in)**
specifies an attribute of a metadata object.

***value* (out)**
returns the value of the specified attribute.

Return Values

0
Successful completion

-1
Unable to connect to the metadata server

-2
The attribute was not found

-3
No objects match the URI

Example

```
data _null_;

    length name $200
           desc $200
           modified $100;

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

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


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

run;
```

See Also

Functions

- “METADATA_GETNATR Function” on page 227
- “METADATA_SETATTR Function” on page 249

METADATA_GETNASL Function

Returns the *n*th association for the specified object.

Syntax

rc = METADATA_GETNASL(*uri*, *n*, *asn*);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***n* (in)**
numeric index value that indicates which row to return from the array; see “Array Parameters” on page 193 .

***asn* (out)**
returns the association name.

Return Values

n
The number of objects that match the URI

-1
Unable to connect to the metadata server

-3
No objects match the URI

-4
n is out of range

Example

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

    do while(rc>0);

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

        rc=metadata_getnasl("omsobj:Machine?@Name='bluedog'",
                            n,
```

```
                                    assoc);
            put assoc=;
            n=n+1;
        end;
    run;
```

See Also

Functions

- “METADATA_GETNASN Function” on page 226
- “METADATA_SETASSN Function” on page 247

METADATA_GETNASN Function

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

Syntax

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

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***asn* (in)**
specifies an association name.

n (in)
Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 193 .

nuri
returns the URI of the *n*th associated object.

Return Values

n
The number of associated objects

-1
Unable to connect to the metadata server

-3
No objects match the URI

-4
n is out of range

Example

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

    do while(rc>0);

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

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

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

METADATA_GETNATR Function

Returns the *n*th attribute of the specified object.

Syntax

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

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***n* (in)**
specifies a numeric index value that indicates which row to return from the array; see “Array Parameters” on page 193.

***attr* (out)**
returns the name of a metadata object attribute.

***value* (out)**
returns the value of the specified attribute.

Return Values

n
The number of attributes for the URI.

-1
Unable to connect to the metadata server.

-2
No attributes are defined for the object.

-3
No objects match the URI.

-4
n is out of range.

Details

See "Best Practices" on page 192 for important information about submitting this function.

Examples

Example 1: Using an Object URI

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

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

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

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

        n=n+1;

    end;
run;
```

Example 2: Using a Repository URI

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

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

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


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

See Also

Functions

- “METADATA_GETATTR Function” on page 223
- “METADATA_SETATTR Function” on page 249

METADATA_GETNOBJ Function

Returns the *n*th object that matches the specified URI.

Syntax

rc = **METADATA_GETNOBJ**(*uri*, *n*, *nuri*);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***n* (in)**
specifies a numeric index value that indicates which row to return from the array; see "Array Parameters" on page 193.

***nuri* (out)**
returns the URI of the *n*th object that matches the input URI or matches a subtype object of the input URI.

Return Values

n
The number of objects and subtype objects that match the specified URI.

-1
Unable to connect to the metadata server.

-4
n is out of range.

Examples

Example 1: Determining How Many Machine Objects Exist

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

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

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

run;
```

Example 2: Looping Through Each Repository on a Metadata Server

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";
```

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

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

    do while(nobj >= 0);

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

See Also

Functions

- “METADATA_DELOBJ Function” on page 222
- “METADATA_NEWOBJ Function” on page 238

METADATA_GETNPRP Function

Returns the *n*th property of the specified object.

Syntax

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

Required Arguments

uri **(in)**
specifies a Uniform Resource Identifier.

n **(in)**
specifies a numeric index value that indicates which row to return from the array; see “Array Parameters” on page 193.

prop **(out)**
returns the name of an abstract property string.

value **(out)**
returns the value of the specified property string.

Return Values

n
The number of properties for the URI.

-1
Unable to connect to the metadata server.

-2
No properties are defined for the object.

-3
No objects match the URI.

-4
n is out of range.

Details

See “Best Practices” on page 192 for important information about submitting this function.

Example

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

    do while(rc>0);

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

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

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

See Also

Functions

- “METADATA_APPPROP Function” on page 218
- “METADATA_GETPROP Function” on page 234

- “METADATA_SETPROP Function” on page 250

METADATA_GETNTYP Function

Returns the *n*th object type on the server.

Syntax

rc = **METADATA_GETNTYP**(*n*, *type*);

Required Arguments

***n* (in)**
specifies a numeric index value that indicates which row to return from the array; see “Array Parameters” on page 193.

***type* (out)**
returns the metadata type in the specified row.

Return Values

0
No matching type found.

n
The number of objects that match the URI.

-1
Unable to connect to the metadata server.

-4
n is out of range.

Example

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

    do while(rc>0);

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

```
run;
```

METADATA_GETPROP Function

Returns the value and Uniform Resource Identifier (URI) of the specified property for the specified object.

Syntax

rc = **METADATA_GETPROP**(*uri*, *prop*, *value*, *propuri*);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***prop* (in)**
specifies an abstract property string.

***value* (out)**
returns the value of the specified property string.

***propuri* (out)**
returns the URI of the property object that is associated with the input URI.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Named property is undefined.

-3
No objects match the specified URI.

Details

See “Best Practices” on page 192 for important information about submitting this function.

Example

```
data _null_;
```

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

See Also

Functions

- “METADATA_APPPROP Function” on page 218
- “METADATA_GETNPRP Function” on page 231
- “METADATA_SETPROP Function” on page 250

METADATA_GETURI Function

Returns a URL for the application specified by the SoftwareComponent object.

Syntax

rc = **METADATA_GETURI**("*softwareComponent*", *url*, *protocol*, *hostname*, *port*, *service*);

Required Argument

"*softwareComponent*" (in)
specifies the name of a SoftwareComponent object. Valid values are "Stored Process Web App" or "BI Web Services for Java".

Optional Arguments

url (out)
returns a Uniform Resource Locator (URL) that can be used to open the application.

protocol (out)
returns the protocol associated with the application.

hostname (out)
returns the name of the host on which the application is installed.

port (out)
returns the port number associated with the application.

service (out)
returns the service name associated with the application.

Return Values

0
Successful completion.

-1
Unable to connect to the SAS Metadata Server.

-2
No metadata objects match the SoftwareComponent object.

-3
No attributes found for the specified SoftwareComponent.

-4
Failure finding Foundation repository.

-5
Memory allocation failure.

-6
Invalid parameters.

Details

See "Best Practices" on page 192 for important information about submitting this function.

The function returns values for specified output parameters only. The parameters are positional. That is, the requested values are interpreted by the specified output parameter's position in the parameter list, rather than by the parameter's name. Use null values to indicate the position in the parameter list.

The following are examples of valid syntax:

```
/* Get the completed url */
rc = METADATA_GETURI("softwareComponent", url);

/* Get just the service component of the uri */
rc = METADATA_GETURI("softwareComponent",,,,, service);

/* Get the protocol and port parameters */
rc = metadata_geturi("softwareComponent",,protocol,,port, );
```

Examples

Example 1: Getting All Data from the Stored Process Web App SoftwareComponent

```
options metaserver="computer.company.com";
options metaport=8561;
options metaprotocol=bridge;
options metauser="myid";
options metapass="mypassword";
```

```
options metarepository="foundation";


data _null_;
    length protocol $20 hostname $80 port $20 service $100 url $200;
    rc = metadata_geturi("Stored Process Web App", url, protocol,
hostname,
port, service);

    put rc=;
    put url=;
    put protocol=;
    put hostname=;
    put port=;
    put service=;


 run;
```

Here is the example log output:

```
rc = 0
 url = http://computer.company.com:8080/SASStoredProcess
 protocol = http
 hostname = computer.company.com
 port = 8080
 service = /SASStoredProcess
```

Example 2: Getting Selected Output from the BI Web Services for Java SoftwareComponent

```
options metaserver="computer.company.com";
options metaport=8561;
options metaprotocol=bridge;
options metauser="myid";
options metapass="mypassword";
options metarepository="foundation";


data _null_;
    length service $100;
    rc = metadata_geturi("BI Web Services for Java",,,,,service);

    put rc=;
    put url=;
    put protocol=;
    put hostname=;
    put port=;
    put service=;

run;
```

Here is the expected log output:

```
rc = 0
url =
protocol =
hostname =
port =
service = /SASBIWS
```

See Also

Functions:

- “METADATA_PURGE Function” on page 244

METADATA_NEWOBJ Function

Creates a new metadata object.

