# SAS® 9.3 Language Interfaces to Metadata

# Contents

# What's New in the SAS 9.3 Language Interfaces to Metadata

## Overview

Changes and enhancements include the following:

- a new METHOD argument for PROC METADATA

- new options for PROC METAOPERATE ACTION=REFRESH in support of the new metadata server backup facility

- a new option for PROC METAOPERATE PAUSE and RESUME actions in support of the new metadata server backup facility

- a new option for PROC METAOPERATE ACTION=REFRESH to enable metadata server alert e-mail testing

- PROC METAOPERATE no longer requires the <SERVER/> option to be specified with ACTION=REFRESH

- the METAAUTORESOURCES system option now assigns the LIBNAME engine based on a pre-assignment type in the library definition

- a new SPN format for the METASPN system option

- documentation changes

## Procedures

The METADATA procedure is enhanced as follows:

- Depending on the value of a new METHOD= argument, DOREQUEST or STATUS, the METADATA procedure submits either a SAS Open Metadata Interface IOMI DoRequest or IServer Status method call to the SAS Metadata Server. Support for METHOD=STATUS is important because the DoRequest method (the legacy behavior) does not work when the SAS Metadata Server is paused. Using METHOD=STATUS, PROC METADATA can be used to get metadata server configuration, backup information, and various server statistics while the server is paused. For more information, see Chapter 11, "METADATA Procedure," on page 67.

The METAOPERATE procedure is enhanced as follows:

- The REFRESH action has several new options in support of the new metadata server backup facility

<BACKUP *attribute(s)*/>
> invokes an ad hoc backup of the SAS Metadata Server to the location indicated in the server's backup configuration.

<BACKUPCONFIGURATION *attribute(s)*/>
> modifies the value of the specified backup configuration attribute. Backup configuration attributes are BackupLocation="*directory*", RunScheduledBackups="Y | N", and DaysToRetainBackups="*number*".

<RECOVER *options*/>
> recovers the SAS Metadata Server from the specified backup, and can perform roll-forward recovery from the metadata server journal. The roll-forward feature recovers all journal transactions, or transactions up to a specified point in time.

<SCHEDULE EVENT="Backup" WEEKDAY*n*="*time*R"/>
> sets or modifies the server backup schedule. SCHEDULE EVENT="Backup" specifies the event that will be scheduled. WEEKDAY*n*="*time*" specifies the backup schedule. The SAS 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.; 1300 is 1 p.m. To modify the schedule, specify the appropriate WeekDay*n*= attribute with the backup time. R can be used to specify that a REORG be performed with a backup.

<SCHEDULER/>
> rebuilds or restarts the backup scheduler thread, depending on the XML subelement that is specified.

<OMA ALERTEMAILTEST="*text*"/>
> sends a test alert e-mail message to the address configured in the <OMA ALERTEMAIL="*email-address*"/> option in the metadata server's omaconfig.xml configuration file. The option is provided for testing the metadata server's alert e-mail notification subsystem. The subsystem sends an alert e-mail message to configured recipients whenever a server backup or recover fails, or when the server itself fails.

- The PAUSE and RESUME actions support a new <FORCE/> option. <FORCE/> regains control of the SAS 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. When used with PAUSE, you can include the <SERVER STATE="ADMIN"/> option to enable administrators to examine the recovered system before making the server available to clients.

For more information, see .

## System Options

- The METAAUTORESOURCES system option now assigns the LIBNAME engine based on a pre-assignment type setting in the library definition. Libraries that are marked as being assigned by external configuration (AUTOEXEC file) are ignored by METAAUTORESOURCES. Libraries that are marked as being assigned by the native library engine are assigned by the library engine defined for that library in metadata. Libraries that are marked as being assigned by the metadata LIBNAME

engine are assigned with the metadata LIBNAME engine (MLE). For more information, see "METAAUTORESOURCES System Option" on page 31.

- The SPN format for the METASPN system option has changed. The following formats are supported: SAS/*machine-name*, or SAS/*machine-name*.company.com. For more information, see "METASPN= System Option" on page 42.

## Documentation Enhancements

- A new chapter describes how SAS language elements that read and write metadata are affected by the SAS type dictionary. See "Using Language Elements That Read and Write Metadata" on page 7.

- Additional examples have been added that show how to use SAS metadata DATA step functions to create reports that track the data libraries, servers, users, user group memberships, and logins defined in metadata. See "Examples: DATA Step Functions for Reading Metadata" on page 144.

# Accessibility Features of the SAS Language Interfaces to Metadata

## Overview

The SAS language interfaces to metadata are command-line interfaces that are accessible using a keyboard or alternative keyboard assistive technologies. For this release, no accessibility testing was done and no additional features were added to address accessibility. If you have specific questions about the accessibility of SAS products, send them to **accessibility@sas.com** or call SAS Technical Support.

# Recommended Reading

- *SAS Intelligence Platform: Data Administration Guide*

- *SAS Intelligence Platform: Overview*

- *SAS Intelligence Platform: Security Administration Guide*

- *SAS Intelligence Platform: System Administration Guide*

- *SAS Language Reference: Concepts*

- *SAS Functions and CALL Routines: Reference*

- *SAS 9.3 Metadata Model: Reference*

- *SAS Open Metadata Interface: Reference and Usage*

- *SAS XML LIBNAME Engine: User's Guide*

For a complete list of SAS publications, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

*Part 1*

# Introduction

*Chapter 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.

*Chapter 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.

To use SAS language elements to create or read any metadata object, you must know the SAS Metadata Model metadata type that represents the object in the SAS Metadata Repository. You must know the attributes and associations defined for the metadata type in the SAS Metadata Model. 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 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.

SAS 9.3 includes a type dictionary that SAS Intelligence Platform applications and solutions use to standardize the usage of common and shared resources and information assets in their applications or solutions. An aspect of this type dictionary is that for resources that are persisted in metadata, it standardizes their logical metadata definitions.

This chapter describes how the type dictionary affects read and write requests made with SAS language elements.

## Objects Included in the Dictionary

The type dictionary includes metadata objects that describe common and shared resources and information assets, as well as metadata objects that need to be displayed in the SAS Management Console **Folders** tree, and metadata 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, to name a few.

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

## What is the SAS Type Dictionary?

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

A type definition is metadata that contains the information that is necessary to display and manage instances of an object type in a SAS application. Whereas an object's *logical metadata definition* contains information that describes a resource or information asset, and this definition might configure the resource in the enterprise, the *type definition* contains information that an application can use to display and manage the object, and this definition represents the resource within the application.

A goal of the type dictionary is to hide the details of logical metadata definitions from clients. 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. The primary metadata objects in logical metadata definitions that conform to the type dictionary store the name of their type definition in a PublicType= attribute. The SAS Open Metadata Interface GetMetadata method supports a flag that clients can set to instruct the SAS Metadata Server to use the type definition referenced in the PublicType= attribute to expand an object's logical metadata definition.

## How the Type Dictionary Affects SAS Language Elements

### *Creating Metadata*

In SAS 9.3, you should use SAS wizards and procedures to automate the creation of metadata, such as PROC METALIB and PROC OLAP. This is opposed to using PROC METADATA and metadata DATA step functions to create metadata. By using SAS wizards and procedures, you can be assured that the logical metadata definitions conform to the type dictionary. A client 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 display 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, which is new.

### Reading Metadata

Metadata language elements that request specific object instances require you to identify the object instance by its primary metadata type and metadata identifier or name. As long as you are using the correct metadata type and identifier or name, no additional information is required to retrieve the specified object instance.

For SAS language elements that read metadata, the type dictionary makes it easy to identify the primary metadata type representing the object in the SAS Metadata Repository. Open the type definition of the object that you are interested in in the dictionary. The object's primary metadata type is specified in the **Metadata Type** field on the **Advanced** tab of its properties. For information about metadata identifiers or names, see "What Is a Metadata Identifier?" on page 11. Also, see "Obtaining Metadata Names and Identifiers" on page 11.

PROC METADATA enables you to get specific object instances by submitting the SAS Open Metadata Interface GetMetadata method. Clients that use PROC METADATA can set the OMI_FULL_OBJECT flag in the GetMetadata request to return an object's full logical metadata definition. Metadata DATA step functions do not support the ability to return an object's full logical metadata definition. For an example of a PROC METADATA request that sets the OMI_FULL_OBJECT flag, see "Example 7: Request the Metadata for One Object" on page 85.

Metadata requests that list object instances should use the values specified in the type definition's **MetadataType** and **TypeName** fields. Many type definitions use the same metadata type as their primary metadata object (for example, Information Map and SAS Report). The **TypeName** value is unique across type definitions. The **TypeName** value is specified on the **Advanced** tab of a type definition's properties. The **TypeName** value maps to the PublicType= attribute of the primary metadata object in logical metadata definitions that conform to the type dictionary.

For an example of a PROC METADATA request that uses the type dictionary to list objects, see "Example 8: Request the Metadata for One Type of Object" on page 88.

SAS provides the METADATA_GETNOBJ function for getting metadata objects in a SAS Metadata Repository. The METADATA_GETNOBJ function and many other 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 12. Also, see "METADATA_GETNOBJ Function" on page 170.

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 178.

### *Deleting Metadata*

SAS metadata interfaces automatically uses 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.

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*.

*Chapter 3*

# Metadata Object Identifiers and URIs

## What Is a Metadata Identifier?

The SAS Metadata Server uses a unique identifier for every metadata object. 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. 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 182 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 13 .

# 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.

*Chapter 4*

# Examples: Using Metadata Language Elements to Create Reports

## Overview of the Examples

These examples 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.

## 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 outputs the values in an XML file. Then, the XML LIBNAME engine uses an XML map 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.

- 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 XML map, created by a DATA step

- two SAS data sets, created by the XML LIBNAME engine and an XML map

- 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 67. The XML LIBNAME engine and XML maps are not documented in this book; see *SAS XML LIBNAME Engine: 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 XML map. 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 XML map is created in the following code to illustrate the process. You can use a graphical interface, SAS XML Mapper, to generate an XML map. For more information, see *SAS XML LIBNAME Engine: 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 .

# 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 144.

*Part 2*

# System Options

*Chapter 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= |
| | 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;
```

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, SAS OLAP Server, and SAS Table 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 28.

- Specify a stored metadata server connection profile with the METACONNECT= and METAPROFILE options; see "Specifying a Stored Connection Profile" on page 29.

- Specify connection properties when you issue a metadata procedure; see "Overview of Procedures for Metadata" on page 63.

- Specify connection properties when you issue the metadata LIBNAME statement; see "LIBNAME Statement for the Metadata Engine " on page 51.

- 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, see the *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= System Option" on page 36, "METAPORT= System Option" on page 37, "METAPROTOCOL= System Option" on page 39, "METAREPOSITORY= System Option" on page 40, "METASERVER= System

and 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 **9999**, repository **myrepos**, metadata server **a123.us.company.com**, and user ID **myuserid**, you would add the following lines to the configuration file:

```
-METAPASS   "sasuser1"
-METAPORT   9999
-METAREPOSITORY "myrepos"
-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="myrepos"
        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 and .

METAPROFILE must be specified at SAS invocation or in a configuration file. It specifies the pathname of an XML document that contains connection profiles. METACONNECT= can be submitted at any time. It specifies one named connection profile in the XML document. 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 the user ID and password of the requesting user.

You can create connection profiles with the Metadata Server Connections dialog box. Open the dialog box by executing the SAS windowing environment command METACON. The dialog box enables you to save (export) one or more connection profiles to a permanent XML document. To learn more about the METACON command, see the online Help in the SAS windowing environment.

The connection profiles are similar to the ones that are used by SAS Management Console. However, SAS Management Console stores its connection profiles in a different way. For more information about connection profiles in SAS Management Console, see the online Help that is available from SAS Management Console.

### Configuration File Example

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

```
-METAPROFILE "!SASROOT\metauser.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 34 and "METAENCRYPTLEVEL System Option" on page 35.

# 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 31.

*Chapter 6*
# System Options for Metadata

## Dictionary

## METAAUTORESOURCES System Option

Identifies the metadata resources that are assigned when SAS starts.

| | |
|---|---|
| **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 invoke SAS. 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.

In SAS 9.2, libraries that had the **Library is pre-assigned** check box selected in their library definitions were pre-assigned using the native library engine defined for that library in metadata, unless the library was already pre-assigned by an AUTOEXEC file.

In SAS 9.3, the SAS Management Console Data Library Manager enables you to specify pre-assignment in the library definition. Specifying the PreAssignmentType property affects METAAUTORESOURCES as follows, unless the library was already pre-assigned by an AUTOEXEC file:

• Libraries marked as pre-assigned by external configuration (for example, the AUTOEXEC file) are ignored by METAAUTORESOURCES.

• Libraries marked as pre-assigned by the native library engine are assigned using the native library engine defined for that library in metadata.

• Libraries marked as pre-assigned by the metadata LIBNAME engine are assigned by the metadata LIBNAME engine (MLE). MLE is a proxy library engine that enforces access controls placed on the library and its tables and columns, as defined in metadata.

Library definitions that were created with the SAS 9.2 Data Library Manager, and that had the **Library is pre-assigned** check box selected, continue to behave as if the native library engine option was specified.

The PreAssignmentType property is not recognized by SAS 9.2 sessions that connect to a SAS 9.3 Metadata Server that was configured with the SAS 9.3 Data Library Manager.

If the SAS Metadata Server is not available, METAAUTORESOURCES is ignored.

METAAUTORESOURCES 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 user connection profiles for connecting to the metadata server.

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

### Syntax

**METACONNECT**="*named-connection*"

### Syntax Description

**"*named-connection*"**
is a named connection that is contained in the metadata user profiles. The maximum length is 256 characters. Quotation marks are required.

### Details

This system option is one of a category of system options that define a connection to the metadata server. Instead of specifying individual connection options for the metadata server, you can use the METACONNECT= and METAPROFILE on page 38 options.

METAPROFILE must be specified at SAS invocation or in a configuration file. It specifies the pathname of an XML document that contains connection profiles. METACONNECT= can be submitted at any time. It specifies one named connection profile in the XML document. 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 the user ID and password of the requesting user.

You can create connection profiles with the Metadata Server Connections dialog box. Open the dialog box by executing the SAS windowing environment command METACON. The dialog box enables you to save (export) one or more connection profiles to a permanent XML document. To learn more about the METACON command, see the online Help in the SAS windowing environment.

The connection profiles are similar to the ones that are used by SAS Management Console. However, SAS Management Console stores its connection profiles in a different way.

*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: Use in a Configuration File

Here is an example from a configuration file, followed by an explanation:

