



THE
POWER
TO KNOW.

SAS[®] 9.2

Language Interfaces to Metadata



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS® 9.2 Language Interfaces to Metadata*. Cary, NC: SAS Institute Inc.

SAS® 9.2 Language Interfaces to Metadata

Copyright © 2009 by SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59047-781-6

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

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

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

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

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

1st electronic book, February 2009

2nd electronic book, May 2010

1st printing, March 2009

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

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

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

Contents

<i>What's New</i>	vii
Overview	vii
System Options	vii
Metadata LIBNAME Engine	vii
Procedures	viii
DATA Step Functions	viii
Documentation Enhancements	ix

PART 1 Introduction 1

Chapter 1 △ What Are the Metadata Language Elements? 3

Overview of Metadata Language Elements	3
When to Use Metadata Language Elements	3
Accessibility Features of SAS Language Interfaces to Metadata	4

Chapter 2 △ Metadata Object Identifiers and URIs 5

What Is a Metadata Identifier?	5
Obtaining Metadata Names and Identifiers	5
What Is a URI?	6

Chapter 3 △ Examples: Using Metadata Language Elements 7

Overview of the Examples	7
Example: Pausing the Server for an Administration Task	7
Example: Creating a PropertySet Object for Use with LIBOPTSET=	8
Example: Creating a Report with the METADATA Procedure and the XML Engine	9
Example: Creating a Report with the DATA Step	14

PART 2 System Options 21

Chapter 4 △ Introduction to System Options for Metadata 23

Overview of System Options for Metadata	23
Connection Options	24
Encryption Options	25
Resource Option	26

Chapter 5 △ System Options for Metadata 27

PART 3 Metadata LIBNAME Engine 43

Chapter 6 △ Introduction to the Metadata LIBNAME Engine 45

Overview of the Metadata LIBNAME Engine	45
What Is Supported?	46

Advantages of Using the Metadata Engine	46
The Metadata Engine and Authorization	47
How the Metadata Engine Constructs a LIBNAME Statement	47
How the Metadata Engine Constructs Options	47

Chapter 7 △ Reference for the Metadata Engine 51

LIBNAME Statement for the Metadata Engine	51
SAS Data Set Options for the Metadata Engine	55

Chapter 8 △ Reference to Metadata Objects for the Metadata Engine 59

Overview of Metadata Requirements	59
Diagrams of the SAS Metadata Model	59
Metadata Objects, Listed by Language Element	63
Metadata Objects, Listed by Type	66

Chapter 9 △ Examples for the Metadata Engine 75

Example: Submitting the LIBNAME Statement	75
Example: Before and After the Metadata Engine	75

PART 4 Procedures 79

Chapter 10 △ Introduction to Procedures for Metadata 81

Overview of Procedures for Metadata	81
Comparison of the METADATA Procedure and the METAOPERATE Procedure	81

Chapter 11 △ METADATA Procedure 83

Overview: METADATA Procedure	83
Syntax: METADATA Procedure	84
Concepts: METADATA Procedure	87
Results: METADATA Procedure	89
Examples: METADATA Procedure	90

Chapter 12 △ METALIB Procedure 99

Overview: METALIB Procedure	99
Syntax: METALIB Procedure	100
Concepts: METALIB Procedure	108
Results: METALIB Procedure with the REPORT Statement	109
Examples: METALIB Procedure	111

Chapter 13 △ METAOPERATE Procedure 117

Overview: METAOPERATE Procedure	117
Syntax: METAOPERATE Procedure	117
Concepts: METAOPERATE Procedure	123
Examples: METAOPERATE Procedure	124

PART 5 DATA Step Functions 129

Chapter 14 △ Introduction to DATA Step Functions for Metadata 131

Overview of DATA Step Functions for Metadata 131

Best Practices 131

Array Parameters 132

Chapter 15 △ DATA Step Functions for Reading and Writing Metadata 133

Introduction to DATA Step Functions for Reading and Writing Metadata 135

METADATA_DELASSN Function 137

METADATA_DELOBJ Function 139

METADATA_GETATTR Function 140

METADATA_GETNASL Function 142

METADATA_GETNASN Function 143

METADATA_GETNATR Function 145

METADATA_GETNOBJ Function 147

METADATA_GETNPRP Function 149

METADATA_GETNTYP Function 151

METADATA_GETPROP Function 152

METADATA_NEWOBJ Function 153

METADATA_PATHOBJ Function 155

METADATA_PAUSED Function 157

METADATA_PURGE Function 158

METADATA_RESOLVE Function 160

METADATA_SETASSN Function 162

METADATA_SETATTR Function 165

METADATA_SETPROP Function 166

METADATA_VERSION Function 167

Chapter 16 △ DATA Step Functions for Metadata Security Administration 169

Introduction to DATA Step Functions for Metadata Security Administration 171

METASEC_APPLYACT Function 173

METASEC_BEGTRAN Function 174

METASEC_DELACT Function 176

METASEC_ENDTRAN Function 177

METASEC_GETACTA Function 178

METASEC_GETNACT Function 180

METASEC_GETNACTA Function 182

METASEC_GETNAUTH Function 183

METASEC_GETNID Function 187

METASEC_NEWACT Function 189

METASEC_REMACT Function 191

METASEC_SETACTA Function 192

METASEC_SETAUTH Function 194

Examples: DATA Step Functions for Metadata Security Administration 196

Appendix 1 △ Recommended Reading 207

Recommended Reading 207

Glossary 209

Index 213

What's New

Overview

Changes and enhancements in the SAS language interfaces to metadata include the following:

- a new system option METASPN= and defaults for METAREPOSITORY=, METAENCRYPTALG, and METAENCRYPTLEVEL system options
- read-only access to metadata with the metadata LIBNAME engine, and enhancements for data processing
- several new arguments in the METALIB procedure
- changes to what the METAOPERATE procedure can control, including the SAS Metadata Server's pause state and the metadata server journal file
- new arguments and values for some DATA step functions
- a new set of DATA step functions for security administration and reporting
- new syntax in the metadata LIBNAME engine, PROC METALIB, and the new METADATA_PATHOBJ function for the pathname where metadata is stored in SAS folders
- documentation enhancements

System Options

- The system option METASPN= is new. This option specifies the service principal name (SPN) for the metadata server. The SPN is a feature of Integrated Windows authentication (IWA).
- The default for the METAREPOSITORY= system option is Foundation.
- The default for the METAENCRYPTALG is SASPROPRIETARY.
- The default for the METAENCRYPTLEVEL system option is CREDENTIALS.

Metadata LIBNAME Engine

The metadata LIBNAME engine has the following changes and enhancements:

- Metadata is read-only. If you want to update metadata, use PROC METALIB.
- The argument METAOUT=DATAREG is new. This argument specifies that you can read, update, and delete only the tables and columns that are defined in the metadata. The other values of METAOUT= have been changed to reflect the fact that metadata is read-only.
- The LIBRARY= value can specify a pathname where metadata is stored in SAS folders.
- The metadata LIBNAME engine supports SAS views.
- SAS file passwords can be passed to the underlying engine.

Procedures

The METALIB procedure has the following changes and enhancements:

- In the third maintenance release for SAS 9.2, column names in metadata are updated to match the case of the column names in the data source.
- The LIBRARY=, FOLDER=, or FOLDERID= value can specify a pathname where metadata is stored in SAS folders. In the third maintenance release for SAS 9.2, a table can be defined in more than one folder.
- The IMPACT_LIMIT statement limits the number of Job or Transformation objects that can be changed.
- The PREFIX= statement adds a text string to the beginning of all new metadata object names.
- Arguments in the REPORT statement determine the level of detail in the output report.
- The READ= argument in the SELECT statement enables PROC METALIB to read password-protected data sets.
- PROC METALIB updates WorkTable objects.

The METAOPERATE procedure has the following changes and enhancements:

- The PAUSE and RESUME actions affect the metadata server, not an individual SAS Metadata Repository or the repository manager. The pause state is still 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.
- The PURGE action is no longer supported.
- XML strings in the OPTIONS statement can specify a pathname for the metadata server journal file and provide a comment to users about the metadata server's pause state.

The METADATA procedure has the following changes and enhancements:

- The input and output XML strings might differ from previous releases. This change is a result of enhancements to the SAS 9.2 Metadata Model, with new metadata types, modifications to existing metadata types, and modifications to the object hierarchy. See *SAS Metadata Model: Reference*.

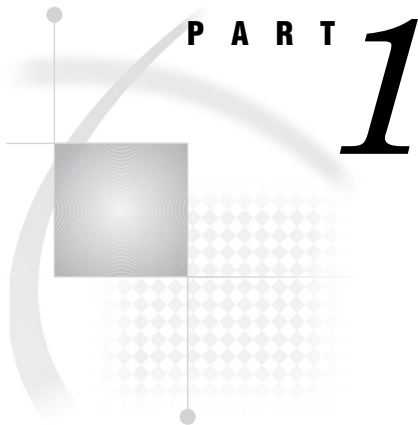
DATA Step Functions

- An argument in the DATA step functions METADATA_GETPROP and METADATA_SETPROP provides the Uniform Resource Identifier (URI) of the Property object.