Syntax

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

Required Arguments

***type* (in)**
specifies a metadata type.

uri (out)
returns a Uniform Resource Identifier (URI).

Optional Arguments

***name* (in)**
specifies a value for the Name attribute of the new metadata object.

***repos* (in)**
specifies the repository identifier of an existing repository. By default, the new object is created in the default repository.

***parent* (in)**
specifies the Uniform Resource Identifier (URI) of an existing metadata object with which to associate the new metadata object. Must be used with ASN. ASN specifies the association name that relates the two metadata objects.

***asn* (in)**
specifies an association name that is relative to the parent metadata object.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Unable to create object; see the SAS log for details.

-9
Unable to allocate memory.

Details

When you create a new metadata object, the object might be unusable if you do not create the proper attributes and associations. For more information, see the SAS Metadata Model: Reference.

The following example creates a SASLibrary object, PhysicalTable object, and Column objects, and associates the library with the table. Note that each object has PublicType= and UsageVersion= attributes defined. The SASLibrary and PhysicalTable objects also have a containing folder defined.

Example

```
data _null_;
    length uri $256
           curi $256
           curi1 $256
           curi2 $256
           luri $256;

    rc=0;

    /* Create a SASLibrary object in the Shared Data folder. */

    rc=metadata_newobj("SASLibrary",
                       luri,
                       "DS Test Library",
                        "Foundation",
                        "omsobj:Tree?@Name='Shared Data'",
                        "Members");
    put rc=;
    put luri=;

    /* Add PublicType= and UsageVersion= attribute values. */

        rc=metadata_setattr(luri,
                            "PublicType",
                            "Library");
        put rc=;
```

```
        put luri=;

        rc=metadata_setattr(luri,
                            "UsageVersion",
                            "1000000.0");
        put rc=;
        put luri=;


    /* Create a PhysicalTable object in the Shared Data folder. */

        rc=metadata_newobj("PhysicalTable",
                       uri,
                       "TestTable",
                        "Foundation",
                        "omsobj:Tree?@Name='Shared Data'",
                        "Members");
    put rc=;
    put uri=;

    /* Add PublicType= and UsageVersion= attribute values. */

        rc=metadata_setattr(uri,
                            "PublicType",
                            "Table");
        put rc=;

        rc=metadata_setattr(uri,
                            "UsageVersion",
                            "1000000.0");
        put rc=;

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

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

        put rc=;
        put curi=;

        /* Add PublicType= and UsageVersion= attribute values to
Column. */
        rc=metadata_setattr(curi,
                            "PublicType",
                            "Column");
        put rc=;

        rc=metadata_setattr(curi,
                            "UsageVersion",
                            "1000000.0");
        put rc=;
```

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

        put rc=;
        put curi1=;

/* Add PublicType= and UsageVersion= attribute values to Column2. */
        rc=metadata_setattr(curi1,
                            "PublicType",
                            "Column");
        put rc=;

        rc=metadata_setattr(curi1,
                            "UsageVersion",
                            "1000000.0");
        put rc=;


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

        put rc=;
        put curi2=;

/* Add PublicType= and UsageVersion= attribute values to Column3. */
        rc=metadata_setattr(curi2,
                            "PublicType",
                            "Column");
        put rc=;

        rc=metadata_setattr(curi2,
                         "UsageVersion",
                         "1000000.0");
        put rc=;

/* Create an association between library and the table */

        rc=metadata_setassn(luri,
                            "Tables",
                            "Append",
                            uri);
        put=rc;

    run;
```

See Also

Functions

- “METADATA_DELOBJ Function” on page 222
- “METADATA_GETNOBJ Function” on page 229

METADATA_PATHOBJ Function

Returns the Id and Type attributes of the specified folder object.

Syntax

rc = **METADATA_PATHOBJ**(*proj*, *path*, *deftype*, *type*, *ID*);

Required Arguments

proj **(in)**
not currently used. Set this argument to null by submitting an empty string “ ”.

path **(in)**
specifies the pathname of the object in SAS folders. Pathname begins with a forward slash. Can include the `deftype` in parentheses as a suffix.

deftype **(in)**
Optionally specifies the value in the TypeName= attribute of the object’s type definition in the SAS type dictionary. Can be omitted if `deftype` is specified as a suffix in the `path` argument. The `deftype` is not the same as the `type`. `type` corresponds to the value in a type definition’s MetadataType= attribute. For example, If you submit a `deftype` of StoredProcess, the returned `type` will be ClassifierMap. For more information, see “What Is the SAS Type Dictionary?” on page 8.

type **(out)**
specifies the metadata type of the returned ID.

ID **(out)**
returns the object's unique identifier.

Return Values

n
Number of objects that match the URI.

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Syntax error in the path.

-3
Named object not found in the path.

Details

See “Best Practices” on page 192 for important information about submitting this function.

Examples

Example 1: Specifying the TypeName Value in Path

```
data _null_;
    length id $20;
    length type $256;
    proj="";
    deftype="";
    id="";
    type="";

    rc=metadata_pathobj(proj,"/Samples/Stored Processes/
Sample(StoredProcess)",
                        deftype,type,id);

    put rc=;
    put id=;
    put type=;

run;
```

Example 2: Specifying the TypeName Value in Deftype

```
data _null_;
    length id $20;
    length type $256;
    proj="";
    deftype="StoredProcess";
    id="";
    type="";

    rc=metadata_pathobj(proj,"/Samples/Stored Processes/Sample",
                        deftype,type,id);

    put rc=;
```

```
    put id=;
    put type=;

run;
```

METADATA_PAUSED Function

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

Syntax

rc = **METADATA_PAUSED**();

Return Values

0
Server is not paused.

1
Server is paused.

-1
Unable to connect to the metadata server.

Example

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

METADATA_PURGE Function

Purges the specified URI.

Syntax

rc = **METADATA_PURGE**(<*uri*>);

Optional Argument

***uri* (in)**
specifies a Uniform Resource Identifier; if no argument is specified, the entire connection is purged from the cache.

Return Values

0
Object successfully purged.

Details

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

Example

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

    do while(rc>0);

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

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

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

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

METADATA_RESOLVE Function

Resolves a URI into an object on the metadata server.

Syntax

rc = METADATA_RESOLVE(*uri*, *type*, *ID*);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***type* (out)**
returns the metadata type for the first object (or subtype object) that matches the input URI.

ID
returns the unique identifier for the first object (or subtype object) that matches the input URI.

Return Values

n
Number of objects and subtype objects that match the specified URI.

0
No objects match the URI.

-1
Unable to connect to the metadata server.

Details

See “Best Practices” on page 192 for important information about submitting this function.

Examples

Example 1: Using an Object URI

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

Example 2: Using a Repository URI

```
data _null_;
    length id $20
       type $256
       attr $256
       value $256;

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

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


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

    end;
run;
```

METADATA_SETASSN Function

Modifies an association list for an object.

Syntax

rc = METADATA_SETASSN(*uri*, *asn*, *mod*, *auri-1*< , *auri-2* … >);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***asn* (in)**
specifies an association name.

***mod* (in)**
specifies the modification to be performed on the metadata object. Values include the following:

APPEND
Appends the specified associations to the end of the specified object's association element list without modifying any of the other associations on the list.

MERGE
Modifies existing associations in the specified object's association list, and adds any associations that do not already exist. (New and changed associations are placed at the end of the association list. Use REPLACE if you need to specify the order of the association list.)

MODIFY
Modifies an existing association, or adds an association that does not already exist. (Use MODIFY with a single association; use MERGE for a multiple association[1].)

REMOVE
Deletes the specified associations from the specified object's association element list without modifying any of the other associations on the list.

REPLACE
For a single association, replaces an existing association with the specified association. For a multiple association, replaces an existing association list with the specified association list. Any existing associations that are not represented in the new association list are deleted.

***auri–1*<, *auri-2* …> (in)**
specifies a list of the URIs of the associated objects; see “Array Parameters” on page 193.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-3
No objects match the input URI.

-4
Unable to perform modification; see the SAS log for details.

-5
Invalid modification.

-6
Unable to resolve association list URIs.

Example

```
data _null_;
```

1. A single association refers to an association name with a 0-to-1 or 1-to-1 cardinality. 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. Any association in which the upper bound for each side of the association is “many” is considered to be a many-to-many association. For more information about single and multiple associations, see “Understanding Associations” in SAS Metadata Model: Reference.