```
-METAPROFILE "C:\userprofile.xml"
-METACONNECT "B"
```

### Explanation

1. The METAPROFILE system option specifies the file **C:\userprofile.xml**. The file contains three named connection profiles: **A**, **B**, and **C**. Each named connection profile contains properties for connecting to the metadata server.

2. The METACONNECT system option specifies the named connection profile **B**.

3. The metadata server connection properties that are specified in the named connection profile **B** are loaded from the metadata user **B** and are used as the properties for connecting to the metadata server.

For information about the metadata server, see *SAS Intelligence Platform: System Administration Guide*.

### See Also

- "Configuration File Example" on page 29

### System Options

- "METAPROFILE System Option" on page 38

## METAENCRYPTALG System Option

Specifies the type of encryption to use when communicating with the 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 SASPROPRIETARY (alias SAS), you
must have a product license for SAS/SECURE software.

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

Specifies the level of encryption when communicating with the 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.

If the METAENCRYPTALG system option specifies an encryption algorithm other than SASPROPRIETARY (alias SAS), you must have a product license for SAS/SECURE software. For more information about encryption levels, 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

•

# METAPASS= System Option

Specifies the password for the metadata server.

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

## 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:** To specify an encoded password, use the PWENCODE procedure to disguise the text string, and specify the encoded password for METAPASS=. The metadata server decodes the encoded password. For more information, see the PWENCODE procedure in the *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

**System Options**

## METAPORT= System Option

Specifies the TCP port for the 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

### System Options

## METAPROFILE System Option

Identifies the XML document that contains user connection profiles for the metadata server.

| | |
|---|---|
| **Valid in:** | configuration file, SAS invocation |
| **Category:** | Communications: Metadata |
| **PROC OPTIONS GROUP=** | META |
| **Default:** | metaprofile.xml in the current working directory, except under z/OS |
| **See:** | METAPROFILE System Option in *SAS Companion for z/OS* |

## Syntax

**METAPROFILE** "*XML-document*"

### Syntax Description

**"*XML–document*"**
   is the pathname of the XML document that contains user connection profiles for connecting to the metadata server. The pathname is the physical location that is recognized by the operating environment. The maximum length is 32,000 characters. Quotation marks are required.

## Details

This system option is one of a category of system options that define a connection to the metadata server. Instead of specifying individual connection options for the metadata server, you can use the METACONNECT= on page 33 and METAPROFILE options.

METAPROFILE must be specified at SAS invocation or in a configuration file. It specifies the pathname of an XML document that contains connection profiles. METACONNECT= can be submitted at any time. It specifies one named connection profile in the XML document. 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 the user ID and password of the requesting user.

You can create connection profiles with the Metadata Server Connections dialog box. Open the dialog box by executing the SAS windowing environment command METACON. The dialog box enables you to save (export) one or more connection profiles to a permanent XML document. To learn more about the METACON command, open the online Help and in the SAS windowing environment.

The connection profiles are similar to the ones that are used by SAS Management Console. However, SAS Management Console stores its connection profiles in a different way.

METAPROFILE behavior is different under z/OS than under other operating environments. See *SAS Companion for z/OS*.

*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.

## Comparisons

Here is a configuration file example that invokes a user connection profile named Mike's profile from the metauser.xml file:

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

## See Also

*   "Configuration File Example" on page 29

### System Options

*   "METACONNECT= System Option" on page 33

## METAPROTOCOL= System Option

Specifies the network protocol for connecting to the 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 29

# METAREPOSITORY= System Option

Specifies the SAS Metadata Repository to use with the 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 36
- "METAPORT= System Option" on page 37
- "METASERVER= System Option" on page 41
- "METAUSER= System Option" on page 43

## METASERVER= System Option

Specifies the host name or address of the 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.

### 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

-

### System Options

-
-
-
-

## 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

When using IWA, a site can assign an SPN that is used by clients such as the object spawner or a batch SAS job to connect to the metadata server. METASPN= is used with METASERVER= and METAPORT= to establish that connection. If you specify METAUSER= and METAPASS=, then the METASPN= value is *not* used. For information about the SPN 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.

## 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

## METAUSER= System Option

Specifies the user ID for connecting to the 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 29

**System Options**

- "METAPASS= System Option" on page 36
- "METAPORT= System Option" on page 37
- "METASERVER= System Option" on page 41

*Part 3*

---

# Metadata LIBNAME Engine

*Chapter 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*
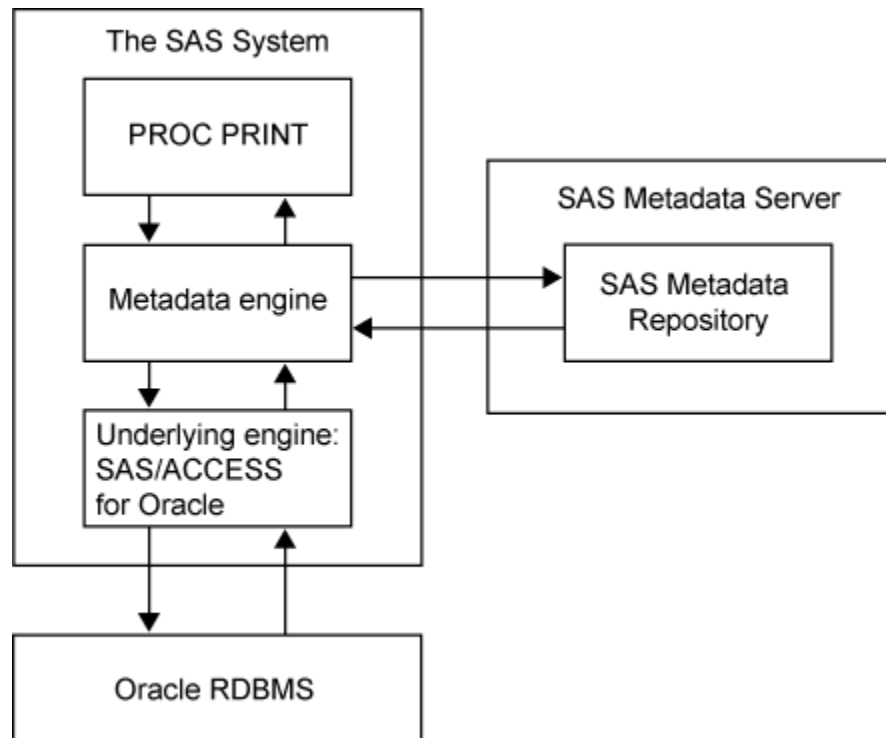


## What Is Supported?

The metadata engine supports the following features:

- Enforces authorizations that are set in the metadata by an administrator.

- 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 53.

## What Is Not Supported

The metadata LIBNAME 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 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, 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 49.

- 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. For more information, see the *SAS Intelligence Platform: Security Administration Guide*.

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

- The administrator can associate authorizations to 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 to 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 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.

# How the Metadata Engine Constructs a LIBNAME Statement

As noted in "Overview of the Metadata LIBNAME Engine" on page 47, 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.

*Chapter 8*
# Reference for the Metadata Engine

## LIBNAME Statement for the Metadata Engine

### Overview: Metadata LIBNAME Statement

To learn how the metadata engine works, see "Introduction to the Metadata LIBNAME Engine" on page 47.

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 53 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*>
    <METAOUT=ALL | DATA | DATAREG | META>

### *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 11.

> 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 "Metadata Object Identifiers and URIs" on page 11. 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 28.

**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 41. 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 36. 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 37. 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 40 . 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 "Metadata Object Identifiers and URIs" on page 11.

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 43 . The maximum length is 256 characters.

**Alias:** ID= or METAUSER= or USERID=

## METAOUT= Argument

**METAOUT=ALL | DATA | DATAREG | META**

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 existing physical tables that are defined in metadata. You cannot create or delete entire physical tables. This is the default behavior.

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

DATA
>    specifies that you can read, create, update, alter, and delete physical tables. The user can access any table, regardless of whether it has been defined in the repository.
>
>    **Restriction:** 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 restricted functionality for tables that are not defined in metadata.

DATAREG
>    specifies that you can read, update, and delete physical tables that are defined in metadata. You can create a table, but you cannot read, update, or delete the new table until it is defined in metadata. This value is like ALL, but it adds the ability to create new tables.
>
>    **Interaction:** The user is restricted to only the tables that have been defined in the repository.

META
>    specifies that you can read physical tables that are defined in metadata. You cannot create, update, or delete physical tables or observations. This value is like ALL, without the ability to create, update, and delete observations.
>
>    **Interaction:** The user is restricted to only the tables that have been defined in the repository.
>
>    **Note:** The METAOUT=META value might not be supported in future releases of the software.

**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:* Although the METAOUT= data set option enables you to specify behavior for individual tables, you can use the METAOUT= argument for the LIBNAME statement to specify behavior for an entire library. However, 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 | META**

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. This is the default behavior.

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

DATA

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

**Restriction:** 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 restricted functionality for tables that are not defined in metadata.

DATAREG

specifies that you can read, update, and delete a physical table that is defined in metadata. You can create a table, but you cannot read, update, or delete the new table until it is defined in metadata. This value is like ALL, but it adds the ability to create new tables.

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

META

specifies that you can read a physical table that is defined in metadata. You cannot create, update, or delete a physical table or observations. This value is like ALL, without the ability to create, update, and delete observations.

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

**Note:** The METAOUT=META value might not be supported in the future releases of the software.

**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.

*Chapter 9*

# Examples for the Metadata Engine

## Example: Submitting the LIBNAME Statement

This example shows two metadata LIBNAME statements. One statement uses defaults, and one statement specifies all of the arguments.

• The following LIBNAME statement uses defaults. The connection information for the SAS Metadata Server is obtained from the metadata system options. Other defaults are obtained from metadata.

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

• This example specifies all of the LIBNAME statement options for the metadata engine to connect to the metadata server.

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

## 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

*Chapter 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 .

## 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 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 9.02.02B0P012308.
NOTE: Server a123.us.company.com SAS Long Version is 99.02.02B0P01232008.
NOTE: Server a123.us.company.com Operating System is XP_PRO.
NOTE: Server a123.us.company.com Operating System Family is WIN.
NOTE: Server a123.us.company.com Operating System Version is Service Pack 2.
NOTE: Server a123.us.company.com Client is janedoe.
NOTE: Server a123.us.company.com Metadata Model is Version 11.02.
NOTE: Server a123.us.company.com is RUNNING on 11Aug08:15:54:15.
```

PROC METADATA can perform a similar check with the following code:

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

In SAS 9.3, you can also use the following 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>12.04</ModelVersion><PlatformVersion>9.3.0.0</PlatformVersion>
<ServerState>ONLINE</ServerState><PauseComment/><ServerLocale>en_US</ServerLocale>
```

The legacy behavior of PROC METADATA is to issue method calls through the SAS Open Metadata Interface DoRequest method. The first PROC METADATA example formats the Status method request for the DoRequest method. The DoRequest method is not available when the SAS Metadata Server is paused. In SAS 9.3, to enable clients to query a paused SAS Metadata Server for status information, the METHOD= argument is implemented to issue Status method requests directly to the server. The second PROC METADATA example shows how METHOD=STATUS is used to get default status information.

PROC METADATA can submit parameters that are not supported by PROC METAOPERATE. Here is an example:

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

The code returns the journal state:

```
<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 180 to determine whether the SAS Metadata Server is paused. SAS provides the "METADATA_VERSION Function" on page 188 to get the model version number.

*Chapter 11*
# METADATA Procedure

## Overview: METADATA Procedure

The METADATA procedure sends an XML string to the SAS Metadata Server. In SAS 9.3, depending on the value in the METHOD= argument, DOREQUEST or STATUS, the IN= argument can contain a SAS Open Metadata Interface method call that is formatted for the IOMI DoRequest method. (This is legacy behavior.) Or, it can contain an XML element that is supported in the IServer Status method.

The IOMI DoRequest method is a messaging interface for SAS Open Metadata Interface methods. It accepts all methods from the IOMI server interface, which consists of methods for reading and writing metadata. It also accepts the IServer Status method, which supports options for monitoring the server and its configuration.

The ability to issue Status requests outside of DoRequest is important because the DoRequest interface is not available when the SAS Metadata Server is paused. METHOD=STATUS enables you to use PROC METADATA to get SAS Metadata Server status information while the SAS Metadata Server is paused.

For more information, see "Concepts: METADATA Procedure" on page 72.

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 63.

# Syntax: METADATA Procedure

| | |
|---|---|
| **Requirement:** | The metadata server must be running. |
| **Tip:** | Be careful when you modify metadata objects, because many objects have dependencies on other objects. 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. For more information, see Chapter 13, "METAOPERATE Procedure," on page 115. |
| **See:** | "Example: Creating a Report with the METADATA Procedure and the XML Engine" on page 13 |

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

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

# PROC METADATA Statement

Sends an XML string to the metadata server

## Syntax

**PROC METADATA** *<server-connection-arguments>*

    <METHOD = DOREQUEST | STATUS>

    IN = "*XML–string*" | *fileref*

    <OUT = *fileref*>

    <HEADER = NONE | SIMPLE | FULL>

    <VERBOSE>;

### *Summary of Optional Arguments*

    HEADER= NONE | SIMPLE | FULL

    METHOD=DOREQUEST | STATUS

    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 "Concepts: METADATA Procedure" on page 72.