- The METADATA_SETASSN function can perform replace, modify, and merge.
- The METADATA_PATHOBJ function is new. This function returns the attributes of an object that you specify by its pathname in SAS folders.
- A new set of DATA step functions can define and query the authorization settings for the metadata server. You can use macros with the functions to create reports.
- The DATA step functions' input and output parameters might differ from previous releases. This change is a result of enhancements to the SAS 9.2 Metadata Model, with new metadata types, modifications to existing metadata types, and modifications to the object hierarchy. See *SAS Metadata Model: Reference*.

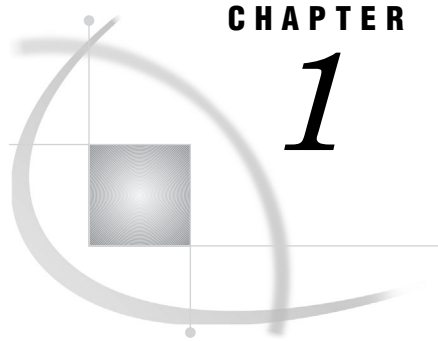
Documentation Enhancements

This book is new. *SAS Language Interfaces to Metadata* is a reference to the language elements that you can submit in a SAS session (in batch or from the SAS windowing environment) to use, query, and maintain the metadata server. Most of these language elements were previously documented in several other books. This book brings the documentation together, adds new language elements and new features, and expands the conceptual information.



Introduction

<i>Chapter 1</i>	What Are the Metadata Language Elements?	3
<i>Chapter 2</i>	Metadata Object Identifiers and URIs	5
<i>Chapter 3</i>	Examples: Using Metadata Language Elements	7



CHAPTER

1

What Are the Metadata Language Elements?

<i>Overview of Metadata Language Elements</i>	3
<i>When to Use Metadata Language Elements</i>	3
<i>Accessibility Features of SAS Language Interfaces to Metadata</i>	4

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 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.

This book is a reference to the metadata language elements. For information about metadata administration tasks, see the administration books in “Recommended Reading” on page 207.

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 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 maintenance task. Here is an overview of the language elements:

System options

Use the system options to set defaults for metadata. They are organized into three groups: connection to the metadata server, 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

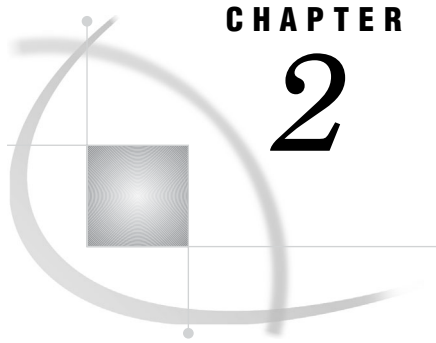
You can use the procedures to perform many common maintenance tasks on metadata and the metadata server.

DATA step functions

The DATA step functions cover some of the same maintenance tasks as the procedures and, in many cases, do more than the procedures. The DATA step functions execute within a DATA step, so you can use the output from one function as the input to another function.

Accessibility Features of SAS Language Interfaces to Metadata

This product has not been tested for compliance with U.S. Section 508 standards. If you have specific questions about the accessibility of SAS products, send them to accessibility@sas.com or call SAS Technical Support.



CHAPTER

2

Metadata Object Identifiers and URIs

<i>What Is a Metadata Identifier?</i>	5
<i>Obtaining Metadata Names and Identifiers</i>	5
<i>What Is a URI?</i>	6

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 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 9.

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.

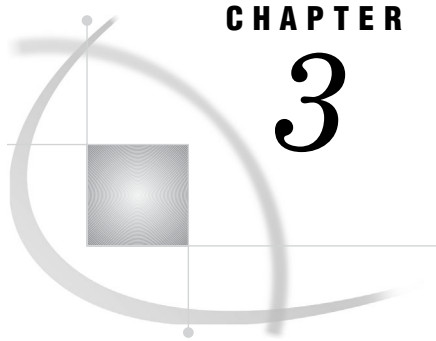
type/ID

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 **SASLibrary?@engine='base'**. 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

3

Examples: Using Metadata Language Elements

Overview of the Examples 7

Example: Pausing the Server for an Administration Task 7

Example: Creating a PropertySet Object for Use with LIBOPTSET= 8

Example: Creating a Report with the METADATA Procedure and the XML Engine 9

Example: Creating a Report with the DATA Step 14

Overview of the Examples

These examples show a few typical maintenance tasks that can be performed with metadata language elements. For information about metadata administration tasks, see the administration books in “Recommended Reading” on page 207.

Example: Pausing the Server for an Administration Task

The following example is copied from *SAS Intelligence Platform: System Administration Guide*. See that book for information about system administration.

This example uses the COMPARE procedure to compare the MDASSOC and RPOSCTRL data sets in the current and backup repository manager directories. PROC METAOPERATE pauses the metadata server temporarily while the PROC COMPARE statements complete.

```
options
metaserver="localhost"
metaport=8561
metauser="userID"
metapass="encoded-password"
;
/* Pause the metadata server */
proc metaoperate
    action=pause;
run;

/* Assign libraries to the physical locations in your operating environment */
libname origrpos "C:\SAS\BIservr\Levl\SASMeta\MetadataServer\rposmgr";
libname backup "C:\SAS\BIservr\Levl\SASMeta\MetadataServer\SASBackup\REPOSMGR";

proc compare base=origrpos.mdassoc compare=backup.mdassoc; run;
proc compare base=origrpos.rposctrl compare=backup.rposctrl; run;
```

```

/* Resume the metadata server */
proc metaoperate
    action=resume;
run;

```

Example: Creating a PropertySet Object for Use with LIBOPTSET=

The metadata LIBNAME statement supports the LIBOPTSET= argument, which specifies a PropertySet object. The PropertySet object is associated to Property objects that store information. This information is used to construct a LIBNAME statement for the underlying engine.

In this release of SAS, no products enable you to directly create a PropertySet object. Instead, you can use code like this example to programmatically create the object.

The example uses the DATA step with PUT statements to create a temporary XML file. Then the METADATA procedure submits the XML file to the metadata server in order to add the metadata objects.

The PropertySet object has an OwningObject of SASLibrary, and the Property objects need an OwningType and PropertyType. This example defines a PropertyType with Name="String" and SQLType="1".

```

options metaserver="my-metadata-server"
    metaport=8561
    metauser="my-id"
    metapass="my-pw"
    metarepository="Foundation";

/*provide the name of the metadata library with which to associate this set of options*/
%let LibraryName=Oracle Library;

/*provide a name for the PropertySet object*/
%let PropSetName=SetOraBuffOptions;

/*create temporary files for building and receiving XML for PROC METADATA*/
filename inxml temp lrecl=32767;
filename outxml temp lrecl=32767;

/*generate the temporary IN= XML file for PROC METADATA*/
data _null_;
    length uri $256 LibId $17;

    /*retrieve the uri for the requested library*/
    rc=metadata_getnobj("omsobj:SASLibrary?@Name=''||"&LibraryName"||"', 1, uri);

    /*verify that the library exists in the metadata, if so retrieve the metadata Id*/
    if rc <= 0 then put 'Library not found in metadata';
    else rc=metadata_getattr(uri,'Id',LibId);

    /*build the XML for PROC METADATA to add the new PropertySet to the metadata*/
    file inxml;
    put '<AddMetadata>';
    put ' <Metadata>';
    /*the <PropertySet> tag creates the object with the name from %LET above*/

```

```

put ' <PropertySet Name=""&PropSetName""';
put '   Desc="Set ReadBuff=50 and InsertBuff=100">';
      /*the PropertySet will be owned by the library specified in the %LET above*/
put '   <OwningObject>';
put '     <SASLibrary ObjRef="'LibId'" />';
put '   </OwningObject>';
      /*each object is defined as a <Property>*/
put '   <SetProperties>';
put '     <Property Name="Block insert buffer size" DefaultValue="50" Delimiter="=" ' ;
put '       PropertyName="INSERTBUFF" UseValueOnly="0">';
put '     </Property>';
put '     <Property Name="Block read buffer size" DefaultValue="100" Delimiter="=" ' ;
put '       PropertyName="READBUFF" UseValueOnly="0">';
put '     </Property>';
put '     <Property Name="Use extended memory" DefaultValue="MEMLIB" UseValueOnly="1">';
put '     </Property>';
put '   </SetProperties>';
put ' </PropertySet>';
put ' </Metadata>';
put ' <Reposid>$METAREPOSITORY</Reposid>';
put ' <NS>SAS</NS>';
put ' <Flags>268435456</Flags>';
put ' <Options/>';
put ' </AddMetadata>';
run;

/*pass the XML to PROC METADATA to create the new PropertySet in the metadata*/
proc metadata in=inxml out=outxml header=full verbose;
run;

```

After an administrator submits the previous code to set the LIBNAME options, a user can submit the following code to assign the library:

```
libname mylib meta library="Oracle Library" liboptset="SetOraBuffOptions";
```

The options INSERTBUFF=50, READBUFF=100, and MEMLIB are applied to the library. For more information, see “How the Metadata Engine Constructs Options” on page 47.