```
      length uri $256;
      rc=0;

      /* Create a TextStore object. */

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

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

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

   run;
```

See Also

Functions

- “METADATA_DELASSN Function” on page 220
- “METADATA_GETNASN Function” on page 226

METADATA_SETATTR Function

Sets the specified attribute for the specified object.

Syntax

rc = **METADATA_SETATTR**(*uri*, *attr*, *value*);

Required Arguments

uri **(in)**
specifies a Uniform Resource Identifier.

attr **(in)**
specifies an attribute of the metadata object.

***value* (in)**
specifies a value for the specified attribute.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Unable to set the attribute.

-3
No objects match the URI.

Example

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

See Also

Functions

- “METADATA_GETATTR Function” on page 223
- “METADATA_GETNATR Function” on page 227

METADATA_SETPROP Function

Sets the specified property for the specified object.

Syntax

rc = **METADATA_SETPROP**(*uri*, *prop*, *value*, *propuri*);

Required Arguments

***uri* (in)**
specifies a Uniform Resource Identifier.

***prop* (in)**
specifies an abstract property string.

***value* (in)**
specifies a value for the specified property.

***propuri* (out)**
returns the URI of the property object that is associated with the input URI.

Return Values

1
New property was created and set.

0
Existing property was successfully set.

-1
Unable to connect to the metadata server.

-2
Unable to set the attribute.

-3
No objects match the URI.

-4
Unable to create a new property.

-9
Unable to allocate memory.

Example

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

See Also

Functions

- “METADATA_GETPROP Function” on page 234
- “METADATA_GETNPRP Function” on page 231

METADATA_VERSION Function

Returns the metadata server's model version number.

Syntax

ver = **METADATA_VERSION**();

Return Values

ver
Metadata server model version number.

-1
Unable to connect to the metadata server.

Example

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

17

Understanding DATA Step Functions for Metadata Security Administration

What Are the DATA Step Functions for Metadata Security Administration?

These DATA step functions enable an administrator to programmatically define or query authorization settings on objects in the SAS Metadata Server. In addition, these functions enable the administrator to create and manipulate access control templates (ACTs) and apply them to objects in the metadata server.

With the metadata security administration functions, the administrator does not need to know how the access controls are stored in metadata. The administrator specifies which permission should be granted or denied to a user, and the metadata server makes the appropriate change in the metadata. These tasks can also be performed with PROC METADATA or the DATA step functions for reading and writing metadata, but those methods can be complicated, and achieving the desired result can be more difficult.

Note: To create security reports about authorization, use the macros that SAS provides. The macros extract authorization information into SAS data sets that you can use to create security reports. For more information, see the *SAS Intelligence Platform: Security Administration Guide*.

Here are the functions, organized by task:

Table 17.1 *Summary Table of DATA Step Functions for Metadata Security Administration*

Task	Functions	Example
Transaction context control	"METASEC_BEGTRAN Function" on page 268 "METASEC_ENDTRAN Function" on page 271	"Example: Begin and End Transaction Context" (p. 256)
Access control definition	"METASEC_APPLYACT Function" on page 267 "METASEC_GETNACT Function" on page 273 "METASEC_GETNAUTH Function" on page 277 "METASEC_GETNID Function" on page 282 "METASEC_REMACT Function" on page 286 "METASEC_SETAUTH Function" on page 289	"Example: Working with ACTs" (p. 257)
ACT manipulation	"METASEC_DELACT Function" on page 269 "METASEC_GETACTA Function" on page 272 "METASEC_GETNACTA Function" on page 275 "METASEC_NEWACT Function" on page 284 "METASEC_SETACTA Function" on page 287	"Example: Working with ACTs" (p. 257)

Transaction Contexts and URIs

The METASEC_BEGTRAN function creates a transaction context (TC), and the METASEC_ENDTRAN function ends it. The TC instance is located in the metadata

server. The TC instance maintains the state of authorization query results and update requests for a client that is using the security administration interface. The TC accumulates changes that are requested for a single object. Submitting the METASEC_ENDTRAN function commits or discards changes, and then ends the TC.

Here are some usage notes:

- For the value of the TC, if you specify an empty string, a temporary context is invoked, no server-side state is maintained, and changes to security settings are made immediately. This choice can be efficient if you have only one change to make, and you want to make the change immediately.
- Specifying the URI is a best practice and is usually required. For DATA step functions that return information, the URI is the key to a cache of information about the object. The information is returned one row at a time in two-dimensional arrays. For more information, see “Array Parameters” on page 193.

If the URI refers to a standard metadata object, but not to an ACT or to a SAS Metadata Repository, you can use a standard URI. For more information, see “What Is a URI?” on page 14.

- If the URI refers to an ACT, the URI must be in the form omsobj:AccessControlTemplate/*my-ACTobj-id*. For example:

  ```
  omsobj:AccessControlTemplate/A5DRX6L4.AT000005
  ```

- If the URI refers to a repository, the URI must be in the form reposid:*my-repos-id*. For example:

  ```
  reposid:A5DRX6L4
  ```

Using the %MDSECCON() Macro

In the DATA step functions for metadata security administration, two arguments are represented in the SAS Open Metadata Architecture as bit flags that can be combined with an OR operation. One argument is *flags*, which is used in many of the functions. The other argument is *auth* in the METASEC_GETNAUTH function.

To simplify usage for the DATA step functions, instead of specifying a numeric parameter, you specify macro variables with easily recognizable names. To use the macro variables, you must first submit the macro %MDSECCON(). The appropriate macro variables are documented with the functions.

Examples: DATA Step Functions for Metadata Security Administration

Overview

These examples are self-contained. Specify your own connection options, and submit the code in a SAS session.

To create security reports about authorization, use the macros that SAS provides. The macros extract authorization information into SAS data sets that you can use to create security reports. For more information, see the *SAS Intelligence Platform: Security Administration Guide*.

CAUTION
Do not run examples against a production metadata server. The examples create objects and identities to demonstrate the use of ACTs. Making changes to security settings poses a risk to a production environment. Be sure to run these examples in an experimental, nonproduction environment.

CAUTION
Do not use this code as an example of creating PhysicalTable and Person objects. The PhysicalTable and Person objects that are created and deleted in these examples are not usable by SAS products because they do not have the appropriate attributes and associations. For information about attributes and associations, see SAS Metadata Model: Reference. For information about metadata administration tasks, see the *SAS Intelligence Platform: System Administration Guide*.

Note: A caller must have administrative access to the metadata server in order to create and delete user definitions.

Example: Begin and End Transaction Context

```
options metaserver="myserver"
        metaport=8561
        metauser="myuser"
        metapass="mypwd"
        metarepository="Foundation";

/* Get macro variable bit flags. */
```

```
%mdseccon();

data _null_;
   format tc $20.;
   length uri $256;
   tc = "";
   uri="";