- 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 "Concepts: METADATA Procedure" on page 72.

    For information about using filerefs, see "Example 3: Filerefs with the IN= and OUT= Arguments" on page 79 and "Example 4: Fileref to a Temporary File with the IN= Argument" on page 80 .

**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 5: HEADER= Argument" on page 82.

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"?>.

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**

specifies whether PROC METADATA is submitting an IOMI DoRequest method call or an IServer Status method call in the IN= argument. If the METHOD= argument is omitted, the default is DoRequest. For more information, see "Concepts: METADATA Procedure" on page 72.

**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 75. See also "Example 3: Filerefs with the IN= and OUT= Arguments" on page 79.

**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. For more information, see "Example 6: VERBOSE Argument" on page 84.

### Server Connection Arguments

The 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. For more information, see "Connection Options " on page 28.

**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 36. 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 37. 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 39. 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 40. 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 41. 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 43. The maximum length is 256 characters.

**Alias:** ID= or METAUSER= or USERID=

# Concepts: METADATA Procedure

## Introduction to the METHOD= Argument

New in SAS 9.3, the METHOD= argument enables PROC METADATA to submit two types of SAS Open Metadata Interface methods calls to the SAS Metadata Server.

- When METHOD=DOREQUEST (or when the METHOD= argument is omitted from the PROC METADATA request), the procedure issues an IOMI DoRequest method call. It accepts as input another SAS Open Metadata Interface method call that is formatted for the IOMI DoRequest method. The IOMI DoRequest method is a messaging interface for SAS Open Metadata Interface methods. It accepts all methods from the IOMI server interface, which consists of methods for reading and writing metadata. It also accepts the IServer Status method, which supports options for monitoring the server and its configuration. METHOD=DOREQUEST is legacy behavior.

- When METHOD=STATUS, the procedure issues an IServer Status method call. It accepts as input an XML element that is valid in the inMeta parameter of the IServer Status method. The IServer Status method supports XML elements that return information about the SAS Metadata Server, including its current status, the values of server configuration and backup configuration options, and journal statistics. For a comprehensive list of inMeta XML elements, see Status documentation in the *SAS Open Metadata Interface: Reference and Usage*.

Use METHOD=STATUS to get SAS Metadata Server status information if the server is paused. The DoRequest interface is not available when the server is paused.

For examples of how METHOD=STATUS is used, see the following:

- "Comparison of the METADATA Procedure and the METAOPERATE Procedure" on page 63 compares and contrasts PROC METAOPERATE status functionality with the two methods that status information can be requested from PROC METADATA. It also shows how to get default status information.

- "Example 9: Use METHOD=STATUS to Get Backup Information" on page 89.

## Formatting an XML Method Call for DoRequest

The IN= argument of PROC METADATA submits one or more XML-formatted method calls to the 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 in *SAS Open Metadata Interface: Reference and Usage.*

PROC METADATA is among several clients that can submit the DoRequest method to the 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 input metadata objects. 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 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 to 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 in the metadata property string, 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 double quotation marks as follows:

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

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

### 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.

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

This example submits four requests that might be useful when the server is unavailable. For more information about the Status method's XML elements, see the Status method documentation in *SAS Open Metadata Interface: Reference and Usage*.

### 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 13 .

- *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.

## 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 an XML file, see "Example 1: Get Information about Metadata Repositories" on page 75.

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 to 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 13. For more information about the XML engine and XML maps, see the *SAS XML LIBNAME Engine: User's Guide*.

The VERBOSE= argument does not affect the XML output. It causes the input XML to be written to the SAS log. For an example of the information that is written to the log, see "Example 6: VERBOSE Argument" on page 84.

## Examples: METADATA Procedure

## Example 1: Get Information about Metadata Repositories

**Features:**  metadata server connection system options

XML string in IN= argument

default output

fileref in OUT= argument

output displayed in a browser

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

This example issues the IOMI GetRepositories method to list the metadata repositories registered on the SAS Metadata Server. By default, the GetRepositories method gets the values of the Id, Name, Desc, and DefaultNS (namespace) attributes. This example sets the OMI_ALL (1) flag to return the values of the RepositoryType, RepositoryFormat, Access, CurrentAccess, PauseState, and Path attributes.

A request like this is useful for getting repository IDs and for getting a quick idea of repository availability. A repository's persisted access mode is indicated in the Access attribute. Its active state is indicated in the PauseState attribute. If the value in PauseState differs from the value in Access (and PauseState is not blank), then the repository is being affected by a metadata server pause.

The Access attribute returns the following values to indicate a repository's availability:

OMS_ADMIN
indicates the repository is registered for administrative access only.

OMS_FULL
indicates a registered access mode of full access.

OMS_READONLY
indicates a registered access mode of Read-Only access.

OMS_OFFLINE
indicates the repository is intentionally unavailable to all users.

**GetRepositories request with default output.**

```
options metaserver="myserver"
        metaport=8561
        metauser="sasadm@saspw"
        metapass="adminpw";


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

    run;
```

The XML output is written to the SAS log as follows. The output is a continuous, unformatted string. To improve readability, you can assign a fileref, specify the OUT= argument to write the output to an XML file, and open the file in a browser.

```
NOTE: Response XML:

<GetRepositories><Repositories><Repository Id="A0000001.A0000001" Name=
"REPOSMGR" Desc="The Repository Manager" DefaultNS="REPOS" RepositoryType=""
RepositoryFormat="12" Access="OMS_FULL"CurrentAccess="OMS_FULL" PauseState=""
Path="rposmgr"/><Repository Id="A0000001.A5JTOPDN" Name="Foundation" Desc=""
DefaultNS="SAS" RepositoryType="FOUNDATION" RepositoryFormat="12"
Access="OMS_FULL"CurrentAccess="OMS_FULL" PauseState=""
Path="MetadataRepositories/Foundation"/>
<Repository Id="A0000001.A521IKPO" Name="Custom1" Desc="" DefaultNS="SAS"
RepositoryType="CUSTOM" RepositoryFormat="12" Access="OMS_FULL" CurrentAccess=
"OMS_FULL" PauseState="" Path="MetadataRepositories/Custom1"/><Repository
Id="A0000001.A5LSUZGP" Name="Custom2" Desc="" DefaultNS="SAS" RepositoryType=
"CUSTOM" RepositoryFormat="12" Access="OMS_READONLY"CurrentAccess="OMS_READONLY"
PauseState="READONLY" Path="MetadataRepositories/Custom2"/><Repository
Id="A0000001.A5EXCXUI" Name="Custom3" Desc="" DefaultNS="SAS" RepositoryType=
"CUSTOM" RepositoryFormat="12" Access="OMS_ADMIN"CurrentAccess="OMS_ADMIN"
PauseState="ADMIN" Path="MetadataRepositories/Custom3"/></Repositories>
<Flags>1</Flags><Options/></GetRepositories>
```

**GetRepositories request that directs output to an XML file.** For more information about using filerefs, see "Example 3: Filerefs with the IN= and OUT= Arguments" on page 79.

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

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

run;
```

This is what the XML output looks like in a browser:

```
- <GetRepositories>
  - <Repositories>
      <Repository Id="A0000001.A0000001"
        Name="REPOSMGR" Desc="The Repository Manager"
        DefaultNS="REPOS" RepositoryType=""
        RepositoryFormat="12" Access="OMS_FULL"
        CurrentAccess="OMS_FULL" PauseState=""
        Path="rposmgr" />
      <Repository Id="A0000001.A5JTOPDN"
        Name="Foundation" Desc="" DefaultNS="SAS"
        RepositoryType="FOUNDATION"
        RepositoryFormat="12" Access="OMS_FULL"
        CurrentAccess="OMS_FULL" PauseState=""
        Path="MetadataRepositories/Foundation" />
      <Repository Id="A0000001.A521IKPO"
        Name="Custom1" Desc="" DefaultNS="SAS"
        RepositoryType="CUSTOM" RepositoryFormat="12"
        Access="OMS_FULL" CurrentAccess="OMS_FULL"
        PauseState=""
        Path="MetadataRepositories/Custom1" />
      <Repository Id="A0000001.A5LSUZGP"
        Name="Custom2" Desc="" DefaultNS="SAS"
        RepositoryType="CUSTOM" RepositoryFormat="12"
        Access="OMS_READONLY"
        CurrentAccess="OMS_READONLY"
        PauseState="READONLY"
        Path="MetadataRepositories/Custom2" />
      <Repository Id="A0000001.A5EXCXUI"
        Name="Custom3" Desc="" DefaultNS="SAS"
        RepositoryType="CUSTOM" RepositoryFormat="12"
        Access="OMS_ADMIN" CurrentAccess="OMS_ADMIN"
        PauseState="ADMIN"
        Path="MetadataRepositories/Custom3" />
  </Repositories>
  <Flags>1</Flags>
  <Options />
</GetRepositories>
```

## Example 2: Change a Metadata Repository's Availability

**Features:**  XML string in IN= argument

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

A metadata repository's availability, referred to as its *state*, is computed from both the repository's registered Access value and the metadata server's state. To change the repository's persisted state, submit the UpdateMetadata method with PROC METADATA to change the value in the repository's Access attribute.

The example code reregisters the repository with a different Access value, causing the repository's state to be recomputed. To properly execute this code, the repository

manager must be in a Read-Write state. For example, if the metadata server is paused offline, the repository manager will be offline and cannot be updated with this code.

The value of the Access attribute is specified as an integer. The following integers are supported:

0    full access or online

1    Read-Only access

2    administrative access

4    offline or intentionally unavailable to all users

**Change the Access attribute for the repository.** The OMI_TRUSTED_CLIENT flag must be set for the UpdateMetadata method to write to a metadata repository. This example results in full access for the repository that was in administrative mode in the previous example.

```
proc metadata
    in="<UpdateMetadata>
        <Metadata>
         <RepositoryBase Id='A0000001.A5EXCXUI' Access='0'/>
        </Metadata>
        <NS>repos</NS>
        <!-- OMI_TRUSTED_CLIENT flag -->
        <Flags>268435456</Flags>
        </UpdateMetadata>";
    run;
```

## Example 3: Filerefs with the IN= and OUT= Arguments

**Features:**    fileref in IN= argument

fileref in OUT= argument

connection arguments

This example shows how filerefs are used with the IN= and OUT= arguments.

**Create filerefs.** These filerefs specify pathnames to XML files that are stored on a C: drive. If you specify the OUT= argument in the procedure, you must first submit a FILENAME statement, because the OUT= value accepts a fileref only, not a pathname. Record length is specified by the LRECL= argument.

```
filename myinput "c:\myxml\query\weeklyquery.xml" lrecl=256;
filename myoutput "c:\myxml\results\weeklyresults.xml" lrecl=256;
```

**Submit PROC METADATA.** The code specifies metadata server connection arguments, so the defaults are not used. REPOS= is an alias for REPOSITORY=. The procedure submits the contents of weeklyquery.xml (the fileref MYINPUT) to the metadata server, and stores the server's response in weeklyresults.xml (the fileref MYOUTPUT).

```
proc metadata
      server="myserver.us.company.com"
      port=8561
      repos="My Repository"
      userid="testid"
```

```
            password="testpw"
            in=myinput
            out=myoutput;
    run;
```

## Example 4: Fileref to a Temporary File with the IN= Argument

**Features:** fileref in IN= argument

**Other features:** DATA _NULL statement

comparison of DATALINES and PUT statements to submit method request

You might want to test your code without creating a permanent input XML file. You can use a DATA _NULL_ step to create a temporary input XML file.

The input file (whether temporary or permanent) can be created with the DATALINES or PUT statements. Advantages of using the DATALINES statement include:

- DATALINES is easier to type.

- You can enter XML without having to worry about quotation marks and other SAS interpreted strings.

PUT statements are useful when the string is complex or dynamic, with values substituted by DATA step logic.

**Create filerefs.** The input fileref is temporary.

```
filename myinput temp lrecl=256;
filename myoutput "c:\myxml\results\weeklyresults.xml" lrecl=256;
```

**Submit the DATALINES statement to create a temporary file that contains a SAS Open Metadata Interface XML method request.** Later in the example, the temporary input XML file is called by the procedure's IN= argument.

```
data _null_;
   file myinput;
   input;
   put _infile_ '';
   datalines;
<GetMetadataObjects>
   <Reposid>$METAREPOSITORY</Reposid>
   <Type>Column</Type>
   <Objects/>
   <Ns>SAS</Ns>
   <Flags>1</Flags>
   <Options/>
</GetMetadataObjects>
;;
run;
proc metadata
      in=myinput
      out=myoutput;
   run;
```

**Perform the same task with PUT statements.** Alternatively, submit PUT statements to create a temporary file. Each PUT statement is one line in the temporary input XML file.

This code references the same filerefs as the previous code, and produces the same output.

```
data _null_;
   file myinput;
   put "<GetMetadataObjects";
   put "   <Reposid>$METAREPOSITORY</Reposid>";
   put "   <Type>Column</Type>";
   put "   <Objects/>";
   put "   <Ns>SAS</Ns>";
   put "   <Flags>1</Flags>";
   put "   <Options/>";
   put "</GetMetadataObjects>";
run;
proc metadata
      in=myinput
      out=myoutput;
run;
```

**Submit a more complex request in PUT statements.** The GetMetadataObjects request sets the OMI_XMLSELECT (128), OMI_GETMETADATA (256), and OMI_TEMPLATES (4) flags, and specifies an <XMLSELECT search="*string*"/> XML element and templates that request specific Column attributes and associations, and specific attributes of associated objects.