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 ODS 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 Chapter 11, "METADATA Procedure," on page 83. 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 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>
  <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 file 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\';

ods html file="tables.html" path=reports;
proc print data=mybase.final;
run;

ods html close;

```

Here is the HTML report:

Table Metadata

Obs	SASTableName	Columns	Column_IDs	Libref	Engine
1	EXAMS	Subject	A58LN5R2.AS000001	myfiles	BASE
2	EXAMS	Test1	A58LN5R2.AS000002	myfiles	BASE
3	EXAMS	Test2	A58LN5R2.AS000003	myfiles	BASE
4	EXAMS	Test4	A58LN5R2.AS000004	myfiles	BASE
5	REPS	name	A58LN5R2.AS0000RT	myfiles	BASE
6	REPS	div	A58LN5R2.AS0000RU	myfiles	BASE
7	SALES	Q1	A58LN5R2.AS0001JL	myfiles	BASE
8	SALES	Q2	A58LN5R2.AS0001JM	myfiles	BASE
9	SALES	Q3	A58LN5R2.AS0001JN	myfiles	BASE
10	SALES	Q4	A58LN5R2.AS0001JO	myfiles	BASE
11	SALES	cost	A58LN5R2.AS0001JP	myfiles	BASE
12	SALES	net	A58LN5R2.AS0001JQ	myfiles	BASE
13	EDUC	div	A58LN5R2.AS0001JR	myfiles	BASE
14	EDUC	rank	A58LN5R2.AS0001JS	myfiles	BASE

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=myserver,
                      metaport=8561,
                      usr=sasadm@saspw,
                      pw=Password1,
                      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;

%if %eval(&metaver>0) %then
  %do; /* connected to metadata server */

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

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

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

```

```

if numcon>0 then
do; /* server with connections */
  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;

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;

%end; /* connected to metadata server */

%else
  %do; /* could not connect to metadata server */
    %put ERROR: could not connect to &metaserver &metaport. ;
    %put ERROR: check connection details, userid and password.;
  %end; /* could not connect to metadata server */

%mend;

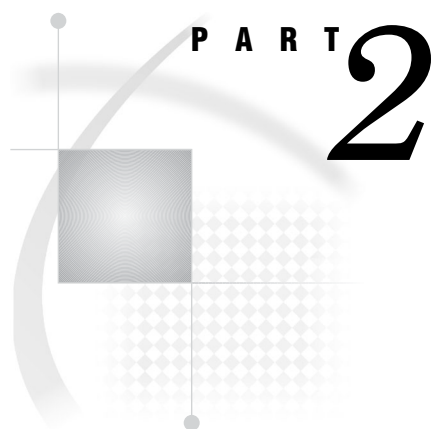
%server_report (metaserver=myserver,
               metaport=8561,
               usr=sasadm@saspw,
               pw=Password1,
               includeopt=N,
               htmlloc=c:\reports\myservers.htm
               );

```

Here is the HTML report:

Server Report for Metadata Server myserver:8561, 20MAY2008

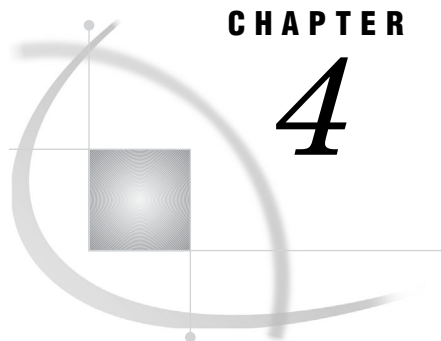
Server Name	Vendor	Product	Version	Host Name	Port	Connection Name	App Protocol	Com Protocol	Authentication Domain
Object Spawner - myserver	SAS Institute	SAS Workspace Spawner	9.2	myserver	8501	A5BH2FQZPortBank_0	PortBank	TCP	none
					8511	A5BH2FQZPortBank_1	PortBank	TCP	none
					8521	A5BH2FQZPortBank_2	PortBank	TCP	none
					8581	Connection: Spawner	Operator	TCPIP	none
SASApp - Workspace Server	SAS Institute	SAS Workspace	9.2	myserver	8591	Connection: Workspace	Bridge	TCP	none
Table Server - Data Service	SAS Institute	SASTS	9.2	myserver	2171	Connection: Logical Table Server	Bridge	TCP	none



System Options

Chapter 4. **Introduction to System Options for Metadata** 23

Chapter 5. **System Options for Metadata** 27



CHAPTER

4

Introduction to System Options for Metadata

<i>Overview of System Options for Metadata</i>	23
<i>Connection Options</i>	24
<i>Introduction to Connection Options</i>	24
<i>Specifying Connection Properties Directly</i>	24
<i>Example: Configuration File</i>	24
<i>Example: OPTIONS Statement</i>	25
<i>Specifying a Stored Connection Profile</i>	25
<i>Configuration File Example</i>	25
<i>Encryption Options</i>	25
<i>Resource Option</i>	26

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.

Category	System Options
Connection	METACONNECT=
	METAPASS=
	METAPORT=
	METAPROFILE
	METAPROTOCOL=
	METAREPOSITORY=
	METASERVER=
	METASPN=
Encryption	METAUSER=
	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: Dictionary*. 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 METAPASS=, METAPORT=, METAPROTOCOL=, METAREPOSITORY=, METASERVER=, METASPN=, and METAUSER= system options. See “Specifying Connection Properties Directly” on page 24.
- Specify a stored metadata server connection profile with the METACONNECT= and METAPROFILE options. See “Specifying a Stored Connection Profile” on page 25.
- You can specify connection properties when you issue a metadata procedure. See “Overview of Procedures for Metadata” on page 81.
- You can 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

The METAPASS=, METAPORT=, METAPROTOCOL=, METAREPOSITORY=, METASERVER=, METASPN=, and METAUSER= options each specify a connection property. Typically these values are set in a configuration file.

Example: Configuration File

To set the default metadata server to use the password **sasuser1**, port **9999**, protocol **bridge**, 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
-METAPROTOCOL BRIDGE
-METAREPOSITORY "myrepos"
-METASERVER "a123.us.company.com"
-METAUSER "myuserid"

```

Example: OPTIONS Statement

The following OPTIONS statement has the same effect as the configuration file example:

```

options metapass="sasuser1"
        metaport=8561
        metaprotocol=bridge
        metarepository="myrepos"
        metaserver="a123.us.company.com"
        metauser="myuserid";

```

Specifying a Stored Connection Profile

Instead of specifying individual connection options for the metadata server, you can use the METACONNECT= 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, 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

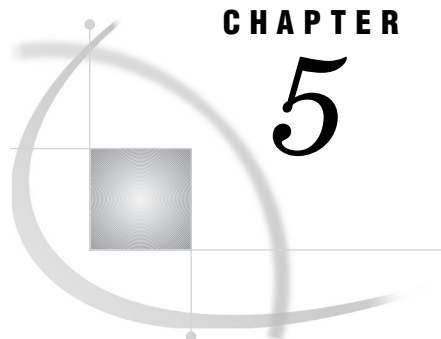
The METAENCRYPTALG and METAENCRYPTLEVEL options are used to encrypt communication with the metadata server. You do not have to license SAS/SECURE software if you specify the SAS proprietary algorithm. For more information, see

“METAENCRYPTALG System Option” on page 30 and “METAENCRYPTLEVEL System Option” on page 32.

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 27.



CHAPTER

5

System Options for Metadata

<i>METAAUTORESOURCES</i> System Option	27
<i>METACONNECT=</i> System Option	29
<i>METAENCRYPTALG</i> System Option	30
<i>METAENCRYPTLEVEL</i> System Option	32
<i>METAPASS=</i> System Option	33
<i>METAPORT=</i> System Option	34
<i>METAPROFILE</i> System Option	35
<i>METAPROTOCOL=</i> System Option	37
<i>METAREPOSITORY=</i> System Option	38
<i>METASERVER=</i> System Option	39
<i>METASPN=</i> System Option	40
<i>METAUSER=</i> System Option	41

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'"
```

Note: 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. Δ

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.

If the metadata server is not available, this option is ignored. If libraries are assigned in an autoexec file, those assignments take precedence over assignment with METAAUTORESOURCES.

See Also

For information about pre-assigning SAS libraries, see the *SAS Intelligence Platform: Data Administration Guide*.