   /* Create a PhysicalTable object. */
   rc=metadata_newobj("PhysicalTable",
                      uri,
                      "My Demo Table for METASEC");

   /* Start transaction on object created above using the URI. */
   rc=METASEC_BEGTRAN(uri,0,tc);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* ... other operations using the TC ... */

   /* End the transaction and commit any changes made to */
   /* the transaction since it was started.              */
   rc=METASEC_ENDTRAN(uri,tc, &_SECAD_COMMIT_TC );
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* Delete the PhysicalTable */
   rc=metadata_delobj(uri);

run;
```

Example: Working with ACTs

```
options metaserver="myserver"
        metaport=8561
        metauser="myuser"
        metapass="mypwd"
        metarepository="Foundation";

/* Get macro variable bit flags. */
%mdseccon();

/*--------------------------------------*/
/* Create a new user for demo purposes. */
/*--------------------------------------*/

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

```
    /* Create a new Person object. */
    rc=metadata_newobj("Person",
                       uri,
                       "Demo User for METASEC");
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;
   put "The new user's URI is " uri;
run;


/*---------------------------------------------------------------*/
/* Create a new ACT that denies PUBLIC ReadMetadata and grants */
/* SASUSERS ReadMetadata. Grant WriteMetadata and Readmetadata */
/* to a specific person to show the ACT working.               */
/*---------------------------------------------------------------*/
data _null_;
   format tc $20.;
   length uri $256
          act_uri $256
          repos_uri $256
          type $60
          id $17;

   tc = "";
   uri="";

   /* Start transaction - No URI specified because the ACT does not
exist. */
   rc=METASEC_BEGTRAN("",0, tc);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* build the uri for the foundation repository */
   rc=metadata_resolve("omsobj:RepositoryBase?
@Name='Foundation'",type,id);
   tmpstr = substr(id, length(id)-7, 8);
   repos_uri="REPOSID:" || tmpstr;

   /* create the ACT */
   rc=METASEC_NEWACT(tc,repos_uri, "Name", "Grant SASUSERS ACT",
                     "Desc", "ACT that denies PUBLIC but grants
SASUSERS.");
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* The URI parameter is blank because the ACT has not been written
yet.  */
   /* Note the use of &_SECAD_ACT_CONTENTS to indicate that this is
setting
```

```
*/
   /* the content of the ACT rather than security on the
ACT.               */
   rc = METASEC_SETAUTH(tc, "","IdentityGroup", "SASUSERS",
                        "Grant",
"ReadMetadata","",&_SECAD_ACT_CONTENTS);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   rc = METASEC_SETAUTH(tc, "","IdentityGroup", "PUBLIC",

"Deny","ReadMetadata","",&_SECAD_ACT_CONTENTS );
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   rc = METASEC_SETAUTH(tc, "","Person", "Demo User for METASEC",
                        "Grant", "WriteMetadata","",
&_SECAD_ACT_CONTENTS);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   rc = METASEC_SETAUTH(tc, "","Person", "Demo User for METASEC",
                        "Grant", "ReadMetadata","",
&_SECAD_ACT_CONTENTS);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* Protect the ACT so the public cannot edit the ACT. */
        /* The unrestricted user will be the only one who can */
   /* modify the ACT.                                        */
   rc = METASEC_SETAUTH(tc, "","IdentityGroup", "PUBLIC",
                        "Grant","ReadMetadata","");
   rc = METASEC_SETAUTH(tc, "","IdentityGroup", "PUBLIC",
                        "Deny","WriteMetadata","");
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* Commit the transaction and write the ACT. */
   rc=METASEC_ENDTRAN("",tc, &_SECAD_COMMIT_TC );
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;
   else
      put "Transaction creating the ACT has been committed.";
run;
```

```
/*-------------------------------------------*/
/* Start a new DATA step to exercise the ACT. */
/*-------------------------------------------*/

data _null_;
   format tc $20.;
   length uri $256
          act_uri $256
          identitytype $60
          identityname $60
          act_uri2 $256
          actname $60
          actdesc $60
          auth $ 18
          permission $ 60
          condval $ 100
          authorization $30
          authint 8
          type $60
          id $17
          attrname $60
          attrvalue $256;

   tc="";
   uri="";
   attrname="";
   attrvalue="";

   /* Create a PhysicalTable object. */
   rc=metadata_newobj("PhysicalTable",
                      uri,
                      "Demo Table 2 for METASEC");

   /* Start transaction on the object using the object's URI. */
   rc=METASEC_BEGTRAN(uri,0, tc);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* In the SAS log, list the object's URI. */
   put "The object's URI is: " uri;

   /* In the SAS log, list the identities (both inherited and
explicit) */
   /* that have access controls related to the object in the
TC.        */

   put "These identities (both inherited and explicit) have access
controls
related to the object:";
   n=1;
   rc =1;
   do while (rc > 0) ;
      identitytype="";
```

```
      identityname="";
      rc=metasec_getnid(tc, uri, n, identitytype, identityname);
      if (rc < 0 ) then do;
         sysmsg = sysmsg();
         put sysmsg;
         end;
      else do;
         put n= identitytype= identityname=;
         n=n+1;
         end;
      end;

   /* Get list of ACTs on the object. */

   put "ACT or ACTs on the object:";
   n=1;
   rc =1;
   do while (rc > 0) ;
      act_uri2="";
      actname="";
      actdesc="";
      rc=metasec_getnact(tc, uri, n, act_uri2, actname, actdesc);
      if (rc < 0 ) then do;
         sysmsg = sysmsg();
         put sysmsg;
         end;
      else do;
         put n= act_uri2= actname= actdesc=;
         n=n+1;
         end;
      end;

   /* Get the URI for the ACT that was created above.          */
   /* For best performance, resolve URI into an ID instance to */
   /* exploit object caching. (See the best practices topic.)  */

   id="";
   type="";
   rc=metadata_resolve("omsobj:AccessControlTemplate?@Name='Grant
SASUSERS
ACT'",
                       type,id);
   act_uri="omsobj:AccessControlTemplate/" || id;

   /*---------------------------------*/
   /* Apply the ACT to the object.    */
   /*---------------------------------*/
   rc = METASEC_APPLYACT(tc, uri, act_uri);


   /* In the SAS log, list the identities (both inherited and
explicit) */
   /* that have access controls related to the object in the
TC.        */
```

```
   put "After ACT has been applied, these identities have access
controls
related to the object:";
   n=1;
   rc =1;
   do while (rc > 0) ;
      identitytype="";
      identityname="";
      rc=metasec_getnid(tc, uri, n, identitytype, identityname);
      if (rc < 0 ) then do;
         sysmsg = sysmsg();
         put sysmsg;
         end;
      else do;
         put n= identitytype= identityname=;
         n=n+1;
         end;
      end;

   /* Get list of ACTs on the object. */

   put "After ACT has been applied, ACT or ACTs on the object:";
   n=1;
   rc =1;
   do while (rc > 0) ;
      act_uri2="";
      actname="";
      actdesc="";
      rc=metasec_getnact(tc, uri, n, act_uri2, actname, actdesc);
      if (rc < 0 ) then do;
         sysmsg = sysmsg();
         put sysmsg;
         end;
      else do;
         put n= act_uri2= actname= actdesc=;
         n=n+1;
         end;
      end;

   /*-------------------------------------------------------------*/
   /* Next in the log, list all the authorizations on the object. */
   /* Authorizations will be returned in a loop.  The Auth output */
   /* parameter is a bit field that returns much information.     */
   /* It contains bit fields indicating if grants and denies are  */
   /* explicit, from an ACT, or indirect (group or inheritance).  */
   /* Use the macro variable defined in %mdseccon() to determine  */
   /* what is in the fields.                                      */
   /* To create security reports about authorization, use the     */
   /* macros that SAS provides. See information above.            */
   /*-------------------------------------------------------------*/

   put "These are authorizations on the object:";
   rc = 0;
   n=1;
   do while (rc = 0) ;
      condval="";
```

```
      auth="";
      identityname="";
      identitytype="";
      authorization="";
      permission="";

      rc=metasec_getnauth(tc, uri,n,

identitytype,identityname,auth,permission,condval);
      if (rc = 0 )then do;
         n=n+1;
         authint = input(auth, 16.);

         /* The comparisons below must be done in the proper order */
         /* to assure precedence is honored.                       */
         authorization = "Neither Granted nor Denied";
         if (band(authint, &_SECAD_PERM_EXPM) ) then do;
            if (band(authint,&_SECAD_PERM_EXPD )) then
               authorization = "Denied Explicitly";
            else
               authorization = "Granted Explicitly";
            end;
         else if (band(authint, &_SECAD_PERM_ACTM) ) then do;
            if (band(authint,&_SECAD_PERM_ACTD )) then
               authorization = "Denied by ACT";
            else
               authorization = "Granted by ACT";
            end;
         else if (band(authint, &_SECAD_PERM_NDRM) ) then do;
            if (band(authint,&_SECAD_PERM_NDRD )) then
               authorization = "Denied Indirectly";
            else
               authorization = "Granted Indirectly";
            end;

         put identityname= permission= authorization=;
         end; /* if rc =0 */
      end; /* while */

   /* Commit the transaction and write the ACT. */
   rc=METASEC_ENDTRAN("",tc, &_SECAD_COMMIT_TC );
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;
   else
      put "Transaction has been committed.";
      put ;

   /*-----------------------------------------------------------------*/
   /* The ACT calls below will be made without a transaction handle. */
   /* Changes will be immediate.                                      */
   /* This code shows how to change the description of an ACT        */
   /*-----------------------------------------------------------------*/

   /* Get the Desc attribute */
```

```
   attrvalue  = "";
   rc = METASEC_GETACTA("",act_uri,"Desc", attrvalue);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;
   else
      put "Existing ACT Description:" attrvalue;


   /* change the ACT description */
   rc = METASEC_SETACTA("",act_uri,"Desc",
                        "ACT that denies PUBLIC and grants SASUSERS");
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   /* Get the Desc attribute */
   attrvalue  = "";
   rc = METASEC_GETACTA("",act_uri,"Desc", attrvalue);
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;
   else
      put "New ACT Description:" attrvalue;

   /* list all the attributes on the ACT */
   put "These are the new attributes on the ACT:";

   n=1;
   rc =1;
   do while (rc > 0) ;
      attrname="";
      attrvalue="";
      rc=metasec_getnacta("", act_uri, n, attrname, attrvalue);
      if (rc < 0 ) then do;
         sysmsg = sysmsg();
         put sysmsg;
         end;
      else do;
         put "Attribute #" n "Name=" attrname "Value=" attrvalue;
         n=n+1;
         end;
      end;

run;
```