```
data _null_;
  file myinput;
put '<GetMetadataObjects>';
put '<Reposid>$METAREPOSITORY</Reposid>';
put '<Type>Column</Type>';
put '<Objects/>';
put '<NS>SAS</NS>';
put '<Flags>388</Flags>';
put '<Options>';
put '<XMLSelect search="*[Table/PhysicalTable[@Id=''A5BCTYE3.B4000004'']]"/>';
put '<Templates>';
put '  <Column ColumnName="" BeginPosition="" ColumnLength="" ColumnType="" ';
put '     desc="" EndPosition="" IsDiscrete="" IsNullable="" MetadataCreated="" ';
put '     MetadataUpdated="" Name="" SASColumnLength="" SASColumnName="" ';
put '     SASColumnType="" SASExtendedLength="" SASFormat="" SASInformat="" ';
put '     SASPrecision="" SASScale="">';
put '  <AccessControls/>';
put '  <Properties/>';
put '  <PropertySets/>';
put '  </Column>';
put '  <AccessControlEntry Id="" Name=""/> ';
put '  <AccessControlTemplate Id="" Name="" Use=""/> ';
put '  <Property UseValueOnly="" PropertyName="" Delimiter="" DefaultValue=""/>';
put '  <PropertySet PropertySetName="" SetRole=""/> ';
put '</Templates>';
put '</Options>';
put '</GetMetadataObjects>';
run;

proc metadata
  in=myinput
```

```
     out=myoutput;
  run;
```

# Example 5: HEADER= Argument

**Features:** HEADER= argument

This example shows how the HEADER=SIMPLE and HEADER=FULL arguments can be used to specify a header and encoding for the output XML file. For a listing of encoding values that can be used with HEADER=FULL, see the *SAS National Language Support (NLS): Reference Guide*.

**Example of HEADER=SIMPLE** This code inserts the static header <?xml version="1.0" ?> in the output XML file that is identified by the fileref MYOUTPUT. A sample of the content of the output file, opened in a browser, follows.

```
filename myoutput "u:\out1.xml";
proc metadata
     header=simple
     out=myoutput
     in="<GetTypes>
          <Types/>
          <Ns>SAS</Ns>
          <Flags/>
          <Options/>
          </GetTypes>";
run;
```

Here is a display of the top portion of the output file:

```
<?xml version="1.0" ?>
- <GetTypes>
  - <Types>
      <Type Id="AbstractExtension" Desc="Abstract
        extension" HasSubtypes="1" />
      <Type Id="AbstractJob" Desc="Abstract job"
        HasSubtypes="1" />
      <Type Id="AbstractPrompt" Desc="Abstract prompt"
        HasSubtypes="1" />
      <Type Id="AbstractProperty" Desc="Abstract
        property" HasSubtypes="1" />
      <Type Id="AbstractTransformation" Desc="Abstract
        transformation Type" HasSubtypes="1" />
      <Type Id="AccessControl" Desc="Access control"
        HasSubtypes="1" />
      <Type Id="AccessControlEntry" Desc="Access
        control entry" HasSubtypes="0" />
      <Type Id="AccessControlTemplate" Desc="Access
        control template" HasSubtypes="0" />
      <Type Id="Action" Desc="Action" HasSubtypes="0" />
```

**Example of HEADER=FULL** When you specify HEADER=FULL, but do not specify an encoding value in the FILENAME statement, PROC METADATA includes a header with the encoding that is active in your SAS session. A sample of the output follows.

```
filename myoutput "u:\out2.xml";
proc metadata
     header=full
     out=myoutput
     in="<GetTypes>
         <Types/>
         <Ns>SAS</Ns>
         <Flags/>
         <Options/>
         </GetTypes>";
run;
```

Here is a display of the top portion of the output file:

```xml
<?xml version="1.0" encoding="windows-1252" ?>
- <GetTypes>
  - <Types>
    <Type Id="AbstractExtension" Desc="Abstract
      extension" HasSubtypes="1" />
    <Type Id="AbstractJob" Desc="Abstract job"
      HasSubtypes="1" />
    <Type Id="AbstractPrompt" Desc="Abstract prompt"
      HasSubtypes="1" />
    <Type Id="AbstractProperty" Desc="Abstract
      property" HasSubtypes="1" />
    <Type Id="AbstractTransformation" Desc="Abstract
      transformation Type" HasSubtypes="1" />
    <Type Id="AccessControl" Desc="Access control"
      HasSubtypes="1" />
    <Type Id="AccessControlEntry" Desc="Access
      control entry" HasSubtypes="0" />
    <Type Id="AccessControlTemplate" Desc="Access
      control template" HasSubtypes="0" />
    <Type Id="Action" Desc="Action" HasSubtypes="0" />
```

**Example of HEADER=FULL with an Encoding Value** This example creates the output
file with an ASCII encoding. The encoding is specified in the FILENAME statement. An
example of the output follows.

```
filename myoutput "u:\out3.xml" encoding=ascii;
proc metadata
     header=full
     out=myoutput
     in="<GetTypes>
         <Types/>
         <Ns>SAS</Ns>
         <Flags/>
         <Options/>
         </GetTypes>";
run;
```

Here is a display of the top portion of the output file:

```xml
<?xml version="1.0" encoding="us-ascii" ?>
- <GetTypes>
  - <Types>
      <Type Id="AbstractExtension" Desc="Abstract
        extension" HasSubtypes="1" />
      <Type Id="AbstractJob" Desc="Abstract job"
        HasSubtypes="1" />
      <Type Id="AbstractPrompt" Desc="Abstract prompt"
        HasSubtypes="1" />
      <Type Id="AbstractProperty" Desc="Abstract
        property" HasSubtypes="1" />
      <Type Id="AbstractTransformation" Desc="Abstract
        transformation Type" HasSubtypes="1" />
      <Type Id="AccessControl" Desc="Access control"
        HasSubtypes="1" />
      <Type Id="AccessControlEntry" Desc="Access
        control entry" HasSubtypes="0" />
      <Type Id="AccessControlTemplate" Desc="Access
        control template" HasSubtypes="0" />
      <Type Id="Action" Desc="Action" HasSubtypes="0" />
```

## Example 6: VERBOSE Argument

**Features:**     VERBOSE= argument

IN= argument

**Issue PROC METADATA with VERBOSE.** The VERBOSE= argument causes the preprocessed XML from the IN= argument to be written to the SAS log with the title "NOTE: Input XML."

```
proc metadata
  in="<GetMetadataObjects>
        <Reposid>$METAREPOSITORY</Reposid>
        <Type>PhysicalTable</Type>
        <Objects/>
        <Ns>SAS</Ns>
        <Flags>0</Flags>
        <Options/>
        </GetMetadataObjects>"
    verbose;
run;
```

Here is the SAS log. If the OUT= argument had been included in the request, "NOTE: Input XML" would have been the only note in the SAS log.

```
123  proc metadata
124      in="<GetMetadataObjects>
125          <Reposid>$METAREPOSITORY</Reposid>
126          <Type>PhysicalTable</Type>
127          <Objects/>
128          <Ns>SAS</Ns>
129          <Flags/>
130          <Options/>
131          </GetMetadataObjects>"
132      verbose;
133  run;

NOTE: Input XML:

<GetMetadataObjects>   <Reposid>A0000001.A5TJRDIT</Reposid>
<Type>PhysicalTable</Type>   <Objects/>   <Ns>SAS</Ns>      <Flags/>
<Options/>   </GetMetadataObjects>

NOTE: Response XML:

<GetMetadataObjects><Reposid>A0000001.A5TJRDIT</Reposid><Type>PhysicalTable
</Type><Objects><PhysicalTable Id="A5TJRDIT.B2000001" Name="ODSSTYLE"/>
<PhysicalTable Id="A5TJRDIT.B2000003" Name="CENSUS"/><PhysicalTable Id=
"A5TJRDIT.B200000K" Name="EMPLOYEE"/><PhysicalTable Id="A5TJRDIT.B20000SX"
Name="CENSUS"/><PhysicalTable Id="A5TJRDIT.B20000TD" Name="CENSUS"/>
<PhysicalTable Id="A5TJRDIT.B20001L8"Name="CENSUS"/></Objects><Ns>SAS</Ns>
<Flags/><Options/></GetMetadataObjects>
```

## Example 7: Request the Metadata for One Object

**Features:**   IN= argument

**Other features:**   <Flags> element

This code submits a GetMetadata method for a table whose object identifier is A58LN5R2.AR000001. The GetMetadata method retrieves the values of specified properties for a specified metadata object. The requested properties can be specified in the input property string or indicated by GetMetadata flags. GetMetadata supports the following flags in order to get information about an object:

OMI_ALL_SIMPLE (8)
   gets all attributes of the specified object.

OMI_ALL (1)
   gets all of the attributes and direct associations of the specified object.

OMI_FULL_OBJECT (2)
   is new in SAS 9.3. Gets the requested values of the specified object, and the Id=, Name=, and Desc= attributes of all of the secondary objects in the specified object's logical metadata definition. The specified object must be a PrimaryType subtype in the SAS Metadata Model, and it must have a type definition in the type dictionary. For more information about the type dictionary, see "Using Language Elements That Read and Write Metadata" on page 7. OMI_FULL_OBJECT expands both an object's direct and nested associations.

OMI_TEMPLATES (4)

gets the attributes and associations specified in one or more property strings submitted in a <TEMPLATES> XML element in the OPTIONS parameter of the GetMetadata method.

The flags can be set alone or in combination. If you want to return only information about properties that have values stored for them, include the OMI_SUCCINCT (2048) flag. To combine GetMetadata flags, add their numeric values together, and specify the total in the <Flags> parameter. For more information about the flags, see *SAS Open Metadata Interface: Reference and Usage*.

**Request the full metadata definition of a table object.** Metadata objects are identified to the GetMetadata method with their metadata type and Id= values. This example requests information about a PhysicalTable object, and sets the OMI_FULL_OBJECT (2) and OMI_SUCCINCT (2048) flags.

```
filename myoutput "C:\results1.xml" lrecl=256;

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;
```

A sample of the content of the results1.xml file, opened in a browser, follows:

**Output 11.1**    *Contents of the results1.xml File*

```
- <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 8: Request the Metadata for One Type of Object

|  |  |
|---|---|
| **Features:** | IN= argument |
| **Other features:** | <Flags> element |
|  | <XMLSelect> element |

This code submits a GetMetadataObjects method to list all objects of the specified type from the specified repository. The metadata server returns metadata for SAS Information Maps that are registered in the specified repository. A SAS Information Map is represented in a metadata repository with a primary object of the Transformation metadata type. Several other objects in the type dictionary use the Transformation metadata type as their primary metadata type. For example, SAS reports also use the Transformation metadata type. SAS supports two types of information maps: information maps for relational tables and information maps for cubes.

In this request, which specifies a flag value of 128, the GetMetadataObjects OMI_XMLSELECT flag filters the request to retrieve only Transformation objects describing information maps for relational tables. An information map for a relational table has a TypeName= value of "InformationMap.Relational" in its type definition.

The request is submitted from a temporary input file to simplify quoting requirements for the <XMLSELECT search="*string*"/> element. The results are written to an output XML file so that they can be easily viewed in a browser.

For details about information maps, see the *Base SAS Guide to Information Maps*. For information about type definitions and the type dictionary, see "Using Language Elements That Read and Write Metadata" on page 7. For information about PROC METADATA quoting requirements, see "Quotation Requirements" on page 74. This example shows a simple search string. For information about how to build a more complex search string, see *SAS Open Metadata Interface: Reference and Usage*.

**Submit the GetMetadataObjects method.** This request specifies to get information maps for relational tables.

```
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>
 <Flags>2440</Flags>
 <Options>
  <XMLSelect search="*[@PublicType='InformationMap.Relational']"/>
 </Options>
</GetMetadataObjects>
;;
run;
```

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

Here is an example of the output. This request returns one object, shown in the <OBJECTS> XML element. To get the information, such as the responsible parties for the information map and the tables on which the information map is defined, you can set the OMI_GET_METADATA (256), OMI_ALL (8), and OMI_SUCCINCT (2048) flags in the request. To set the flags, add their values to OMI_XMLSELECT (128), and specify the total in the <FLAGS> XML element. The OMI_GET_METADATA flag enables you to set GetMetadata flags in the GetMetadataObjects request. These flags can return a lot of information, so they should be set only when a few objects are expected to be returned by GetMetadataObjects.

The following is the content of the output file, opened in a browser.

**Output 11.2** *Contents of the results2.xml File*

```
- <GetMetadataObjects>
    <Reposid>A0000001.A5TJRDIT</Reposid>
    <Type>Transformation</Type>
  - <Objects>
      <Transformation Id="A5TJRDIT.B100001C" Name="Employee Statistics Sample" />
    </Objects>
    <NS>SAS</NS>
    <Flags>128</Flags>
  - <Options>
      <XMLSelect search="*[@PublicType='InformationMap.Relational']" />
    </Options>
  </GetMetadataObjects>
```

# Example 9: Use METHOD=STATUS to Get Backup Information

**Features:** METHOD= argument

    IN= argument

The SAS 9.3 Metadata Server performs unassisted, scheduled server backups, and supports roll-forward recovery to a specified point in time. For detailed information about this server backup utility and its recovery features, see the *SAS Intelligence Platform: System Administration Guide*.

The recommended interface for managing server backups and performing recoveries is the Server Backup node of SAS Management Console. However, PROC METAOPERATE can be used to configure and execute backups and recoveries, and PROC METADATA can be used to get information about the backup configuration, backup history, and specific backups.

For information about tasks that can be performed with PROC METAOPERATE, see The following examples show how to get information about server backups using METHOD=STATUS.

The SAS 9.3 Metadata Server backup facility uses four system files in the **SASMeta/ MetadataServer** directory to manage backup and recovery processes.

- MetadataServerBackupConfiguration.xml contains the backup configuration and backup schedule.

- MetadataServerBackupHistory.xml contains a history of backup and recovery activity.

- MetadataServerBackupManifest.xml contains a record of the repositories and files copied in a backup.

- MetadataServerRecoveryManifest.xml contains a record of the repositories and files applied in the latest recovery.