See other system options:

- “METACONNECT= System Option” on page 29
- “METAENCRYPTALG System Option” on page 30
- “METAENCRYPTLEVEL System Option” on page 32
- “METAPASS= System Option” on page 33
- “METAPORT= System Option” on page 34
- “METAPROFILE System Option” on page 35
- “METAPROTOCOL= System Option” on page 37
- “METAREPOSITORY= System Option” on page 38
- “METASERVER= System Option” on page 39
- “METASPN= System Option” on page 40
- “METAUSER= System Option” on page 41

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.

Note: 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. △

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 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.

Example

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

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

- 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.

See Also

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

See the “Configuration File Example” on page 25.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METAENCRYPTALG System Option” on page 30

“METAENCRYPTLEVEL System Option” on page 32

“METAPASS= System Option” on page 33

“METAPORT= System Option” on page 34

“METAPROFILE System Option” on page 35

“METAPROTOCOL= System Option” on page 37

“METAREPOSITORY= System Option” on page 38

“METASERVER= System Option” on page 39

“METASPN= System Option” on page 40

“METAUSER= System Option” on page 41

METAENCRYPTALG System Option

Specifies the type of encryption to use when communicating with the metadata server.

Valid in: configuration file, SAS invocation

Alias: METAENCRYPTALGORITHM

Category: Communications: Metadata

PROC OPTIONS GROUP= META

Default: SASPROPRIETARY

Syntax

METAENCRYPTALG NONE | RC2 | RC4 | DES | TRIPLEDES |
SASPROPRIETARY | SAS

Syntax Description

NONE

specifies that no encryption is used.

RC2

specifies to use the RC2 encryption algorithm that was developed by RSA Security. The RC2 algorithm uses a block cipher to encrypt 64-bit blocks. The RC2 key size is variable and can range from 8 to 256 bits. A single message can be expanded to 8 bytes. RC2 encryption is an alternative to Data Encryption Standard (DES) encryption.

RC4

specifies to use the RC4 encryption algorithm that was developed by RSA Security. The RC4 algorithm uses a stream cipher to encrypt 1 byte at a time. The RC4 key size is variable and can range from 8 to 2,048 bits.

DES

specifies to use the Data Encryption Standard (DES) encryption algorithm that was developed by IBM. The DES algorithm uses a block cipher to encrypt 64-bit blocks. DES uses a 56-bit key.

TRIPLEDES

specifies to use the TRIPLEDES encryption algorithm, which processes the DES encryption algorithm sequentially three times on the data. TRIPLEDES uses a different 56-bit key for each iteration of the encryption. A single message can be expanded to 8 bytes.

SASPROPRIETARY

specifies to use basic encryption services in all operating environments. The SASPROPRIETARY algorithm uses a 32-bit key and can expand a single message by one-third. No additional product license is required. This is the default.

Alias: SAS

Note: 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. △

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, see the *SAS Intelligence Platform: Security Administration Guide*.

See Also

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

For information about security, see the *SAS Intelligence Platform: Security Administration Guide*.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTLEVEL System Option” on page 32

“METAPASS= System Option” on page 33
 “METAPORT= System Option” on page 34
 “METAPROFILE System Option” on page 35
 “METAPROTOCOL= System Option” on page 37
 “METAREPOSITORY= System Option” on page 38
 “METASERVER= System Option” on page 39
 “METASPN= System Option” on page 40
 “METAUSER= System Option” on page 41

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.

Note: 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. △

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, see the *SAS Intelligence Platform: Security Administration Guide*.

See Also

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

For information about security, see the *SAS Intelligence Platform: Security Administration Guide*.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTALG System Option” on page 30

“METAPASS= System Option” on page 33

“METAPORT= System Option” on page 34

“METAPROFILE System Option” on page 35

“METAPROTOCOL= System Option” on page 37

“METAREPOSITORY= System Option” on page 38

“METASERVER= System Option” on page 39

“METASPN= System Option” on page 40

“METAUSER= System Option” on page 41

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.

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*. Δ

Note: 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. Δ

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*.

See Also

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

See “Example: Configuration File” on page 24.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTALG System Option” on page 30

“METAENCRYPTLEVEL System Option” on page 32

“METAPORT= System Option” on page 34

“METAPROFILE System Option” on page 35

“METAPROTOCOL= System Option” on page 37

“METAREPOSITORY= System Option” on page 38

“METASERVER= System Option” on page 39

“METASPN= System Option” on page 40

“METAUSER= System Option” on page 41

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

Default: 8561

Range: 1–65535

Syntax

METAPORT=*number*

Syntax Description

number

is the TCP port that the metadata server is listening to for connections. An example is **metaport=5282**.

Note: 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. △

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*.

See Also

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

See “Example: Configuration File” on page 24.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTALG System Option” on page 30

“METAENCRYPTLEVEL System Option” on page 32

“METAPASS= System Option” on page 33

“METAPROFILE System Option” on page 35

“METAPROTOCOL= System Option” on page 37

“METAREPOSITORY= System Option” on page 38

“METASERVER= System Option” on page 39

“METASPN= System Option” on page 40

“METAUSER= System Option” on page 41

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.

Note: 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. △

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 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*.

Example

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

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

See "Configuration File Example" on page 25.

See other system options:

 "METAAUTORESOURCES System Option" on page 27

“METACONNECT= System Option” on page 29
 “METAENCRYPTALG System Option” on page 30
 “METAENCRYPTLEVEL System Option” on page 32
 “METAPASS= System Option” on page 33
 “METAPORT= System Option” on page 34
 “METAPROTOCOL= System Option” on page 37
 “METAREPOSITORY= System Option” on page 38
 “METASERVER= System Option” on page 39
 “METASPN= System Option” on page 40
 “METAUSER= System Option” on page 41

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. This is the default. In this release, it is the only supported value.

Note: 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. 

Details

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

See Also

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

See “Example: Configuration File” on page 24.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTALG System Option” on page 30

“METAENCRYPTLEVEL System Option” on page 32

“METAPASS= System Option” on page 33

“METAPORT= System Option” on page 34

“METAPROFILE System Option” on page 35

“METAREPOSITORY= System Option” on page 38

“METASERVER= System Option” on page 39

“METASPN= System Option” on page 40

“METAUSER= System Option” on page 41

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.

Note: 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. Δ

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.

See Also

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

See other system options:

- “METAAUTORESOURCES System Option” on page 27
- “METACONNECT= System Option” on page 29
- “METAENCRYPTALG System Option” on page 30
- “METAENCRYPTLEVEL System Option” on page 32
- “METAPASS= System Option” on page 33
- “METAPORT= System Option” on page 34
- “METAPROFILE System Option” on page 35
- “METAPROTOCOL= System Option” on page 37
- “METASERVER= System Option” on page 39
- “METASPN= System Option” on page 40
- “METAUSER= System Option” on page 41

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.

Note: 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. 

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*.

See Also

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

See “Example: Configuration File” on page 24.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTALG System Option” on page 30

“METAENCRYPTLEVEL System Option” on page 32

“METAPASS= System Option” on page 33

“METAPORT= System Option” on page 34

“METAPROFILE System Option” on page 35

“METAPROTOCOL= System Option” on page 37

“METAREPOSITORY= System Option” on page 38

“METASPN= System Option” on page 40

“METAUSER= System Option” on page 41

METASPN= System Option

Specifies the service principal name (SPN) for the 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:port*

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.

Note: 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. Δ

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*.

Example

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

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

See Also

For information about the SPN and IWA, see the *SAS Intelligence Platform: Security Administration Guide*.

For information about the SAS Metadata Server, see the *SAS Intelligence Platform: System Administration Guide*.

See other system options:

- “METAAUTORESOURCES System Option” on page 27
- “METACONNECT= System Option” on page 29
- “METAENCRYPTALG System Option” on page 30
- “METAENCRYPTLEVEL System Option” on page 32
- “METAPASS= System Option” on page 33
- “METAPORT= System Option” on page 34
- “METAPROFILE System Option” on page 35
- “METAPROTOCOL= System Option” on page 37
- “METAREPOSITORY= System Option” on page 38
- “METASERVER= System Option” on page 39
- “METAUSER= System Option” on page 41

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=*ID*

Syntax Description

ID

is the user ID for connecting to the metadata server. The maximum length is 256 characters.

Note: 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. △

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*.

See Also

For information about the SAS Metadata Server, see *SAS Intelligence Platform: System Administration Guide*.

See “Example: Configuration File” on page 24.