If you issue the METABROWSE command to open the Metadata Browser window, you can see the new ACT, "My Demo ACT for METASEC." It is associated with the new table, "My Demo Table 2 for METASEC."

The following code shows how to remove the ACT from the object. The calls in the code are submitted without a transaction context, so the changes are made immediately.

With METASEC_REMACT, you must specify the ID instance form of URI for the ACT. Use the METADATA_RESOLVE function to find the ID. You can specify the search form for the object from which you remove the ACT.

```
data _null_;
   length type $60
          id $17;
   type='';
   id='';
   rc=metadata_resolve("omsobj:AccessControlTemplate?@Name='Grant
SASUSERS
ACT'",
                       type,id);
   rc2 = METASEC_REMACT("",
                       "omsobj:PhysicalTable?@Name='Demo Table 2 for
METASEC'",
                       "omsobj:AccessControlTemplate/"||id,
                       "0");
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;
run;
```

If you look at the Metadata Browser window again, you can see that the ACT has been removed from the table.

The following code deletes the table, the ACT, and the person by name, with the search form of URI. The calls in the code are submitted without a transaction context, so the changes are made immediately. Because the PUBLIC user group was denied access to the ACT earlier, only the unrestricted user can perform this task. Administrative access is required to add and delete users.

```
options metaserver="myserver"
metaport=8561
metauser="sasadm@saspw"
metapass="adminpwd"
metarepository="Foundation";

data _null_;
   rc=metadata_delobj("omsobj:PhysicalTable?@Name='Demo Table 2 for
METASEC'");

   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   rc=metadata_delobj("omsobj:AccessControlTemplate?@Name='Grant
SASUSERS
ACT'");
   if (rc < 0 ) then do;
      sysmsg = sysmsg();
      put sysmsg;
      end;

   rc=metadata_delobj("omsobj:Person?@Name='Demo User for METASEC'");
   if (rc < 0 ) then do;
```

```
      sysmsg = sysmsg();
      put sysmsg;
      end;
run;
```

18

DATA Step Functions for Metadata Security Administration

Dictionary

METASEC_APPLYACT Function

Applies an ACT to an object.

Syntax

rc = **METASEC_APPLYACT**(*tc*, *uri*, *act_uri,flags*);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***uri* (in)**
specifies a character variable or constant that contains the URI of the object to which you are applying the ACT.

***act_uri* (in)**
specifies a character variable or constant that contains the URI of the ACT that you are applying to the object; use the following form of URI: "omsobj:AccessControlTemplate/*xxxxxxxx.yyyyyyyy*".

***flags* (in)**
not currently used; set to 0 (zero).

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method ApplyACTToObj(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See "Example: Working with ACTs" on page 257 for a usage example.

METASEC_BEGTRAN Function

Begins the TC.

Syntax

rc = METASEC_BEGTRAN(*uri*, *flags*, *tc*);

Required Arguments

***uri* (in)**
specifies a character variable or constant that contains the URI of the object to be manipulated by the transaction, or an empty string if the object is not

immediately known. When an empty string is used, identify the target resource in the METASEC_ENDTRAN function.

***flags* (in)**
not currently used; set to 0 (zero).

***tc* (out)**
returns a character variable that contains the handle of the new TC; must be at least $16.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Output variable for TC is too small to hold the TC handle.

-3
No objects match the specified URI.

-4
Numeric value (*flag*) exceeds the maximum usable value.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method BeginTransactionContext(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See "Example: Begin and End Transaction Context" on page 256 for a usage example.

See Also

Functions

- "METASEC_ENDTRAN Function" on page 271

METASEC_DELACT Function

Deletes ACT from the metadata server.

Syntax

rc = METASEC_DELACT(*tc*, *act_uri*);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context; if *tc* is returned from the METASEC_BEGTRAN function, then *tc* references an existing ACT.

***act_uri* (in)**
specifies a character variable or constant that contains the URI of the ACT that you are deleting; use the following form of URI: “omsobj:AccessControlTemplate/*xxxxxxxx.yyyyyyyy*”.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
ACT was not deleted; see sysmsg() for information.

Details

When the ACT is deleted, any associations are also deleted.

This function calls the ISecAdmin method DestroyAccessControlTemplate(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See “Example: Working with ACTs” on page 257 for a usage example.

See Also

Functions

- “METASEC_GETACTA Function” on page 272
- “METASEC_GETNACT Function” on page 273
- “METASEC_NEWACT Function” on page 284
- “METASEC_SETACTA Function” on page 287

METASEC_ENDTRAN Function

Ends the TC.

Syntax

rc = **METASEC_ENDTRAN**(*uri*, *tc*, *flags*);

Required Arguments

***uri* (in)**
specifies a character variable or constant that contains the URI of the object to be manipulated by the transaction.

***tc* (in)**
specifies a character variable that contains the handle of the TC to be ended.

***flags* (in)**
specifies an integer bit field that specifies whether the transaction should be committed. Use one of the following macro variables from %MDSECCON() or an integer. A value is required. The function will return an error if you do not specify a value.

_SECAD_COMMIT_TC (1)	Commit transaction.
_SECAD_DISCARD_TC (2)	Do not commit transaction.