These system files should never be opened directly. To read the files with PROC METADATA, submit backup-related XML elements that are supported in the IServer Status method in the IN= argument.

**To get your server's backup configuration and backup schedule, submit the following:**

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

Here is output from the request:

```
NOTE: Response XML:
<MetadataServerBackupConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="MetadataServerBackupConfiguration.xsd">
1 <MetadataServer GUID="5FC4B20A-EEFE-449D-AA16-8D7745B0F0C6"/>
2 <Schedules><Schedule Event="Backup"
Weekday1="0200" Weekday2="0200R" Weekday3="0200" Weekday4="0200" Weekday5="0200"
Weekday6="0200" Weekday7="0200"/></Schedules>
3 <BackupConfiguration BackupLocation="Backups" DaysToRetainBackups="7"
RunScheduledBackups="Y"/></MetadataServerBackupConfiguration>
```

1 The <MetadataServer> XML element contains the unique metadata server identifier in a GUID= attribute.

2 The <Schedules> XML element contains the backup schedule. For a description of the schedule attributes and values, see "<SCHEDULE EVENT="Backup" WEEKDAYn="timeR"/>" on page 123.

3 The <BackupConfiguration> XML element contains values for the BackupLocation=, DaysToRetainBackups=, and RunScheduledBackups= configuration attributes.

**To get the backup schedule for Tuesday only, submit the following:**

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

**To get the backup retention policy only, submit the following:**

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

**To list the backup history:**

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

**To get status information about the last backup from the history:**

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

**To list the backup manifest of the last successful backup, submit the following:**

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

**To list the backup manifest of an earlier backup:** 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;
```

**To list the backup manifest of a backup that is not in the configured backup
location:** 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;
```

**To list the contents of the MetadataServerRecoveryManifest.xml file:** The server
maintains a record of the last recovery only. This manifest is located in the **SASMeta/
MetadataServer** directory.

```
proc metadata
    method=STATUS
    in='<MetadataServerRecoveryManifest/>';
run;
```

**To check the health of the backup scheduler thread:** Possible return values are Alive,
TimeOut, Down, or Unconfigured.

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

*Chapter 12*
# METALIB Procedure

## Overview: METALIB Procedure

The METALIB procedure supports SAS data sets (data files and data views) and DBMS data. The data source is referred to as a table in this documentation and the procedure's output.

When you run PROC METALIB, you specify a SAS library that is already defined in the SAS Metadata Server. You can create a metadata definition for a SAS library using the New Library wizard in the SAS Management Console Data Library Manager or SAS Data Integration Studio.

The METALIB procedure updates the metadata in the metadata server to match the tables in a library. By default, the procedure performs the following tasks:

• creates metadata for any table that does not have metadata

• updates metadata about all of the tables' columns, indexes, and integrity constraints

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

• Select or exclude specific tables from processing.

• Specify where new metadata is stored in SAS folders.

• Perform an impact analysis to see whether any Transformation or Job objects are associated with the tables. (Information maps are modeled with Transformation objects.)

• Limit the update of table definitions that would affect Job or Transformation objects.

• Add a prefix to the name of all new metadata 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.

• Suppress the metadata add action, the metadata update action, or both.

• Delete metadata that has no corresponding data source or is duplicated.

For more information, see "How PROC METALIB Works" on page 103.

# Syntax: METALIB Procedure

**Requirements:** The SAS Metadata Server must be running. The SAS library must already be defined in the metadata by using a product such as SAS Management Console or SAS Data Integration Studio.

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*>);
    <EXCLUDE<=> (*table-specification*<*table-specification-n*>);>
    | <SELECT (*table-specification*<READ=*read-password*>
      <*table-specification-n*<READ=*read-password-n*>>);>
    <FOLDER= "/*pathname*"; > | <FOLDERID= "*identifier.identifier*";>
    <IMPACT_LIMIT = *n*;>
    <NOEXEC;>
    <PREFIX<=><">*text*<">;>
    <REPORT<<=> (*report-arguments*)>;>
    <UPDATE_RULE<=> (<DELETE><NOADD><NODELDUP><NOUPDATE>);>

| Statement | Task | Example |
|---|---|---|
| "PROC METALIB Statement" | Update metadata in the metadata server to match the tables in a library | Ex. 1 |
| "OMR Statement" | Specify the data source and server connection parameters | Ex. 1 |
| "EXCLUDE or SELECT Statement" | Exclude or select a table, or a list of tables, for processing | Ex. 3 |
| "FOLDER= or FOLDERID= Statement" | Specify where new metadata is stored in SAS folders | Ex. 6 |
| "IMPACT_LIMIT Statement" | Specify the maximum number of Job or Transformation objects that can be affected by updates to table definitions | Ex. 4 |
| "NOEXEC Statement" | Suppress the metadata changes from being made | Ex. 4 |
| "PREFIX Statement" | Specify a text string to add to the beginning of all new metadata object names | Ex. 5 |
| "REPORT Statement" | Create a report that summarizes metadata changes | Ex. 2 |
| "UPDATE_RULE Statement" | Override default update behavior | Ex. 2 |

## PROC METALIB Statement

Updates metadata to match the data source.

### Syntax

**PROC METALIB**;

### Details

Supports SAS data sets (data files and data views), DBMS data, and SAS Information Maps in a SAS Metadata Repository.

## OMR Statement

Specifies the data source and 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. This SAS library
>   contains the tables whose metadata is updated. The SASLibrary object can be
>   specified 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 "Metadata Object
>>   Identifiers and URIs" on page 11.
>
>   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 "Metadata Object Identifiers and URIs" on page 11. 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'"**.
>>
>>   **Requirements:**
>>>   You must enclose the LIBURI= value in quotation marks.
>>>
>>>   The URI must resolve to a single metadata object. When using an attribute
>>>   qualifier like @engine='base', if you have more than one Base library
>>>   defined in metadata, PROC METALIB returns **WARNING: Multiple**
>>>   **metadata objects found**.
>
>   **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;**
>
>   **Tip:** In SAS Management Console, avoid the **Pre-assigned Library** template. When
>   you pre-assign a library, choose the resource template that is specific to the type

of data source library that you are creating, and select the **This library is pre-assigned** check box. The **Pre-assigned Library** template is intended for certain system libraries only, and it does not work for other libraries. In addition, for PROC METALIB, you must submit the library pre-assignment in the current SAS session. You can store the LIBNAME statement in an autoexec file, or you can submit the LIBNAME statement in your SAS session before you submit the procedure.

### *Server Connection Arguments*

The server connection arguments establish communication with the 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 28.

**PASSWORD="***password***"**
> is the password for the authorized 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 36. The maximum length is 256 characters.
>
> **Alias:** METAPASS= or PW=

**PORT="***number***"**
> is 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 PORT=, the value of the METAPORT= system option is used. For more information, see "METAPORT= System Option" on page 37. 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 40. 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 "Metadata Object Identifiers and URIs" on page 11.
>
> 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 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 41. The maximum length is 256 characters.
>
> **Alias:** HOST= or IPADDR= or METASERVER=

USER="*authorized-user-ID*"

is an authorized user ID on the 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 43. The maximum length is 256 characters.

**Alias:** ID= or METAUSER= or USERID=

## 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 select or exclude tables, be aware that the tables that you select can affect the associated objects that are updated. For example, both the primary key and foreign key tables must be selected for foreign key metadata to be updated. The primary key and foreign key tables must be in the same library and in the same repository. |

### Syntax

**EXCLUDE**<=>(*table-specification* <*table-specification-n*>)

| **SELECT**<=>(*table-specification* <READ=*read-password*> <*table-specification-n*<READ=*read-password-n*>>);

### *Required Argument*

#### *table-specification*

<">*table-name*<">

is the SAS name of a physical table that is referenced by the SASLibrary object. If metadata already exists for the table, the table name is the value of the SASTableName= attribute of the PhysicalTable object. Do not specify the value of the Name= attribute, which is a user-defined name that can differ from the SAS name.

If any of the table names in the list contain special or mixed-case characters, you must enclose each table name in quotation marks. If any of the table names contain special or mixed-case characters, PROC METALIB converts all unquoted table names to uppercase. In the following example, all of the values must be enclosed in quotation marks, because the fourth value in the statement is mixed case. If the first three values were not enclosed in quotation marks, they would be uppercased as TAB1, TAB2, and TAB3.

```
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 "Metadata Object Identifiers and URIs" on page 11. 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 95.

### 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:**

## 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 is misspelled, PROC METALIB returns an error. The pathname begins with a forward slash, and is given relative to the branch of the folder tree in which the folder resides. Here is an example:

```
folder="/User Folders/MyUserID/My Folder/Test";
```

**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

When you specify FOLDER= or FOLDERID=, you add or update the table definition in the specified SAS folder. 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 definition 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 definitions in the Data Library Manager on the **Plug-ins** tab of SAS Management Console. The multiple table definitions will have the same name, but they will be in different SAS folder locations. Every table definition has a unique metadata identifier.

If you do not specify a folder, and if table definitions exist in more than one folder, PROC METALIB updates the first table definition that is found for each table in the specified library. (If you submit SELECT=, PROC METALIB updates the specified table in the specified library.) If you do not specify a folder, and if a table is new, PROC METALIB adds the new table definition to the SASLibrary object's folder.

## IMPACT_LIMIT Statement

Specifies the maximum number of Job or Transformation objects that can be affected by an update to a table definition.

**See:**   "Example 4: Generating an Impact Analysis" on page 109

### 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 definition. 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 definitions 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 many other objects that could also 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 all new metadata object names.

**See:** "Example 5: Adding a Prefix to New Metadata Names" on page 112

### Syntax

**PREFIX** <=> <">*text*<">;

### *Required Argument*

**<">*text*<">**

is the text string to add. If you do not enclose the text string in quotation marks, the text string is converted to uppercase. If the text string includes special or mixed-case characters, you must enclose the text string in quotation marks. The text string is added to the beginning of the value of the Name= attribute. This modification does not affect PROC METALIB processing because the procedure uses the value of the SASTableName= attribute to compare the metadata to the tables in the data source.

## REPORT Statement

Creates a report that summarizes metadata changes in the Output window.

**Default:** TYPE=SUMMARY report

**See:** "Example 4: Generating an Impact Analysis" on page 109

### Syntax

**REPORT** <<=> (*report-arguments*)>;

### *Optional Arguments*

**TYPE=DETAIL | SUMMARY**

DETAIL

specifies that the report includes all of the information generated by TYPE=SUMMARY, and 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 would be) made to the table that is being processed.

When specified with IMPACT_LIMIT, the following occurs:

- Only tables that have Job or Transformation objects associated with them 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 source (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 with the REPORT Statement" on page 104 .

## UPDATE_RULE Statement

Overrides one or both of the default add and update actions, and specifies the delete actions.

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

## Syntax

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

### *Required Arguments*

**DELETE**

specifies to delete a table definition in the repository if a corresponding table is not found in the data source. If duplicate table definitions exist, the additional table definitions are deleted unless NODELDUP is specified.

**NOADD**

specifies not to add a table definition to the repository for a table that has no metadata.

**NODELDUP**

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

**NOUPDATE**

specifies not to update existing table definitions in the repository to match the corresponding tables in the data source.

# Concepts: METALIB Procedure

### How PROC METALIB Works

The procedure examines the data source (the SAS library) that is referenced by the SASLibrary object. Then, the procedure examines the SAS table names in the data source, and compares them to the values of the SASTableName= attributes in the metadata. For each SAS table name in the data source, the procedure checks the repository association list to see whether a matching table definition exists.

- If a matching table definition does not exist, one is created.

- If a matching table definition exists, it is updated to match the table definition in the data source.

- If duplicate table definitions exist, only the first table definition is updated. The additional table definitions are ignored by default. If you specify UPDATE_RULE=(DELETE), the additional table definitions are deleted. If you specify UPDATE_RULE=(DELETE NODELDUP), the additional table definitions are not deleted.

- If a table definition exists that does not correspond to a table in the data source, it is ignored by default. If you specify UPDATE_RULE=(DELETE), the table definition is deleted.

- If a column name in a table definition matches a column name in the data source, but it is in a different case (for example, lowercase instead of uppercase), then the following change occurs in the table definition. If the data source is a SAS table or view, the column name in metadata is updated to match the case of the column name in the data source. If the data source is a DBMS, the column name in metadata is deleted and added to match the case of the column name in the data source. If the DBMS has column mappings in a SAS Data Integration Studio job, you might have to recreate the column mappings.

- When it creates metadata, PROC METALIB creates column attributes in the metadata to match the data. It does not use the DBSASTYPE data set option settings. You can manually adjust a column's attributes in the metadata, and the metadata LIBNAME engine will use the modified settings. To do this, you must manually adjust a column's attributes in the metadata after each run of the METALIB procedure or you must exclude tables that use the DBSASTYPE data set option.

For more information, see topics about managing table metadata in the *SAS Intelligence Platform: Data Administration Guide*.

### What Metadata Is Updated?

The procedure updates all table definitions that are associated with the specified SASLibrary object. The affected metadata objects include PhysicalTable, Column, Index, PrimaryKey, ForeignKey, UniqueKey, and KeyAssociation, with the following exceptions:

- PROC METALIB does not create UniqueKey objects for data in Oracle.

- PROC METALIB does not create Index, PrimaryKey, and ForeignKey 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.

*Note:* The maximum length for index names that can be registered by PROC METALIB is 256 characters. However, other components of SAS or the DBMS might enforce a shorter length, causing the name to be truncated.

For more information about these metadata objects, see their descriptions in *SAS Metadata Model: Reference*.

# Results: METALIB Procedure with the REPORT Statement

### 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 definitions 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 101.

### *Output Format*

The default report destination in SAS 9.3 is HTML.

### *Details in the Report*

The METALIB procedure updates the attribute values of the table definition and the attribute values of associated objects to match the data in the specified SAS library, and 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 unique key, this value is a two-part identifier in the form *SASTableName.data-source-key-name*.
>
> * For a primary unique key, 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:**   PROC METALIB

OMR statement with server connection arguments

REPORT statement

This example creates metadata that describes the physical tables in a new SAS library in a SAS Metadata Repository. The SAS library must already exist and contain SAS data sets. You must have already created a metadata definition for the SAS library in the SAS Management Console Data Library Manager.

**Specify the data source and server connection properties.** PROC METALIB creates new metadata or updates any existing metadata describing the data sets in the SAS library identified in the OMR statement. A SAS library is identified by referencing its metadata definition. To read the metadata definition, you must establish a connection to the SAS Metadata Server. Here, the optional SERVER, PORT, USER, and PASSWORD arguments are used to establish the server connection. The library definition is identified in the LIBID= argument by its 8-character metadata identifier. The object is assumed to be in the Foundation repository unless the METAREPOSITORY system option specifies a different repository.