See other system options:

“METAAUTORESOURCES System Option” on page 27

“METACONNECT= System Option” on page 29

“METAENCRYPTALG System Option” on page 30

“METAENCRYPTLEVEL System Option” on page 32

“METAPASS= System Option” on page 33

“METAPORT= System Option” on page 34

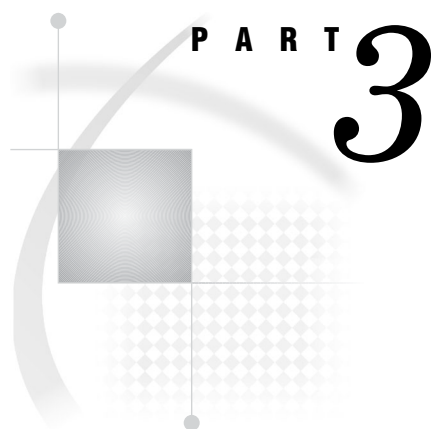
“METAPROFILE System Option” on page 35

“METAPROTOCOL= System Option” on page 37

“METAREPOSITORY= System Option” on page 38

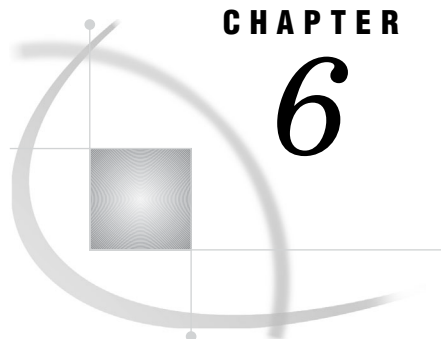
“METASERVER= System Option” on page 39

“METASPN= System Option” on page 40



Metadata LIBNAME Engine

<i>Chapter 6</i>	Introduction to the Metadata LIBNAME Engine	45
<i>Chapter 7</i>	Reference for the Metadata Engine	51
<i>Chapter 8</i>	Reference to Metadata Objects for the Metadata Engine	59
<i>Chapter 9</i>	Examples for the Metadata Engine	75



CHAPTER

6

Introduction to the Metadata LIBNAME Engine

<i>Overview of the Metadata LIBNAME Engine</i>	45
<i>What Is Supported?</i>	46
<i>Advantages of Using the Metadata Engine</i>	46
<i>The Metadata Engine and Authorization</i>	47
<i>How the Metadata Engine Constructs a LIBNAME Statement</i>	47
<i>How the Metadata Engine Constructs Options</i>	47
<i>Examples: CONOPTSET= or LIBOPTSET= Argument</i>	48
<i>Examples: OPTSET= Data Set Option</i>	50

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.

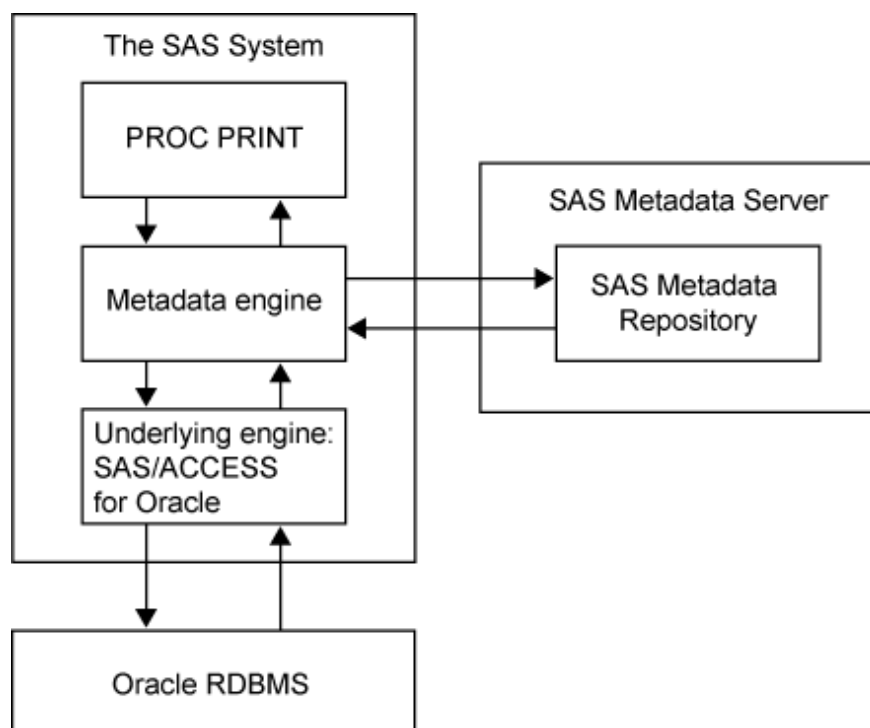
The objects contain the specifications that you would normally submit with a LIBNAME statement. The metadata engine uses the information in the objects to construct a LIBNAME statement that specifies the data source, the engine that processes the data (referred to as the *underlying engine*), and options.

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 6.1 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 and data set options that are set in the metadata by an administrator.
- Passes data set options directly to the underlying engine, including SAS file passwords. (If a password is required, but it is not submitted or is incorrect, and you are running interactively, SAS displays a dialog box. You can specify a password that lasts for the duration of the SAS session.)
- Supports SQL implicit pass-through.

The DBLOAD procedure and the LOCK statement are not supported.

For metadata requirements, see “Overview of Metadata Requirements” on page 59.

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 47.
- 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.

How the Metadata Engine Constructs a LIBNAME Statement

As noted in “Overview of the Metadata LIBNAME Engine” on page 45, 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.

How the Metadata Engine Constructs Options

Each option is constructed from the PropertyName, Delimiter, DefaultValue, and UseValueOnly attributes in a Property object. The default option values are represented

by one or more Property objects. These Property objects are associated directly with an owning object (a SASLibrary, SASClientConnection, or PhysicalTable object).

To provide users with a different set of values from the default option values, an administrator can group one or more Property objects by using a PropertySet object. If you do not want the default options, use the LIBOPTSET= or CONOPTSET= argument, or the OPTSET= data set option to specify a PropertySet object. One or more PropertySet objects can be associated with an owning object.

However, only the default options or one PropertySet object can be used with an owning object at one time. For more information about the metadata objects, see Property and PropertySet in “How the Metadata Engine Uses SAS Metadata Types” on page 66.

Because some options do not take delimiters and a value, the UseValueOnly attribute specifies whether to use the DefaultValue only, or to precede it with the PropertyName and Delimiter. Notice in the examples when UseValueOnly is set to 0, and when it is set to 1. See “Examples: CONOPTSET= or LIBOPTSET= Argument” on page 48 and “Examples: OPTSET= Data Set Option” on page 50.

See also “Example: Creating a PropertySet Object for Use with LIBOPTSET=” on page 8.

Examples: CONOPTSET= or LIBOPTSET= Argument

This example shows how the metadata engine constructs a LIBNAME statement for the underlying engine. In particular, the example illustrates how option values are obtained from metadata objects.

In this example, the user assigns a libref to a SASLibrary object that has the unique identifier AD000001. This object represents the physical SAS library. The metadata engine uses the SASLibrary object AD000001 to obtain all of the necessary values to construct a LIBNAME statement for the underlying engine (in this case, the V9 engine).

The following table shows how the option values (for the underlying engine) are obtained from the attributes of metadata objects.

Table 6.1 Metadata for a Base SAS LIBNAME Statement

Metadata Object	Relevant Object Attributes	Language Element	Value
SASLibrary (unique identifier = AD000001)	Libref='sas9'	libref	SAS9
SASLibrary (same object as above)	Engine='V9'	engine	V9
Directory	DirectoryName='sales' IsRelative='1'	'SAS-data-library'	sales
Directory (parent to above Directory object)	DirectoryName='C:\' IsRelative='0'	'SAS-data-library'	C:\
Property	PropertyName='repeempty' Delimiter='=' DefaultValue='no' UseValueOnly='0'	option	REPEMPTY=NO

Metadata Object	Relevant Object Attributes	Language Element	Value
Property	PropertyName='access' Delimiter='=' DefaultValue='readonly' UseValueOnly='0'	option	ACCESS=READONLY
Property	DefaultValue='nodltrunchk' UseValueOnly='1'	option	NODLTRUNCHK
Property	DefaultValue='extend' UseValueOnly='1'	option	EXTEND
PropertySet	Name='basetrn'	group of options	LIBOPTSET=BASETRUN

With the metadata already registered in a repository, the user submits the following metadata LIBNAME statement to SAS:

```
libname mytest meta libid=AD000001 repname=sasrepos
      ipaddr='a123.us.company.com' port=8561
      userid=sasabc pw=srvpw;
```

The metadata engine constructs the following LIBNAME statement for the V9 engine:

```
libname sas9 v9 'C:\sales' repempty=no access=readonly nodltrunchk extend;
```

The next example illustrates the use of a PropertySet object with the LIBOPTSET argument. The following table shows the option values that are represented by a PropertySet object named BASETRUN, which groups several Property objects. Three of the option values are identical to the default option values for the data library. One value, the z/OS option DLTRUNCHK, is different from the default option value.