For more information, see “Using the %MDSECCON() Macro” on page 255.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-3
No objects match the specified URI.

-4
Numeric value (*flag*) exceeds the maximum usable value.

-5
No TC handle was specified.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method EndTransactionContext(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See “Example: Begin and End Transaction Context” on page 256 for a usage example.

See Also

Functions

- “METASEC_BEGTRAN Function” on page 268

METASEC_GETACTA Function

Returns an ACT attribute.

Syntax

rc = **METASEC_GETACTA**(*tc*, *act_uri*, *attr*, *attr_value*);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context; if *tc* is returned from the METASEC_BEGTRAN function, then *tc* references an existing ACT.

***act_uri* (in)**
specifies a character variable or constant that contains the URI of the ACT that is requested; can be blank if the ACT was specified when creating the TC; use the following form of URI: “omsobj:AccessControlTemplate/*xxxxxxxx.yyyyyyyy*”.

***attr* (in)**
specifies a character variable that specifies the ACT attribute whose value you are requesting; see Details for more information.

attr_value
returns a character variable that contains the value of the ACT attribute; see Details for more information.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-99 or less
Attribute is not set; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method GetAccessControlTemplateAttribs(). For information about this method, see *SAS Open Metadata Interface: Reference and Usage*.

Lowercase or mixed-case ACT attributes (`Name`, `Desc`, `Use`) are automatically uppercased (`NAME`, `DESC`, `USE`). The following table provides more information about the *attr* and *attr_value* arguments.

Table 18.1 *ACT Attribute Specifications*

Attribute	Attribute Value	Maximum Length of Attribute Value	Notes
`NAME`	Character string	60	Optional
`DESC`	Character string	200	Optional
`USE`	`REPOS` or empty string	5	Optional; the `REPOS` value indicates that the specified ACT is the default repository ACT

See “Example: Working with ACTs” on page 257 for a usage example.

See Also

Functions

- “METASEC_DELACT Function” on page 269
- “METASEC_GETNACTA Function” on page 275
- “METASEC_NEWACT Function” on page 284
- “METASEC_SETACTA Function” on page 287

METASEC_GETNACT Function

Returns the *n*th ACT.

Syntax

rc = **METASEC_GETNACT**(*tc*, *uri*, *n*, *act_uri*, *name*, *desc*, *use*<, *flags*);>

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***uri* (in)**
specifies a character variable or constant that contains the URI of the object from which you want to return the ACTs.

If the URI is for a repository (in the form "ReposID:*xxxxxxxx*"), then the metadata server function GetAccessControlTemplateList() is called to obtain the ACT information.

If the URI is for an object (in the form "omsobj:*ObjectType*/*xxxxxxxx*.*yyyyyyyy*"), then GetACTsOnObj() is called.

Search syntax is supported, such as "omsobj:*ObjectType*?@Name='*My Object*' ".

***n* (in)**
specifies a one-based numeric index value that indicates which row to return from the array. For information, see "Array Parameters" on page 193.

***act_uri* (out)**
returns a character variable that contains the URI of the ACT that is requested; this URI is in the following form: "omsobj:AccessControlTemplate/*xxxxxxxx*.*yyyyyyyy*".

***name* (out)**
returns a character variable that contains the name of the *n*th ACT.

***desc* (out)**
returns a character variable that contains the description of the *n*th ACT.

***use* (out)**
returns a character variable that contains the value of the USE attribute of the *n*th ACT. If the ACT is for a repository, the returned value is "REPOS". Otherwise, the returned value is an empty string.

Optional Argument

***flags* (in)**
specifies an optional integer bit field. If the *uri* argument is in the form ReposID:*yyyyyyyy* (that is, GetAccessControlTemplateList() is called), you can use the following macro variable from %MDSECCON();

_SECAD_REPOS_DEPENDENCY_USES (16)	Return all ACTs in all SAS Metadata Repositories that are not of type PROJECT.

For more information, see "Using the %MDSECCON() Macro" on page 255.

Return Values

0
Successful completion, but no ACTs are found to be applied to the object.

-1
Unable to connect to the metadata server.

-2
Error returning the ACT list; see log or sysmsg() for information.

-3
No objects match the specified URI.

-4
Numeric value (*n*) exceeds the maximum usable value.

-5
n is out of range.

-99 or less
Other error; see log or sysmsg() for information.

Details

If the *uri* argument represents a repository, then the ACTs in the repository are returned. If *uri* does not represent a repository, then the ACTs that protect the object are returned.

This function calls the ISecAdmin method GetAccessControlTemplateList() or GetACTsOnObj(), depending on the form of the URI in the *uri* argument. For information about the methods, see *SAS Open Metadata Interface: Reference and Usage*.

See "Example: Working with ACTs" on page 257 for a usage example.

See Also

Functions

- "METASEC_APPLYACT Function" on page 267
- "METASEC_GETNAUTH Function" on page 277
- "METASEC_GETNID Function" on page 282
- "METASEC_REMACT Function" on page 286
- "METASEC_SETAUTH Function" on page 289

METASEC_GETNACTA Function

Returns the *n*th attribute for an ACT.

Syntax

rc = METASEC_GETNACTA(*tc*, *act_uri*, *n*, *attr*, *attr_value*);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context. If *tc* is returned from the METASEC_BEGTRAN function, then *tc* references an existing ACT.

***act_uri* (in)**
specifies a character variable that contains the URI of the ACT that is requested; this URI is in the following form: "omsobj:AccessControlTemplate/*xxxxxxxx.yyyyyyyy*".

***n* (in)**
One-based numeric index value that indicates which row to return from the array. For more information, see "Array Parameters" on page 193.

***attr* (out)**
returns a character variable that contains the name of the *n*th attribute found on the ACT; see Details for more information.

***attr_value* (out)**
returns a character variable that contains the value of the *n*th attribute found on the ACT; see Details for more information.

Return Values

0
Successful completion, but no ACTs are found.

-1
Unable to connect to the metadata server.

-4
Numeric value (*n*) exceeds the maximum usable value.

-5
n is out of range.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method GetAccessControlTemplateAttribs(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

The following table provides more information about the *attr* and *attr_value* arguments.

Table 18.2 ACT Attribute Specifications

Attribute	Attribute Value	Maximum Length of Attribute Value	Notes
NAME	Character string	60	
DESC	Character string	200	
USE	REPOS or empty string	5	The REPOS value indicates that the specified ACT is the default repository ACT

See "Example: Working with ACTs" on page 257 for a usage example.

See Also

Functions

- "METASEC_DELACT Function" on page 269
- "METASEC_GETACTA Function" on page 272
- "METASEC_NEWACT Function" on page 284
- "METASEC_SETACTA Function" on page 287

METASEC_GETNAUTH Function

Returns the *n*th authorization for an object.

Syntax

rc=**METASEC_GETNAUTH**(*tc*, *uri*, *n*, *type*, *name*, *auth*, *perm*, *cond* < , *flags* >< , *display* >)

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***uri* (in)**
specifies a character variable or constant that contains the URI of the object that is requested; can be an empty string " " if *tc* is specified. You can optimize performance by using the following form of URI:

```
omsobj: metatype/identifier.identifier
```

***n* (in)**
specifies a one-based numeric index value that indicates which row to return from the array. For more information, see "Array Parameters" on page 193.

***type* (in/out)**
specifies a character variable that contains the identity type. Valid values are IdentityGroup or Person. If this argument is empty, all identity types associated with authorizations for the object are returned. Can be a comma-delimited list that is parallel to a list for the *name* argument.

***name* (in/out)**
specifies a character variable that contains the identity name, which must be unique for every identity of that type on the metadata server. If this argument is empty, all identities associated with authorizations for the object are returned. Can be a comma-delimited list that is parallel to a list for the *type* argument; for more information, see "About the in/out Arguments" on page 281.