```
proc metalib;
     omr (libid="AZ00000A" server="localhost" port="8561"
user="sasadm@saspw" password="adminpw");
```

**Create a report.** The REPORT statement without options specifies to create a default summary report.

```
     report;
   run;
```

This is the report created in the Output window.

**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:**   PROC METALIB

Default connection properties

OMR statement

UPDATE_RULE statement with DELETE argument

REPORT statement with MATCHING argument

This example adds, updates, and deletes existing metadata describing the tables in a SAS library in the SAS Metadata Repository to match the current physical tables in the SAS library. Because this example does not specify connection arguments for the metadata server, the procedure uses the values of the METAPASS, METAPORT, METAREPOSITORY, METASERVER, and METAUSER system options that are active for the SAS session.

**Specify the data source.** Specify a SAS library that is already defined in the metadata. This example specifies the same library that was in the previous example.

```
proc metalib;
   omr (libid="AZ00000A");
```

**Delete obsolete metadata.** This example deletes any table definition that does not correspond to a table in the SAS library. The default actions of add and update are also performed.

```
   update_rule=(delete);
```

**Create a report.** 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 (which would result in adding, updating, or deleting metadata), the REPORT statement returns only summary statistics.

```
   report(matching);
run;
```

This is the report created in the Output window.

**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:**   SELECT statement

This example adds or updates metadata for a specific table.

**Specify a table name.** The SELECT statement identifies a table definition that contains the value MYTABLE in its SASTableName= attribute. Because the UPDATE_RULE statement is omitted, the default is to update or add the specified metadata. Therefore, if a MYTABLE definition does not exist, a new table definition is created.

```
proc metalib;
   omr (liburi="SASLibrary?@name='MyTestLibrary'");
   select (mytable);
   report;
run;
```

**Specify a table ID.** This example uses the SELECT statement, but specifies the table definition's metadata identifier instead of its name. 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 definition. The second part (B00265DX) identifies the table definition in the repository.

```
proc metalib;
   omr (liburi="SASLibrary?@name='MyTestLibrary'");
  select (A7892350.B00265DX);
   report;
run;
```

## Example 4: Generating an Impact Analysis

**Features:**   IMPACT_LIMIT statement
REPORT statement
REPORT(TYPE=DETAIL) statement

To generate an impact analysis, specify IMPACT_LIMIT=0 and REPORT in PROC METALIB. The generated impact analysis shows which tables have associated Job or Transformation objects. Request a detailed report to determine which type of object is associated with each table. To update the tables regardless of impact, specify IMPACT_LIMIT with an integer representing the number of Transformation or Job objects returned.

### IMPACT_LIMIT=0 and REPORT

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.

```
proc metalib;
   omr (libid=AZ000009);
   impact_limit=0;
```

```
        report;
    run;
```

***Output 12.1***   *Default PROC METALIB Report with IMPACT_LIMIT=0*

**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 |

## IMPACT_LIMIT=0 with REPORT (TYPE=DETAIL)

The detailed report identifies the type of object that is associated with each table.

```
proc metalib;
       OMR=(libid=AZ000009);
       impact_limit=0;
       report (type=detail);
    run;
```

***Output 12.2*** *PROC METALIB Report with Report (Type=Detail)*

**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 |

### IMPACT_LIMIT=2 and REPORT

Specifying that two objects can be affected (based on the previous example) allows PROC METALIB to modify the tables.

```
proc metalib;
        OMR=(libid=AZ000009);
        impact_limit=2;
        report;
run;
```

*Output 12.3 PROC METALIB Report with IMPACT_LIMIT=2*

**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

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 has been added to the SAS library. In the data source, the table is named **ABBA**. In the metadata, the table definition is named **December15ABBA**.

**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;
```

**SAS Output**



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

PROC METALIB creates table definitions in the same folder as the specified library
definition unless you specify a different folder with the FOLDER= or FOLDERID=
statement. The default location for a library definition is the **Shared Data** folder. If
you choose to create table definitions in more than one folder, be aware that you must
run PROC METALIB on each folder to update the metadata. PROC METALIB
maintains a separate set of table definitions for each folder.

**If a 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;
```

**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 definitions for the Class table in

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;
```

**If a library named oraclelib is defined in /Shared Data, the following request creates table definitions 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

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:

  - recover memory

  - reload authorization inheritance rules

  - enable or disable Application Response Measurement (ARM) logging

  - specify a new filename for metadata server journaling

- stop or get the status of the metadata server

Beginning in SAS 9.3, PROC METAOPERATE can be used to:

- 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

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 63.

# Syntax: METAOPERATE Procedure

**PROC METAOPERATE** <*server-connection-arguments*>
    ACTION = PAUSE | REFRESH | RESUME | DELETE | EMPTY | UNREGISTER
     | STATUS | STOP
    <NOAUTOPAUSE>
    <OPTIONS = "*XML–string*">
    <OUT = *SAS-data-set*>;

| Statement | Task |
| --- | --- |
| "PROC METAOPERATE Statement" | Perform administrative tasks associated with the metadata server |

# PROC METAOPERATE Statement

Performs administrative tasks associated with the metadata server

## Syntax

**PROC METAOPERATE** <*server-connection-arguments*>
    ACTION = PAUSE | REFRESH | RESUME | DELETE | EMPTY | UNREGISTER
     | STATUS | STOP
    <NOAUTOPAUSE>
    <OPTIONS = "*XML–string*">
    <OUT = *SAS-data-set*>;

### Summary of Optional Arguments

NOAUTOPAUSE
OPTIONS="*XML-string*"
OUT=*SAS-data-set*

### Required Argument

**ACTION=**
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 access privilege 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.

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 privilege 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 6: Submitting ACTION=EMPTY" on page 131.

PAUSE
in SAS 9.3, PAUSE has the following two uses:

- Limits the availability of the metadata server by setting the metadata server's state to ADMIN, OFFLINE, or READONLY. The READONLY state is new in SAS 9.3.

  The PAUSE action affects the metadata server, not an individual repository or the repository manager. For more information, see "How PAUSE, REFRESH, and RESUME Affect Repositories" on page 125.

  Issue ACTION=PAUSE and use the OPTIONS= argument to specify a <SERVER STATE="ADMIN"/> or a <SERVER STATE="READONLY"/> XML string. The OPTIONS= argument, the <SERVER/> element, and the STATE= parameter are optional. The default is to pause the metadata server to an offline state, which also sets the repositories in an offline state.

  The "<PAUSECOMMENT>text</PAUSECOMMENT>" XML element is optional. It enables you to submit free-form text (for example, details about the pause). For more information, see "Example 2: Submitting ACTION=PAUSE with a Pause Comment" on page 130 .

- When used with the "<FORCE/>"XML element in the OPTIONS parameter, the PAUSE action regains control of the metadata server during the recovery process in the event that the recovery process stops responding. The server is paused to an offline state unless you specify a different <SERVER STATE=""/> value.

Here are examples of situations that might require you to pause the metadata server to an ADMIN, OFFLINE, or READONLY state:

- to troubleshoot system errors

- to install or upgrade software

- to temporarily make all repositories available for reading only

- to close repositories and remove the server lock on them to perform a system backup with an external backup tool

- to terminate a hung recovery process, and move the server into an ADMIN state to perform troubleshooting

REFRESH

affects the metadata server differently depending on the XML string that you specify in the OPTIONS= argument. Here are the choices:

- If you specify REFRESH without an XML string, or if you specify the <SERVER/> XML element, the REFRESH action pauses and resumes (in a single step) the metadata server. Do not specify the STATE= parameter in the <SERVER/> XML element. The REFRESH action recovers memory on the metadata server, and reloads authorization inheritance rules. For more information, see "Example 4: Submitting ACTION=REFRESH to Pause and Resume the Metadata Server" on page 130. After the refresh, all repositories return to the same pause state that they were in before the refresh.For more information, see "How PAUSE, REFRESH, and RESUME Affect Repositories" on page 125.

- 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: Submitting ACTION=REFRESH with ARM Logging" on page 130.

- With the "<OMA ALERTEMAILTEST="text"/>" XML element specified, the REFRESH action sends a test alert e-mail message to the address configured in the <OMA ALERTEMAIL="*email-address*"/> option in the omaconfig.xml configuration file.

- 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 XML elements that are new in SAS 9.3, REFRESH enables you to modify the default SAS Metadata Server backup configuration and backup schedule. It executes ad hoc backups, recovers the metadata server from a previous backup, and rebuilds or restarts the backup scheduler thread. For more information, see the following options: "<BACKUPCONFIGURATION attribute(s)/> " on page 121, "<SCHEDULE EVENT="Backup" WEEKDAYn="timeR"/>" on page 123, "<BACKUP attributes/>" on page 120, "<RECOVER BACKUPNAME="name" | BACKUPPATH="pathname" options/>" on page 122, and "<SCHEDULER/>" on page 123. Also see "Using Backup and Recover XML Elements" on page 127 and "Example 8: Submitting ACTION=REFRESH with Backup and Recover Options" on page 132.

RESUME

restores the paused metadata server to the online state. Beginning in SAS 9.2, the RESUME action affects the metadata server, not an individual repository or the

repository manager. For more information, see "How PAUSE, REFRESH, and RESUME Affect Repositories" on page 125 and "Example 5: Submitting ACTION=RESUME" on page 131 .

Any text that was specified in the "<PAUSECOMMENT>text</ PAUSECOMMENT>" XML element during the PAUSE action is cleared.

Beginning in SAS 9.3, RESUME supports a "<FORCE/>" XML element in the "OPTIONS="XML-string"". argument. <FORCE/> regains control of the metadata server during the recovery process in the event that the recovery process stops responding . The server is returned to an online state. The <SERVER/> XML element is required when <FORCE/> is used.

STATUS

returns the metadata server's SAS version or release number, host operating environment, the user ID that started the metadata server, SAS Metadata Model version number, and whether the metadata server is paused or running. For more information, see "Example 1: Submitting ACTION=STATUS" on page 128.

STOP

stops all client activity, and terminates the metadata server. In complex environments, the metadata server shutdown can take a few minutes. Therefore, PROC METAOPERATE might finish executing before the metadata server finishes its shutdown. Metadata in repositories is unavailable until the metadata server is restarted. You cannot restart the metadata server with PROC METAOPERATE.

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 privilege 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.

**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 <BACKUPCONFIGURATION *attribute(s)*/>, <SCHEDULE Event="BACKUP" WEEKDAY*n*="*timevalue*R"/>, <BACKUP *attributes*/>, <RECOVER BACKUPNAME="*name*" | BACKUPPATH="*pathname*" *options*/> 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.

**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 started, ARM parameters return to the values in the configuration file. For more information, see "Example 3: Submitting ACTION=REFRESH with ARM Logging" on page 130 and the *SAS Intelligence Platform: System Administration Guide*, as well as the ARMSUBSYS= and ARMLOC= system options in *SAS Interface to Application Response Measurement (ARM): Reference*. An <ARM/> element can include the following parameters:

ARMSUBSYS="(ARM_NONE | ARM_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 XML Elements" on page 127.

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 67.

REORG="Y | N"
specifies whether repository data sets should be rebuilt to release unused disk space after 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" WEEKDAYn="timeR"/>" on page 123.

<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. The default location is a **Backups** subdirectory of the **SASMeta/ MetadataServer** directory of your SAS 9.3 configuration. 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 will create it for you.

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 ALERTEMAILTEST="*text*"/>
> supported with the REFRESH action, sends a test alert e-mail message to the address configured in the <OMA ALERTEMAIL="*email-address*"/> option in the omaconfig.xml configuration file. The configured recipients can be viewed on the General tab of the active server's Properties window in SAS Management Console. The NOAUTOPAUSE argument is recommended. For more information, see "Example 7: Submitting ACTION=REFRESH with the Alert E-mail Test Option" on page 131.

<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, details about the pause). Quotation marks are optional around the text.

For more information, see "Example 2: Submitting ACTION=PAUSE with a Pause Comment" on page 130. When you submit the RESUME action, the text in <PauseComment> is cleared.

**<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 XML Elements" on page 127. 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", any recent changes to the omaconfig.xml configuration file will be lost, as will any files that were added to the configuration directory after the backup.** The recovery overwrites all files in the **SASMeta/MetadataServer** directory, except the backup history, backup configuration, and manifests, and replaces them with the configuration files in the backup.

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 XML Elements" on page 127 for information to convert server local time values to GMT time.

<SCHEDULE EVENT="Backup" WEEKDAY*n*="*time*<R>"/>
> supported with the REFRESH action, modifies the server backup schedule.

EVENT="Backup"
> specifies the event that will 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.; 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 daily backups at 1 a.m. 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 repository data sets after a backup. The operation is necessary for repository maintenance, but is time-intensive and causes the metadata server to be paused. It should not be performed often. The default backup schedule performs a REORG on Monday (WeekDay2="0100R").

<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 9: Use METHOD=STATUS to Get Backup Information" on page 89. 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, specifies that the action applies to the metadata server. The <SERVER/> XML element has the following uses:

- It is optional for the PAUSE action, and its STATE= parameter is optional. Supported with the PAUSE action, it specifies an access state to apply to the metadata server. With the PAUSE action, if you do not specify the <SERVER STATE="ADMIN | OFFLINE | READONLY"/> XML element, or if you specify <SERVER/> without a STATE= parameter, the default is to pause the metadata server to an offline state, which also sets the repositories to an offline state.