Table 6.2 The BASETRUN PropertySet Object

SAS Option	PropertyName Attribute	Delimiter Attribute	DefaultValue Attribute	UseValueOnly Attribute
REPEMPTY=NO	REPEMPTY	=	NO	0
ACCESS=READONLY	ACCESS	=	READONLY	0
NODLTRUNCHK			DLTRUNCHK	1
EXTEND			EXTEND	1

With the metadata already registered in a repository, the user submits the following metadata LIBNAME statement to SAS:

```
libname mytest meta libid=AD000001 repname=sasrepos
      ipaddr='a123.us.company.com' port=8561
      userid=sasabc pw=srvpw liboptset=basetrn;
```

The metadata engine constructs a LIBNAME statement as in the previous example, but with the DLTRUNCHK option instead of the default option:

```
libname sas9 v9 'C:\sales' repempty=no access=readonly dltrunchk extend;
```

Examples: OPTSET= Data Set Option

Here are some examples with and without the OPTSET= data set option:

- In the following PRINT procedure, the metadata engine uses the SAS/ACCESS Interface to Oracle engine as its underlying engine. No options are specified in the PROC PRINT statement, and no OPTSET= value is specified. Therefore, this PROC PRINT uses the default options. The default option values are obtained from Property objects that are associated with the PhysicalTable object.

```
proc print data=x.dept;  
run;
```

- The next PROC PRINT uses the SAS/ACCESS Interface to Oracle engine. Options are specified in the PROC PRINT statement, so they take precedence over option values in the metadata.

```
proc print data=oralib.dept (dbnull=(empid=no jobcode=no));  
run;
```

- In the next PROC PRINT, option values are obtained from the NULLSET PropertySet object:

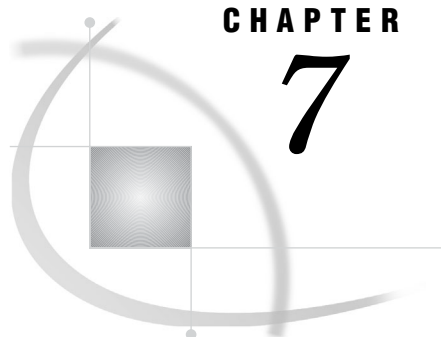
```
proc print data=x.dept (optset=nullset);  
run;
```

- In the next procedure, option values are obtained from the OPTS2 PropertySet object:

```
proc append base=work.sasbig data=mymeta.bigtable (optset='opts2');  
run;
```

- In this DATA step, option values are obtained from the DS1 PropertySet object:

```
data a;  
  set x.b(optset=ds1);  
run;
```



CHAPTER

7

Reference for the Metadata Engine

<i>LIBNAME Statement for the Metadata Engine</i>	51
<i>Overview: Metadata LIBNAME Statement</i>	51
<i>Syntax: Metadata LIBNAME Statement</i>	51
<i>Required Arguments</i>	52
<i>Server Connection Arguments</i>	53
<i>METAOUT= Argument</i>	54
<i>CONOPTSET= and LIBOPTSET= Arguments</i>	55
<i>SAS Data Set Options for the Metadata Engine</i>	55
<i>METAOUT= Data Set Option</i>	55
<i>Overview: METAOUT= Data Set Option</i>	55
<i>Syntax: METAOUT= Data Set Option</i>	56
<i>OPTSET= Data Set Option</i>	56
<i>Overview: OPTSET= Data Set Option</i>	56
<i>Syntax: OPTSET= Data Set Option</i>	57

LIBNAME Statement for the Metadata Engine

Overview: Metadata LIBNAME Statement

To learn how the metadata engine works, see Chapter 6, “Introduction to the Metadata LIBNAME Engine,” on page 45.

The SAS Metadata Server must be running before you submit the metadata LIBNAME statement. The required metadata must already exist in the metadata server. (If you specify METAOUT=DATA, table metadata is not required.) For the necessary metadata, see Chapter 8, “Reference to Metadata Objects for the Metadata Engine,” on page 59.

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

Syntax: Metadata LIBNAME Statement

LIBNAME *libref*

META

LIBID=<">*identifier*<"> | **LIBRARY**=<">*name*<"> |

```

LIBRARY="/folder-pathname/name" | LIBURI="URI-format"
<server-connection-arguments>
<METAOUT=ALL | DATA | DATAREG | META>
<CONOPTSET="propertyset-object">
<LIBOPTSET="propertyset-object">;

```

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<"> | LIBRARY="/folder-pathname/
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 2, "Metadata Object Identifiers and URIs," on page 5.

```

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=

```

LIBRARY="/folder-pathname/name"

```

specifies the folder pathname and the value in the SASLibrary object's Name= attribute. The pathname is the object's location in a SAS folder. The pathname begins with a forward slash. An example is **library="/Users/Dmitri/My Folder/test/mylib"**. The maximum length is 256 characters.