***auth* (out)**
returns an integer bit field that indicates grant or deny, and the origin of the grant or deny. You can use macro variables from %MDSECCON() to translate the integer into a recognizable message. For more information, see "Authorizations and the %MDSECCON() Macro" on page 279.

***perm* (in/out)**
For input, specifies an optional, comma-delimited list of permission names for which authorizations are requested. For more information, see "About the in/out Arguments" on page 281. If this argument is empty, all available permissions are returned.

For output, returns a character variable that contains the name of the permission whose grant or deny state is specified in the *auth* argument.

***cond* (out)**
returns a character variable that contains the condition if a grant permission is conditional; can be very long, so if this argument is too short, the value is truncated.

Optional Arguments

***flags* (in)**
specifies an optional integer bit field. You can use one of the following macro variables from %MDSECCON() or an integer:

_SECAD_ACT_CONTENTS (4)	Return the authorizations that define the contents of an ACT when the *tc* or *uri* argument references an ACT.
_SECAD_DO_NOT_RETURN_PERMCOND (8)	Do not return any available values for the *cond* argument.

For more information, see “Using the %MDSECCON() Macro” on page 255.

***display* (out)**
specifies a character variable that contains the value of the DisplayName attribute, if the identity has a DisplayName attribute.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Error parsing *type* or *name* input list.

-3
No objects match the specified URI.

-4
Numeric value (flag) exceeds the maximum usable value.

-5
n is out of range.

-99 or less
Other error; see log or sysmsg() for information.

Details

Origin

This function calls the ISecAdmin method GetAuthorizationsOnObj(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

Authorizations and the %MDSECCON() Macro

The *auth* parameter of the METASEC_GETNAUTH function returns an integer that indicates grant or deny and the origin of the grant or deny. Here are the authorizations that might be returned by the METASEC_GETNAUTH function, their integer values, associated macro variables, and descriptions.

Table 18.3 *Explicit, ACT, and Indirect Authorizations and Masks*

Authorization Type	Integer value	Macro Variable	Description
Explicit deny	1	_SECAD_PERM_EXPD	Explicit deny that originates from the authorization that is directly associated with the object

Authorization Type	Integer value	Macro Variable	Description
Explicit grant	2	_SECAD_PERM_EXPG	Explicit grant that originates from the authorization that is directly associated with the object
ACT deny	4	_SECAD_PERM_ACTD	Deny that originates from an ACT other than the default ACT
ACT grant	8	_SECAD_PERM_ACTG	Grant that originates from an ACT other than the default ACT
Indirect deny	16	_SECAD_PERM_NDRD	Indirect deny that originates from an IdentityGroup membership, through inheritance, or from the default ACT; an indirect value is always returned
Indirect grant	32	_SECAD_PERM_NDRG	Indirect grant that originates from an IdentityGroup membership, via inheritance, or from the default ACT; an indirect value is always returned

To simplify usage, you can use macro variables from %MDSECCON() instead of the integer values. The macro variables in the table are set to the integer values by %MDSECCON. For more information, see "Using the %MDSECCON() Macro" on page 255. For suggested usage, see "Example: Working with ACTs" on page 257.

Masks

The following masks are provided to test whether a METASEC_GETNAUTH output value applies to a given authorization category. A mask is a filter that returns an indirect result. A bitwise AND between the mask and a value within that mask produces the input value; otherwise, it produces a zero.

Table 18.4 *Explicit, ACT, and Indirect Authorizations and Masks*

Mask Type	Integer value	Macro Variable	Description
Explicit mask	3	_SECAD_PERM_EXPM	Mask to extract explicit value that originates from the authorization that is directly associated with the object
ACT mask	12	_SECAD_PERM_ACTM	Mask to extract indirect value that originates from an ACT other than the default ACT

Mask Type	Integer value	Macro Variable	Description
Indirect mask	48	_SECAD_PERM_NDRM	Mask to extract indirect value that originates from an IdentityGroup membership, via inheritance, or from the default ACT; an indirect value is always returned.

The masks can be used with the BAND function. Here is example code that illustrates how the masks can be used:

```
rc=metasec_getnauth("",objuri,n,
                    identitytypes,identitynames,
                    auth,tmppermissions,condition,
                     &_SECAD_RETURN_ROLE_TYPE, identitydispname);
...
authint = input(auth, 16.);
...
if (band(authint, &_SECAD_PERM_EXPM) ) then do;
            if (band(authint,&_SECAD_PERM_EXPD)) then
               authorization = "Denied Explicitly";
            else
               authorization = "Granted Explicitly";
            end;
         else if (band(authint, &_SECAD_PERM_ACTM) ) then do;
            if (band(authint,&_SECAD_PERM_ACTD)) then
               authorization = "Denied by ACT";
            else
               authorization = "Granted by ACT";
            end;
         else if (band(authint,&_SECAD_PERM_NDRM) ) then do;
            if (band(authint,&SECAD_PERM_NDRD)) then
               authorization = "Denied Indirectly";
            else
               authorization = "Granted Indirectly";
            end;
...
```

About the in/out Arguments

Some of this function's arguments are in/out. After the first call for the specified URI, the in/out parameters do not need to be reset to the initial calling value. Subsequent calls retrieve the output values from the cache, and place them in the output variable without consideration of the value when the call was made. In other words, after the first call is made for the URI, the metadata server ignores the input aspect of the in/out parameters.

Here is an example of comma-delimited lists for *type* and *name* arguments:

```
type = "person,person,person";
name = "Fred,Yolanda,Viktorija";

rc = metasec_getnauth(tc,uri,n,type,name,auth,permission,cond);
```

See Also

Metadata Security Administration Functions

- “METASEC_APPLYACT Function” on page 267
- “METASEC_GETNACT Function” on page 273
- “METASEC_GETNID Function” on page 282
- “METASEC_REMACT Function” on page 286
- “METASEC_SETAUTH Function” on page 289

Other Functions

- “BAND Function” in *SAS Functions and CALL Routines: Reference*

METASEC_GETNID Function

Returns the *n*th identity for an object. Identities can come directly from the object, from the inheritance parents, from the default ACT, and from any ACTs that are directly associated with the object.

Syntax

rc = **METASEC_GETNID**(*tc*, *uri*, *n*, *type*, *name*, *flags*<, *display*, *origin*);>

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***uri* (in)**
specifies a character variable or constant that contains the URI of the object to be manipulated by the transaction; can be an empty string “ ” if *tc* is specified.

***n* (in)**
specifies a one-based numeric index value that indicates which row to return from the array. For more information, see “Array Parameters” on page 193.

***type* (out)**
returns a character variable that contains the identity type. The variable should be large enough to store the two available values, IdentityGroup or Person. (It should probably at least $13.)

***name* (out)**
returns a character variable that contains the identity name, which must be unique for every identity of that type on the SAS Metadata Server.

***flags* (in)**
specifies an optional integer bit field. You can use one of the following macro variables from %MDSECCON() or an integer:

_SECAD_ACT_CONTENTS (4)	If the *uri* argument references an ACT, returns the identities that define the ACT, and not the identities from the access controls that protect the ACT.
_SECAD_RETURN_ROLE_TYPE (32)	Returns `roles` as the type for IdentityGroups that are acting as roles.

For more information, see "Using the %MDSECCON() Macro" on page 255.

Optional Arguments

***display* (out)**
returns an optional character variable that contains the value of the DisplayName attribute if the identity has a DisplayName attribute.

***origin* (out)**
indicates where the identity originates for the object in security:

D	The identity originates from an ACT or ACE that is directly attached to the object.
I	The identity originates from inheritance.
DI	The identity originates from inheritance, but the identity is involved with direct access controls on the object.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-2
Error returned from the metadata server; see log or sysmsg() for information.

-3
No objects match the specified URI.

-4
Numeric value (flag) exceeds the maximum usable value.

-5
n is out of range.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method GetIdentitiesOnObject(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See “Example: Working with ACTs” on page 257 for a usage example.