- It is required for the RESUME action without the STATE= parameter when the <FORCE/> option is used.

STATE= has one of the following values:

ADMIN
allows only users with administrative status to read and write metadata on the metadata server.

OFFLINE
disables all Read and Write access to the metadata server.

READONLY
allows Read-Only access for any user.

**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
The server connection arguments establish communication with the 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 28.

**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 36 . 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 37 . 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 39 . 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 40. 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 41. 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 43. The maximum length is 256 characters.

**Alias:** ID= or METAUSER= or USERID=

# Concepts: METAOPERATE Procedure

## *How PROC METAOPERATE Works*

The administrator of the metadata server can perform three types of maintenance with PROC METAOPERATE.

- Control the metadata server by calling methods in the IServer server interface of SAS Open Metadata Architecture. Use PAUSE, REFRESH, RESUME, STATUS, and STOP.

- Control 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.

## *How PAUSE, REFRESH, and RESUME Affect Repositories*

Beginning in SAS 9.2, the PAUSE and RESUME actions affect the metadata server, not an individual repository or the repository manager. The REFRESH action is equivalent to a PAUSE action followed by a RESUME action.

The pause state is a property of each repository. However, a repository's pause state is not set directly. It is computed from both the metadata server state and the repository's registered access mode.

- You can set the metadata server's state with the PAUSE and RESUME actions in PROC METAOPERATE or with SAS Management Console.

- You cannot set a repository's registered access mode with PROC METAOPERATE. To do so, it is recommended that you use SAS Management Console. Or, you can change the access mode by issuing an UpdateMetadata method call with PROC METADATA. You can determine a repository's registered access mode by issuing a GetRepositories method call with PROC METADATA. For more information, see "Example 1: Get Information about Metadata Repositories" on page 75.

- However, notice in the grid below 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, regardless of the repositories' registered access mode. For more information about the tasks that require PAUSE, REFRESH, or RESUME actions, see *SAS Intelligence Platform: System Administration Guide*.

A repository's computed pause state is one of the following:

admin
   allows Read and Write access for users with administrative status only.

admin(readonly)
   allows Read-Only access for users with administrative status only.

offline
   disables all Read and Write access, unloads the repository from memory, and closes the physical files.

online
   allows normal access to the repository.

readonly
   allows Read-Only access for any user.

The following grid shows how a repository's pause state 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 Server State Affects Repository State*

| Registered Access Mode | Online Server State | Admin Server State | Read-Only Server State | Offline Server State |
|---|---|---|---|---|
| online | online | admin | readonly | offline |
| read-only | readonly | admin(readonly) | readonly | offline |
| administration | admin | admin | admin(readonly) | offline |
| offline | offline | offline | offline | offline |

### *Using Backup and Recover XML Elements*

The SAS 9.3 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 9.3 Metadata Server is configured with a default backup configuration and backup schedule by SAS 9.3 configuration processes. The default backup schedule performs daily backups at 1 a.m., and writes them to a `Backups` subdirectory of the `MetadataServer` directory in your SAS configuration directory. Backups are retained for seven days, and run unassisted unless you want to modify the default backup configuration or backup schedule. 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 an offline state. Therefore, it needs to run when server activity is low. The server backup facility supports ad hoc backups and server recovery with an optional roll-forward capability.

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 by using PROC METAOPERATE. You can 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 118 and "<SCHEDULER/>" on page 123.

- an option to interrupt the recovery process in the event that it stops responding. See ACTION="PAUSE" on page 117, ACTION="RESUME" on page 118, and "<FORCE/>" on page 121.

The PROC METAOPERATE <RECOVER/> option supports a ROLL_FORWARD= attribute that enables you to request roll-forward recovery to a specified datetime value. The metadata server log records datetime values in server local time. The ROLL_FORWARD= attribute 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 ISO 8601 format. The ISO 8601 format is a server local datetime value that includes the GMT offset at the end of the string. For example, consider the backup name:

```
2010-09-20T0_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 (0_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 timezone that is greater than GMT has a plus offset (+7_00). Use the GMT offset in your backup names to determine how you need to adjust the input value from the server log.

*T I P*  To avoid having to make the 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 backup-related tasks that can be performed with PROC METAOPERATE:

| 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 | REFRESH | <RECOVER BACKUPNAME="*name*" \| BACKUPPATH="*pathname*" *options*/> |
| 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 information, see "Example 8: Submitting ACTION=REFRESH with Backup and Recover Options" on page 132.

Backups are 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 67.

## Examples: METAOPERATE Procedure

## Example 1: Submitting ACTION=STATUS

**Features:** Connection arguments
ACTION=STATUS argument
OUT= argument

These examples request the status of the metadata server and show the arguments that can connect to the metadata server. They compare the default output to the output that can be obtained with the OUT= argument.

**Specify connection arguments and query the metadata server for its status.** This example specifies all connection arguments for the metadata server and show the output that is written to the SAS log.

```
proc metaoperate
     server="a123.us.company.com"
     port=8561
     userid="myuserid"
     password="mypassword"
     action=status;
run;
```

```
NOTE: Server a123.us.company.com SAS Version is 9.02.02B0P012308.
NOTE: Server a123.us.company.com SAS Long Version is 9.02.02B0P01232008.
NOTE: Server a123.us.company.com Operating System is XP_PRO.
NOTE: Server a123.us.company.com Operating System Family is WIN
NOTE: Server a123.us.company.com Operating System Version is Service Pack 2.
NOTE: Server a123.us.company.com Client is myuserid.
NOTE: Server a123.us.company.com Metadata Model is Version 11.02.
NOTE: Server a123.us.company.com is RUNNING on 11Aug08:15:54:15.
```

**Specify connection arguments, query the metadata server for its status, and specify the OUT= argument.** LOCALHOST specifies the metadata server that is running on the same host as the SAS session. The status output is written to a data set named WORK.STATOUT.

```
proc metaoperate
     server="localhost"
     port=8561
     userid="myuserid"
     password="mypassword"
     action=status
     out=statout;
run;

proc print data=work.statout;
run;
```

**The SAS System**

| Obs | ATTRIBUTE | VALUE |
|---|---|---|
| 1 | Server | a123.us.company.com |
| 2 | Version | 9.03.01B0P121510 |
| 3 | Sysvlong | 9.03.01B0P12152010 |
| 4 | OS | XP_PRO |
| 5 | OS Family | WIN |
| 6 | OS Version | Service Pack 3 |
| 7 | Client | myuserid |
| 8 | Query Time | 19Jan11:11:07:10 |
| 9 | Status | Running |

## Example 2: Submitting ACTION=PAUSE with a Pause Comment

**Features:**  ACTION=PAUSE argument

OPTIONS= argument with <PauseComment>

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

The following example issues a PAUSE action and includes a comment about the pause.

**Specify a comment when you pause the metadata server.** You can use the <PauseComment> text to explain why 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.

```
proc metaoperate
   action=pause
   options="<Server STATE='ADMIN'/>
            <PauseComment>The server will resume at 2:00
a.m.</PauseComment>";
run;
```

## Example 3: Submitting ACTION=REFRESH with 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.

**Enable ARM logging** The ARMLOC= value specifies a pathname for the ARM log file.

```
proc metaoperate
   action=refresh
   options="<ARM ARMSUBSYS=""(ARM_OMA)"" ARMLOC=""logs/armfile.log""/>";
run;
```

## Example 4: Submitting ACTION=REFRESH to Pause and Resume 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.

**Refresh the metadata server.** To recover memory and reload inheritance rules, submit the REFRESH action without options.

```
proc metaoperate
   action=refresh;
run;
```

## Example 5: Submitting ACTION=RESUME

| | |
|---|---|
| **Features:** | ACTION=RESUME argument |
| **Note:** | You must be an administrative user of the metadata server to perform this action. |

**Resume the metadata server.** This example issues a RESUME action to restore the metadata server to the online state. The RESUME action cannot restart a stopped metadata server.

```
proc metaoperate
    action=resume;
run;
```

## Example 6: Submitting ACTION=EMPTY

| | |
|---|---|
| **Features:** | ACTION=EMPTY argument |
| **Note:** | You must be an administrative user of the metadata server to perform this action. |

**Delete metadata records.** This example removes the metadata records from the specified repository, but does not remove the repository's registration from the repository manager. The EMPTY action is useful for clearing a repository that will be repopulated.

```
proc metaoperate
   action=empty
   repository="MyRepos"
   noautopause;
run;
```

## Example 7: Submitting ACTION=REFRESH with the Alert E-mail Test Option

| | |
|---|---|
| **Features:** | ACTION=REFRESH argument |
| | OPTIONS= argument with <OMA ALERTEMAILTEST="*text*"/> |
| **Note:** | You must be an administrative user of the metadata server to perform this action. |

**Test that the metadata server's alert e-mail system is configured correctly.** The configured alert e-mail recipient is displayed on the General tab of the active server's Properties window. If the addressee does not receive an e-mail message with the specified text from the metadata server, there is a problem with the alert e-mail configuration.

```
proc metaoperate
    action=refresh
    options="<OMA ALERTEMAILTEST='Please disregard. This is only a test.'/>"
    noautopause;
run;
```

## Example 8: Submitting ACTION=REFRESH with Backup and Recover Options

**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 this action.

**To execute an ad hoc backup of the SAS Metadata Server:** This example includes the COMMENT= attribute.

```
proc metaoperate
  action=REFRESH
  options=
    "<Backup Comment='Test of backup system.'/>"
    noautopause;
  run;
```

**To modify the backup configuration:**

```
proc metaoperate
  action=REFRESH
  options=
    "<BackupConfiguration DaysToRetainBackups='4'/>"
   noautopause;
  run;
```

**To modify the default backup schedule:**

```
proc metaoperate
  action=REFRESH
  options=
    "<Schedule Event='Backup' WeekDay1='0200' WeekDay2='0200R' WeekDay3='0200'
 WeekDay4='0200' WeekDay5='0200' WeekDay6='0200' WeekDay7='0200'/>"
  noautopause;
  run;
```

**To recover the metadata server:**

```
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;
```

**To terminate a hung recovery process, and then put the metadata server in an ADMIN state:**

```
proc metaoperate
    action=PAUSE
```

```
        options="<Server State='ADMIN'/><Force/>";
run;
```

*Part 5*

# DATA Step Functions

*Chapter 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 .

For help in 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

Be careful when you modify metadata objects, because many objects have dependencies on other objects. A product like SAS Management Console or SAS Data Integration Studio is recommended for the routine maintenance of metadata. Before you modify metadata, run a full backup of repositories. For more information, see the *SAS Intelligence Platform: System Administration Guide*. If you create a new 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*.

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 good programming practice is to define all variables (for example, with a LENGTH or FORMAT statement) in the DATA step before you call any functions.

For performance reasons, metadata objects are cached by URI. To refresh the metadata object with the most recent data from the metadata server, purge the URI with the "METADATA_PURGE Function" on page 181.

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 182 or "METADATA_GETNOBJ Function" on page 170 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 144 and "Examples: DATA Step Functions for Metadata Security Administration" on page 191.

- 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*.

*Chapter 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_DELASSN Function" (p. 161) | Deletes all objects that make up the specified association |
| "METADATA_DELOBJ" (p. 163) | Deletes the first object that matches the specified URI |
| "METADATA_GETATTR Function" (p. 164) | Returns the value of the specified attribute for specified object |
| "METADATA_GETNASL Function" (p. 165) | Returns the *n*th association of the specified object |

| Name | Description |
| --- | --- |
| "METADATA_GETNASN Function" (p. 167) | Returns the *n*th associated object of the specified association |
| "METADATA_GETNATR Function" (p. 168) | Returns the *n*th attribute on the object specified by the URI |
| "METADATA_GETNOBJ Function" (p. 170) | Returns the *n*th object that matches the specified URI |
| "METADATA_GETNPRP Function" (p. 171) | Returns the *n*th property of the specified object |
| "METADATA_GETNTYP Function" (p. 173) | Returns the *n*th object type on the metadata server |
| "METADATA_GETPROP Function" (p. 174) | Returns the specified property of the specified object |
| "METADATA_NEWOBJ Function" (p. 175) | Creates a new metadata object |
| "METADATA_PATHOBJ Function" (p. 178) | Returns the Id and Type attributes of the specified folder object |
| "METADATA_PAUSED Function" (p. 180) | Determines whether the metadata server is paused |
| "METADATA_PURGE Function" (p. 181) | Purges the specified URI |
| "METADATA_RESOLVE Function" (p. 182) | Resolves a URI into an object on the metadata server |
| "METADATA_SETASSN Function" (p. 183) | Modifies an association list for an object |
| "METADATA_SETATTR Function" (p. 186) | Sets the specified attribute for the specified object |
| "METADATA_SETPROP Function" (p. 187) | Sets the specified property for the specified object |
| "METADATA_VERSION Function" (p. 188) | 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 "Metadata Object Identifiers and URIs" on page 11. 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 19 and "Example: Creating a Report with the METADATA Procedure and the XML Engine" on page 13.

# 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 27.

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 170.

METADATA_GETATTR
> Returns the value of the specified attribute for the specified object. For more information, see "METADATA_GETATTR Function" on page 164.

METADATA_GETNASN
> Returns the nth associated object of the specified association. For more information, see "METADATA_GETNASN Function" on page 167.

METADATA_RESOLVE
> Resolves a URI into an object on the metadata server. For more information, see "METADATA_RESOLVE Function" on page 182.

## 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 9.3 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:

*Display 15.1   PROC PRINT of metadata_libraries Data Set*



## 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:

*Display 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:

*Display 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:

**Display 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 in (for example, sasadm@saspw). This value is known only to the metadata server. External accounts require domain-qualified user IDs to log in. 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:

*Display 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** |

*Chapter 16*
# DATA Step Functions for Reading and Writing Metadata

**Dictionary** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **161**

# Dictionary

## 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

```
options metaserver="a123.us.company.com"
       metaport=8561
       metauser="myid"
       metapass="mypassword"
       metarepository="myrepos";

data _null_;
    length uri $256
          curi $256
          curi1 $256
          curi2 $256;

    rc=0;

    /* Create a PhysicalTable object. */

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

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

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

    put rc=;
    put curi=;

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

    put rc=;
```

```
        put curi1=;

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

        put rc=;
        put curi2=;

        /* 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_DELOBJ

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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

data _null_;

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

run;
```

## See Also

### Functions

## 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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

data _null_;

    length name $200
           desc $200
           modified $100;

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

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


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

run;
```

## See Also

### Functions

-
-

## 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 138 .

*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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

    do while(rc>0);

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

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

## See Also

**Functions**

- "METADATA_GETNASN Function" on page 167
- "METADATA_SETASSN Function" on page 183

## 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 138 .

*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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

    do while(rc>0);

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

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

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

# METADATA_GETNATR Function

Returns the *n*th attribute 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 138.