```

LIBURI="URI-format"

```

specifies a URI, which is a standard from SAS Open Metadata Architecture. For more information, see Chapter 2, "Metadata Object Identifiers and URIs," on page 5. 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 24.

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 39. The maximum length is 256 characters.

Alias: HOST=
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 33. The maximum length is 256 characters.

Alias: METAPASS=
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 34. The default for the METAPORT= system option is 8561. The range is 1–65535.

Alias: METAPORT=

PROTOCOL=BRIDGE

specifies the network protocol for connecting to the metadata server. If you do not specify this argument, the value of the METAPROTOCOL= system option is used; for more information, see “METAPROTOCOL= System Option” on page 37. In this release, the only supported value is BRIDGE, which specifies the SAS Bridge protocol.

Alias: METAPROTOCOL=

Requirement: Do not enclose the value in quotation marks.

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 38. The default for the METAREPOSITORY= system option is **Foundation**.

REPID=<">*identifier*<">

specifies an 8-character identifier. This identifier is the first half of the SASLibrary’s 17-character identifier, and is the second half of the repository’s identifier. For more information, see Chapter 2, “Metadata Object Identifiers and URIs,” on page 5.

REPNAME=<">*name*<">

specifies the value in the repository's Name= attribute. The maximum length is 256 characters.

Alias: METAREPOSITORY=
REPOS=
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 41. The maximum length is 256 characters.

Alias: ID=
METAUSER=
USERID=

METAOUT= Argument

METAOUT=ALL | DATA | DATAREG | META

specifies the metadata engine's output processing of tables in the data source.

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 can use a product like SAS Management Console, or you can submit the METALIB procedure. For more information, see Chapter 12, "METALIB Procedure," on page 99.

Restriction: 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.

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

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, and delete physical tables.

Interaction: The user can access any table, regardless of whether it has been defined in the repository.

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.

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

CONOPTSET= and LIBOPTSET= Arguments

If you do not specify the CONOPTSET= or LIBOPTSET= arguments, the default options for the constructed LIBNAME statement are used. For more information, see “How the Metadata Engine Constructs Options” on page 47. See also “Examples: CONOPTSET= or LIBOPTSET= Argument” on page 48.

CONOPTSET=<">propertyset-object<">

specifies a PropertySet object that is associated with the SASClientConnection object that corresponds to the SASLibrary object specified by the LIBID=, LIBRARY=, or LIBURI= argument. The Property objects that are associated with this PropertySet object are used as connection arguments for the constructed LIBNAME statement for the underlying engine. A SASClientConnection object exists only for engines that connect to a server, such as the SAS/ACCESS engines. The maximum length is 60 characters.

LIBOPTSET=<">propertyset-object<">

specifies a PropertySet object that is associated with the SASLibrary object specified by the LIBID=, LIBRARY=, or LIBURI= argument. The Property objects that are associated with this PropertySet object are used as statement arguments for the constructed LIBNAME statement for the underlying engine. The maximum length is 60 characters.

SAS Data Set Options for the Metadata Engine

METAOUT= Data Set Option

Overview: METAOUT= Data Set Option

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

Note: While 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. Δ

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

Syntax: METAOUT= Data Set Option

METAOUT=ALL | DATA | DATAREG | META

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 can use a product like SAS Management Console, or you can submit the METALIB procedure. For more information, see Chapter 12, “METALIB Procedure,” on page 99.

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

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.

Interaction: The user can access any table, regardless of whether it has been defined in the repository.

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.

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

OPTSET= Data Set Option

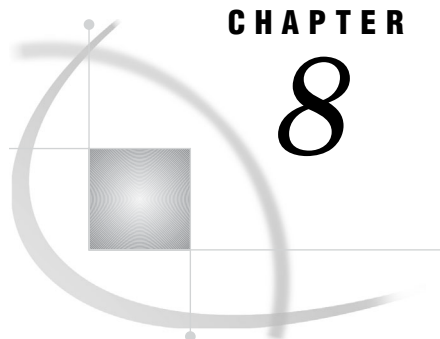
Overview: OPTSET= Data Set Option

If you do not specify the OPTSET= data set option, the default options for the constructed LIBNAME statement are used. For more information, see “How the Metadata Engine Constructs Options” on page 47. See also “Examples: OPTSET= Data Set Option” on page 50.

Syntax: OPTSET= Data Set Option

OPTSET=<">propertyset-object<">

specifies a PropertySet metadata object that is associated with the PhysicalTable object that corresponds to the table that is being referenced. The PropertySet object specifies data set options that will be applied to the table.



CHAPTER

8

Reference to Metadata Objects for the Metadata Engine

<i>Overview of Metadata Requirements</i>	59
<i>Diagrams of the SAS Metadata Model</i>	59