See Also

Functions

- “METASEC_APPLYACT Function” on page 267
- “METASEC_GETNACT Function” on page 273
- “METASEC_GETNAUTH Function” on page 277
- “METASEC_REMACT Function” on page 286
- “METASEC_SETAUTH Function” on page 289

METASEC_NEWACT Function

Creates a new ACT.

Syntax

rc = **METASEC_NEWACT**(*tc*, *repos_uri*, *attr*, *attr_value*<, attr*n*, attr_value*n*>);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***repos_uri* (in)**
specifies a character variable or constant that contains the URI of the repository where you are creating the ACT; use the following form of URI: “Reposid:*xxxxxxxx*”.

***attr* (in)**
specifies a character variable or constant that specifies an ACT attribute. You must pair this argument with an *attr_value* argument, and you can specify up to three *attr* and *attr_value* pairs. See “Details” on page 285 for more information.

***attr_value* (in)**
specifies a character variable or constant that contains the value of an ACT attribute; you must pair this argument with an *attr* argument, and you can specify up to three *attr* and *attr_value* pairs. See “Details” on page 285 for more information.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method CreateAccessControlTemplate(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

Lowercase or mixed-case ACT attributes (`Name`, `Desc`, `Use`) are automatically uppercased (`NAME`, `DESC`, `USE`). The following table provides more information about the *attr* and *attr_value* arguments.

Table 18.5 *ACT Attribute Specifications*

Attribute	Attribute Value	Maximum Length of Attribute Value	Notes
`NAME`	Character string	60	Required; the attribute value must be unique within the repository
`DESC`	Character string	200	Optional
`USE`	`REPOS` or empty string	5	Optional; when you specify `REPOS`, the ACT becomes the new default repository ACT; see the following caution.

CAUTION
Passing in an empty string for the USE attribute is not recommended. Passing in an empty string has an effect only when the ACT already has `USE=REPOS`. However, setting a repository ACT's USE attribute to a blank leaves the repository in a default mode where all permissions are granted. If you want to change the default ACT, it is recommended that you set USE=REPOS on the ACT that you want to use as the repository ACT. The metadata server automatically removes the USE=REPOS attribute from the previous repository ACT. Thus, the repository is not left in a mode with no repository ACT.

See "Example: Working with ACTs" on page 257 for a usage example.

See Also

Functions

- “METASEC_DELACT Function” on page 269
- “METASEC_GETACTA Function” on page 272
- “METASEC_GETNACTA Function” on page 275
- “METASEC_SETACTA Function” on page 287

METASEC_REMACT Function

Removes an ACT from an object.

Syntax

rc = **METASEC_REMACT**(*tc*, *uri*, *act_uri*, flags);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***uri* (in)**
specifies a character variable or constant that contains the URI of the object from which you want to remove the ACT.

***act_uri* (in)**
specifies a character variable or constant that contains the URI of the ACT that you are removing; use the following form of URI: “omsobj:AccessControlTemplate/*xxxxxxxx.yyyyyyyy*”.

***flags* (in)**
not currently used; set to 0 (zero).

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method RemoveACTFromObj(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See “Example: Working with ACTs” on page 257 for a usage example.

See Also

Functions

- “METASEC_APPLYACT Function” on page 267
- “METASEC_GETNACT Function” on page 273
- “METASEC_GETNAUTH Function” on page 277
- “METASEC_GETNID Function” on page 282
- “METASEC_SETAUTH Function” on page 289

METASEC_SETACTA Function

Sets an ACT attribute.

Syntax

rc = **METASEC_SETACTA**(*tc*, *act_uri*, *attr*, *attr_value*);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context. If *tc* is returned from the METASEC_BEGTRAN function, then *tc* references an existing ACT.

***act_uri* (in)**
specifies a character variable or constant that contains the URI of the ACT that you are modifying; can be blank if the ACT was specified when creating the TC. Use the following form of URI: ”omsobj:AccessControlTemplate/*xxxxxxxx.yyyyyyyy*”.

***attr* (in)**
specifies a character variable that specifies the ACT attribute that you are setting; see Details.

***attr_value* (out)**
returns a character variable that contains the value of the ACT attribute that you are setting; any specified attribute values replace the current values for the ACT. See Details.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-99 or less
Attribute is not set; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method SetAccessControlTemplateAttribs(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

Lowercase or mixed-case ACT attributes (`Name`, `Desc`, `Use`) are automatically uppercased (`NAME`, `DESC`, `USE`). The following table provides more information about the *attr* and *attr_value* arguments.

Table 18.6 *ACT Attribute Specifications*

Attribute	Attribute Value	Maximum Length of Attribute Value	Notes
`NAME`	Character string	60	Optional; the attribute value must be unique within the repository.
`DESC`	Character string	200	Optional
`USE`	`REPOS` or empty string	5	Optional; when you specify `REPOS`, the ACT becomes the new default repository ACT. When you specify an empty string, the ACT is removed from being the default repository ACT. See the following caution.

CAUTION

Passing in an empty string for the USE attribute is not recommended. Passing in an empty string has an effect only when the ACT already has `USE=REPOS`. However, setting a repository ACT's USE attribute to a blank leaves the repository in a default mode where all permissions are granted. If you want to change the default ACT, it is recommended that you set USE=REPOS on the ACT that you want to use as the repository ACT. The metadata server automatically removes the USE=REPOS attribute from the previous repository ACT. Thus the repository is not left in a mode with no repository ACT.

See "Example: Working with ACTs" on page 257 for a usage example.

See Also

Functions

- "METASEC_DELACT Function" on page 269
- "METASEC_GETACTA Function" on page 272
- "METASEC_GETNACTA Function" on page 275
- "METASEC_NEWACT Function" on page 284

METASEC_SETAUTH Function

Sets authorization for an object.

Syntax

rc = **METASEC_SETAUTH**(*tc*, *uri*, *type*, *name*, *auth*, *perm*, *cond*<, *flags*>);

Required Arguments

***tc* (in)**
specifies a transaction context handle; can be an empty string " " to invoke with a temporary context.

***uri* (in)**
specifies a character variable or constant that contains the URI of the object to be manipulated by the transaction; can be an empty string " " if transaction context is specified.

***type* (in)**
specifies a character variable or string constant that contains the identity type; the variable should be large enough to store the two available values, IdentityGroup or Person, probably at least $13.

***name* (in)**
specifies a character variable that contains the identity name.

***auth* (in)**
specifies a character variable that indicates the authorization to set for the permission and identity (which are specified in the *perm* and *name* arguments, respectively). Specify one of the following values:

G Grant

D Deny

R Remove

***perm* (in)**
specifies a character variable that contains the name of the permission whose grant, deny, or remove state is specified in the *auth* argument.

***cond* (in)**
specifies a character variable that contains the condition if a grant permission is conditional. The value can be very long, so if this argument is too short, the value is truncated. The permissions are case sensitive and must match the case of the permissions that are defined in the metadata server.

Optional Argument

***flags* (in)**
Optional bit field. You can use one of the following macro variables from %MDSECCON() or an integer:

_SECAD_ACT_CONTENTS (4)	Return the authorizations that define the contents of an ACT when the *tc* or *uri* argument references an ACT.

For more information, see “Using the %MDSECCON() Macro” on page 255.

Return Values

0
Successful completion.

-1
Unable to connect to the metadata server.

-3
No objects match the specified URI.

-99 or less
Other error; see log or sysmsg() for information.

Details

This function calls the ISecAdmin method SetAuthorizationsOnObj(). For information about the method, see *SAS Open Metadata Interface: Reference and Usage*.

See “Example: Working with ACTs” on page 257 for a usage example.

See Also

Functions

- “METASEC_APPLYACT Function” on page 267
- “METASEC_GETNACT Function” on page 273
- “METASEC_GETNAUTH Function” on page 277
- “METASEC_GETNID Function” on page 282
- “METASEC_REMACT Function” on page 286