*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.

## Examples

### *Example 1: Using an Object URI*

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

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

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

        if (rc>0) then put n= 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_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
  .

***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.

**-3**
  No objects match the specified URI.

**-4**
  *n* is out of range.

### Examples

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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
```

```
      metarepository="myrepos";

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

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

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

run;
```

### *Example 2: Looping Through Each Repository on a 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" on page 163
- "METADATA_NEWOBJ Function" on page 175

## 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 138.

*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.

## Example

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

    do while(rc>0);

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

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

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

### See Also

#### Functions

## 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 138.

***type* (out)**
    returns the metadata type in the specified row.

### *Return Values*

***n***
    The number of objects that match the URI.

**-1**
    Unable to connect to the metadata server.

**-4**
    *n* is out of range.

### Example

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

    do while(rc>0);

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

```
            n=n+1;
        end;
    run;
```

## METADATA_GETPROP Function

Returns the 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.

### Example

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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;
```

## 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.

### 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

```
options metaserver="myserver"
        metaport=8561
        metauser="sasadm@saspw"
        metapass="adminpw"
        metarepository="Foundation";


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_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 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 the *SAS Metadata Model: Reference*. 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.

## Examples

### Example 1: Specifying the TypeName Value in Path

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";
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*

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";
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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

## METADATA_PURGE Function

Purges the specified URI.

### Syntax

rc = **METADATA_PURGE**(*<uri>* );

### *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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

    do while(rc>0);
```

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

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

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

                        rc=metadata_getnasl("omsobj:Machine?@Name='bluedog'",
                                            n,
                                            association);
                    put 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.

## Examples

### *Example 1: Using an Object URI*

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

### *Example 2: Using a Repository URI*

```
options metaserver="a123.us.company.com"
    metaport=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=;
        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-n>*);

### *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*).

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.

* 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 with a 0–to-many or 1–to-many cardinality. Many associations between the specified metadata types is supported.

For more information about associations and cardinality, see *SAS Open Metadata Interface: Reference and Usage*.

***auri–1<,...auri-n>* (in)**
specifies a list of the URIs of the associated objects; see "Array Parameters" on page 138 .

### *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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

    /* Create a TextStore object. */

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

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

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

run;
```

## See Also

### Functions

- "METADATA_DELASSN Function" on page 161
- "METADATA_GETNASN Function" on page 167

## 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

```
options metaserver="a123.us.company.com"
        metaport=8561
        metauser="myid"
        metapass="mypassword"
        metarepository="myrepos";

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

### See Also

#### Functions

- "METADATA_GETATTR Function" on page 164
- "METADATA_GETNATR Function" on page 168

## 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.

### Example

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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_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

```
options metaserver="a123.us.company.com"
    metaport=8561
    metauser="myid"
    metapass="mypassword"
    metarepository="myrepos";

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

# 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 202<br><br>"METASEC_ENDTRAN Function" on page 204 | "Example: Begin and End Transaction Context" (p. 192) |
| Access control definition | "METASEC_APPLYACT Function" on page 201<br><br>"METASEC_GETNACT Function" on page 206<br><br>"METASEC_GETNAUTH Function" on page 210<br><br>"METASEC_GETNID Function" on page 213<br><br>"METASEC_REMACT Function" on page 216<br><br>"METASEC_SETAUTH Function" on page 219 | "Example: Working with ACTs" (p. 193) |
| ACT manipulation | "METASEC_DELACT Function" on page 203<br><br>"METASEC_GETACTA Function" on page 205<br><br>"METASEC_GETNACTA Function" on page 208<br><br>"METASEC_NEWACT Function" on page 215<br><br>"METASEC_SETACTA Function" on page 217 | "Example: Working with ACTs" (p. 193) |

# 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 138.

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 12.

- 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;
```

# 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 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 for a usage example.

## See Also

### Functions

-

## 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 193 for a usage example.

## See Also

### Functions

- "METASEC_GETACTA Function" on page 205
- "METASEC_GETNACT Function" on page 206
- "METASEC_NEWACT Function" on page 215
- "METASEC_SETACTA Function" on page 217

## 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(). A value is required. The function will return an error if you do not specify a value.

   _SECAD_COMMIT_TC      Commit transaction.

   _SECAD_DISCARD_TC     Do not commit transaction.

   For more information, see "Using the %MDSECCON() Macro" on page 191.

### *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 192 for a usage example.

## See Also

### Functions

- "METASEC_BEGTRAN Function" on page 202

## 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 193 for a usage example.

## See Also

### Functions

- "METASEC_DELACT Function" on page 203
- "METASEC_GETNACTA Function" on page 208
- "METASEC_NEWACT Function" on page 215
- "METASEC_SETACTA Function" on page 217

## 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 138.

*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 | Return all ACTs in all SAS Metadata Repositories that are not of type PROJECT. |

    For more information, see "Using the %MDSECCON() Macro" on page 191.

## *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 193 for a usage example.

## See Also

### Functions

- "METASEC_APPLYACT Function" on page 201
- "METASEC_GETNAUTH Function" on page 210
- "METASEC_GETNID Function" on page 213
- "METASEC_REMACT Function" on page 216
- "METASEC_SETAUTH Function" on page 219

## 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 138.

***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 193 for a usage example.

## See Also

### Functions

- "METASEC_DELACT Function" on page 203
- "METASEC_GETACTA Function" on page 205
- "METASEC_NEWACT Function" on page 215
- "METASEC_SETACTA Function" on page 217

## 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 138.

*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 to 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 212.

*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 211.

*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 212. 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():

| | |
|---|---|
| _SECAD_ACT_CONTENTS | Return the authorizations that define the contents of an ACT when the *tc* or *uri* argument references an ACT. |
| _SECAD_DO_NOT_RETURN_PERMCOND | Do not return any available values for the *cond* argument. |

For more information, see "Using the %MDSECCON() Macro" on page 191.

*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

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. To simplify usage, you can use macro variables from %MDSECCON() instead of the integer values. Here are the authorizations, macro variables, and descriptions. For more information, see "Using the %MDSECCON() Macro" on page 191. For suggested usage, see "Example: Working with ACTs" on page 193.

*Table 18.3   Explicit, ACT, and Indirect Authorizations and Masks*

| Authorization Type | Macro Variable | Description |
|---|---|---|
| Explicit deny | _SEC_PERM_EXPD | Explicit deny that originates from the authorization that is directly associated to the object |

| Authorization Type | Macro Variable | Description |
| --- | --- | --- |
| Explicit grant | _SEC_PERM_EXPG | Explicit grant that originates from the authorization that is directly associated to the object |
| Explicit mask | _SEC_PERM_EXPM | Mask to extract explicit value that originates from the authorization that is directly associated to the object |
| ACT deny | _SEC_PERM_ACTD | Deny that originates from an ACT other than the default ACT |
| ACT grant | _SEC_PERM_ACTG | Grant that originates from an ACT other than the default ACT |
| ACT mask | _SEC_PERM_ACTM | Mask to extract indirect value that originates from an ACT other than the default ACT |
| Indirect deny | _SEC_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 | _SEC_PERM_NDRG | Indirect grant that originates from an IdentityGroup membership, via inheritance, or from the default ACT; an indirect value is always returned |
| Indirect mask | _SEC_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. |

### *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 will 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

### Functions

## 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 138.

*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():

| | |
|---|---|
| _SECAD_ACT_CONTENTS | 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        Returns **roles** as the type for
                               IdentityGroups that are acting as roles.

For more information, see "Using the %MDSECCON() Macro" on page 191.

### *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 193 for a usage example.

## See Also

### Functions

- "METASEC_APPLYACT Function" on page 201
- "METASEC_GETNACT Function" on page 206
- "METASEC_GETNAUTH Function" on page 210

# 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 215 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 215 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 18.4* *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 193 for a usage example.

## See Also

### Functions

- "METASEC_DELACT Function" on page 203
- "METASEC_GETACTA Function" on page 205
- "METASEC_GETNACTA Function" on page 208
- "METASEC_SETACTA Function" on page 217

## 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 193 for a usage example.

## See Also

### Functions

- "METASEC_APPLYACT Function" on page 201
- "METASEC_GETNACT Function" on page 206
- "METASEC_GETNAUTH Function" on page 210
- "METASEC_GETNID Function" on page 213
- "METASEC_SETAUTH Function" on page 219

# 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 18.5 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 |

| Attribute | Attribute Value | Maximum Length of Attribute Value | Notes |
|-----------|-----------------|-----------------------------------|-------|
| `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 193 for a usage example.

## See Also

### Functions

- "METASEC_DELACT Function" on page 203
- "METASEC_GETACTA Function" on page 205
- "METASEC_GETNACTA Function" on page 208
- "METASEC_NEWACT Function" on page 215

# 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():

_SECAD_ACT_CONTENTS    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 191.

## *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 193 for a usage example.

## See Also

**Functions**

- "METASEC_APPLYACT Function" on page 201
- "METASEC_GETNACT Function" on page 206
- "METASEC_GETNAUTH Function" on page 210
- "METASEC_GETNID Function" on page 213
- "METASEC_REMACT Function" on page 216

# Glossary

**access control template**
a reusable named authorization pattern that you can apply to multiple resources. An access control template consists of a list of users and groups and indicates, for each user or group, whether permissions are granted or denied. Short form: ACT.

**ACT**
See access control template.

**Application Response Measurement**
the name of an application programming interface that was developed by an industry partnership and which is used to monitor the availability and performance of software applications. ARM monitors the application tasks that are important to a particular business. Short form: ARM.

**ARM**
See Application Response Measurement.

**authentication**
See client authentication.

**authorization**
the process of determining which users have which permissions for which resources. The outcome of the authorization process is an authorization decision that either permits or denies a specific action on a specific resource, based on the requesting user's identity and group memberships.

**batch mode**
a noninteractive method of running SAS programs by which a file (containing SAS statements along with any necessary operating system commands) is submitted to the batch queue of the operating environment for execution.

**client authentication**
the process of verifying the identity of a person or process for security purposes.

**column**
a vertical component of a table. Each column has a unique name, contains data of a specific type, and has particular attributes. A column is analogous to a variable in SAS terminology.

**data set**
See SAS data set.

**data view**

See SAS data view.

**database management system**

a software application that enables you to create and manipulate data that is stored in the form of databases. Short form: DBMS.

**DBMS**

See database management system.

**encryption**

the act or process of converting data to a form that is unintelligible except to the intended recipients.

**engine**

a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

**job**

a collection of SAS tasks that can create output.

**library reference**

See libref.

**libref**

a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

**localhost**

the keyword that is used to specify the machine on which a program is executing. If a client specifies localhost as the server address, the client connects to a server that runs on the same machine.

**metadata**

descriptive data about data that is stored and managed in a database, in order to facilitate access to captured and archived data for further use.

**metadata LIBNAME engine**

the SAS engine that processes and augments data that is identified by metadata. The metadata engine retrieves information about a target SAS library from metadata objects in a specified metadata repository.

**metadata object**

a set of attributes that describe a table, a server, a user, or another resource on a network. The specific attributes that a metadata object includes vary depending on which metadata model is being used.

**metadata repository**

a collection of related metadata objects, such as the metadata for a set of tables and columns that are maintained by an application. A SAS Metadata Repository is an example.

**metadata server**

a server that stores information about servers, users, and stored processes and that provides this information to one or more client applications.

**observation**
> a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains either one data value or a missing-value indicator for each variable.

**SAS data file**
> a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data.

**SAS data set**
> a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

**SAS data set option**
> an option that appears in parentheses after a SAS data set name. Data set options specify actions that apply only to the processing of that SAS data set.

**SAS data view**
> a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. Short form: data view.

**SAS library**
> one or more files that are defined, recognized, and accessible by SAS and that are referenced and stored as a unit. Each file is a member of the library.

**SAS Management Console**
> a Java application that provides a single user interface for performing SAS administrative tasks.

**SAS Metadata Model**
> a collection of metadata types that are used for saving information about application elements.

**SAS Metadata Server**
> a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories.

**SAS Open Metadata Architecture**
> a general-purpose metadata management facility that provides metadata services to SAS applications. The SAS Open Metadata Architecture enables applications to exchange metadata, which makes it easier for these applications to work together.

**SAS table**
> another term for SAS data set.

**SAS variable**

a column in a SAS data set or in a SAS data view. The data values for each variable describe a single characteristic for all observations (rows).

**SAS/ACCESS software**

a group of software interfaces, each of which makes data from a particular external database management system (DBMS) directly available to SAS, as well as making SAS data directly available to the DBMS.

**statement option**

a word that you specify in a particular SAS statement and which affects only the processing that that statement performs.

**transformation**

in data integration, an operation that extracts data, transforms data, or loads data into data stores.

**variable**

See SAS variable.

# Index