<i>Metadata Objects, Listed by Language Element</i>	63
<i>The Constructed LIBNAME Statement for a Base SAS Engine</i>	63
<i>The Constructed LIBNAME Statement for a DBMS SAS/ACCESS Engine</i>	64
<i>The Constructed LIBNAME Statement for the REMOTE Engine</i>	65
<i>Metadata Objects, Listed by Type</i>	66
<i>What Is a Metadata Type?</i>	66
<i>How the Metadata Engine Uses SAS Metadata Types</i>	66

Overview of Metadata Requirements

This chapter lists the minimum requirements to use the metadata engine. An administrator can use a product like SAS Management Console to define metadata.

- For the metadata engine to construct a LIBNAME statement for the underlying engine and to process data, the metadata must be available from an existing SASLibrary metadata object, and it must conform to metadata engine model requirements as described in this book.
- The metadata must be consistent with the data source. For example, if metadata defines a column as numeric data, then the data source must define the column as numeric data.
- For the metadata engine to access members in a SAS library, the SASLibrary object must have an associated DatabaseSchema object for a DBMS SAS/ACCESS engine, or a Directory object for a Base SAS library.
- The SAS Metadata Server must be running before you submit the metadata LIBNAME statement.
- For the metadata engine to apply metadata server authorization, the authorization metadata must be defined. Authorizations can control both the availability of specific metadata (ReadMetadata permission) and the actions that can be taken on the data source (Read, Write, Create, and Delete permissions). For more information, see “The Metadata Engine and Authorization” on page 47.

Diagrams of the SAS Metadata Model

The metadata engine uses the SAS Metadata Model as a framework and a common format for metadata modeling. For the metadata engine to access metadata objects that are stored in a SAS Metadata Repository, the library metadata must be configured so that the metadata engine can process it.

The metadata engine supports the following models:

Relational DBMS Model	models a DBMS library, which uses a DBMS SAS/ACCESS engine.
SAS Data Set Model	models a Base SAS library, which uses a Base SAS engine.
Remote Relational DBMS Model	models a remote DBMS library, which uses a SAS/ACCESS engine and the REMOTE engine.
Remote SAS Data Set Model	models a remote Base SAS library, which uses a Base SAS engine and the REMOTE engine.

The following diagrams illustrate the associations between related metadata types in each supported model. The purpose of the diagrams is to help you understand the relationships among the metadata types in the SAS metadata model.

Figure 8.1 Relational DBMS Model

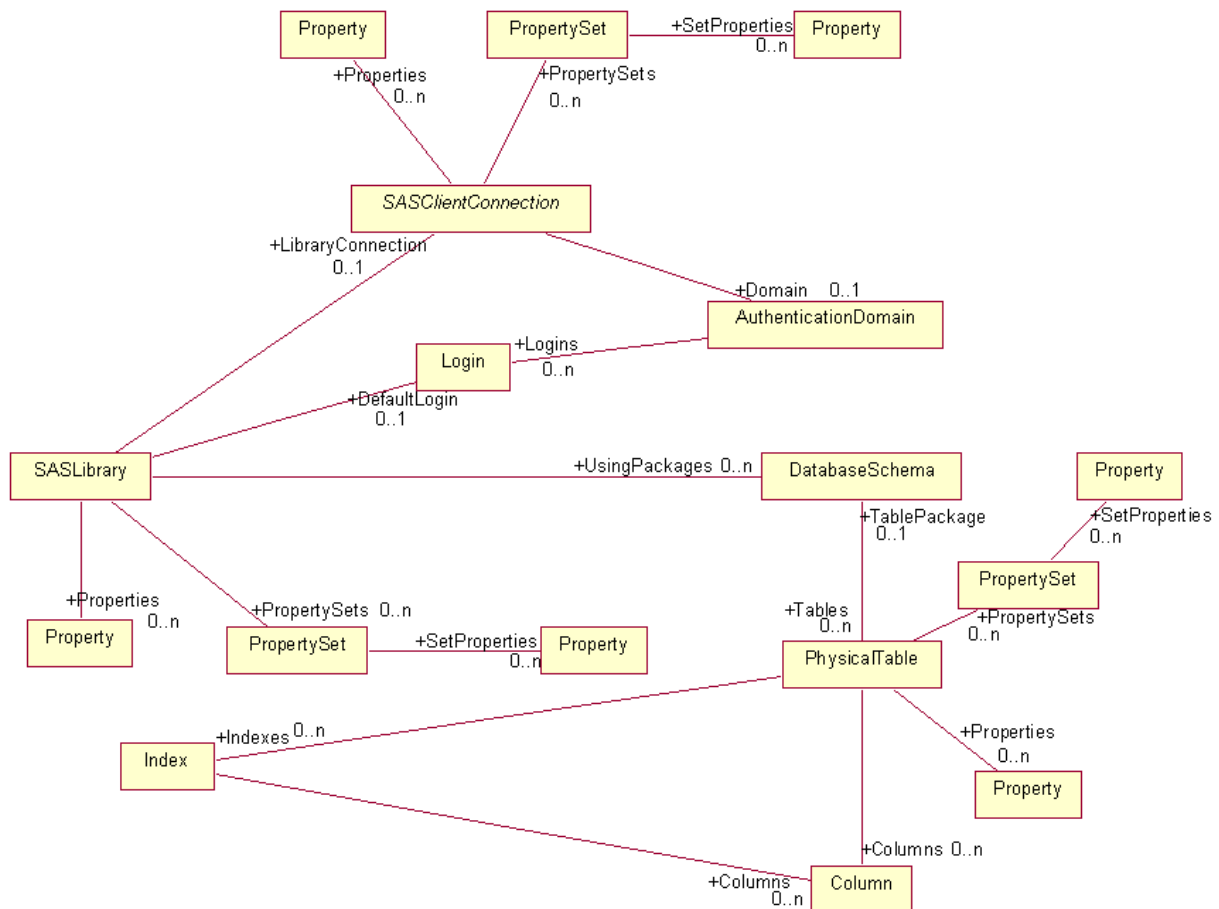


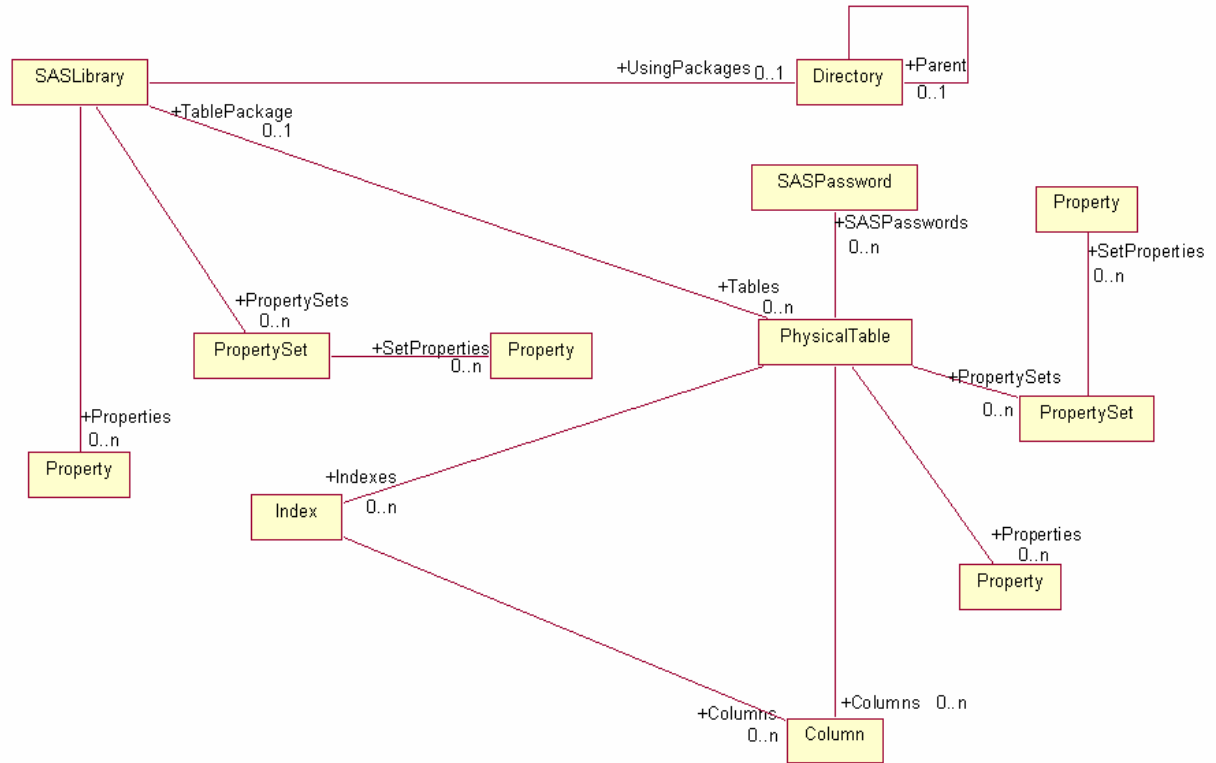
Figure 8.2 SAS Data Set Model

Figure 8.3 Remote Relational DBMS Model

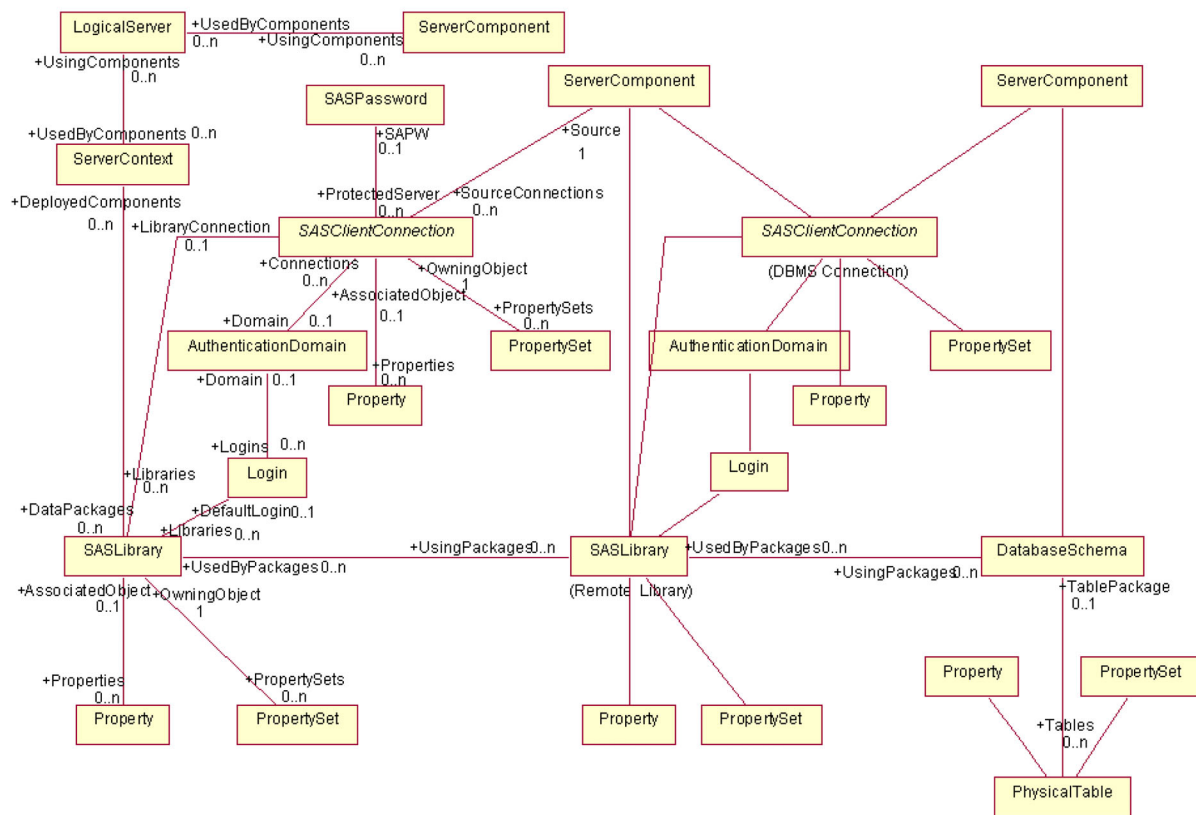
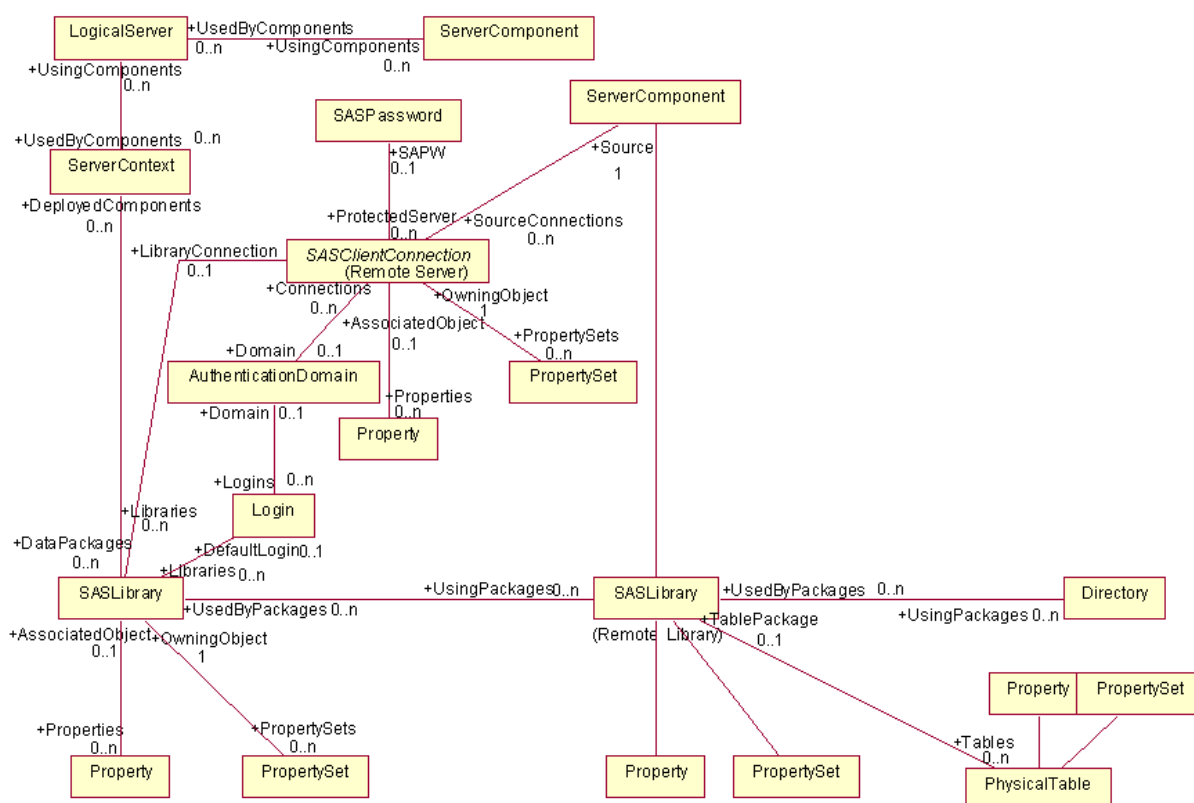


Figure 8.4 Remote SAS Data Set Model



Metadata Objects, Listed by Language Element

The Constructed LIBNAME Statement for a Base SAS Engine

The metadata engine constructs a LIBNAME statement based on the stored metadata and uses this LIBNAME statement to access your data. The construction is transparent; however, it is important to understand the components of the LIBNAME statement so that you can configure the metadata appropriately with a product like SAS Management Console. For more information about LIBNAME syntax, see *SAS Language Reference: Dictionary* for LIBNAME syntax.

Note: The metadata must already be registered. If you specify METAOUT=DATA, table metadata is not required. \triangle

LIBNAME *libref* <engine> 'SAS-data-library' <options>;

libref

references the SAS library that the Base SAS engine will process. This value is obtained from the Libref attribute in the SASLibrary object.

engine

is the name of the Base SAS engine that will process the SAS library. This value is obtained from the Engine attribute in the SASLibrary object.

'SAS-data-library'

is the physical name of the SAS library. This value is obtained from the DirectoryName attribute in the Directory object that is associated with the SASLibrary object. The DirectoryName attribute must include any delimiters that are specific to the operating environment.

The Directory object also contains the IsRelative attribute, which indicates whether the value in the DirectoryName attribute is the complete physical name or is relative to a parent directory. If the IsRelative attribute is set to true (1), the metadata engine retrieves the parent directory and obtains the value from its DirectoryName attribute. The metadata engine appends the parent directory's physical name to the beginning of the subdirectory's physical name.

A parent directory can have an IsRelative attribute set to true (1). Several parent directories can be appended to the beginning of the physical name before it is complete.

options

are the LIBNAME statement options for the Base SAS engine. Each option is constructed from the PropertyName, Delimiter, DefaultValue, and UseValueOnly attributes in a Property object.

Property objects can be associated directly with the SASLibrary object. PropertySet objects can associate different groups of Property objects with the SASLibrary object. For more information, see "How the Metadata Engine Constructs Options" on page 47.

The Constructed LIBNAME Statement for a DBMS SAS/ACCESS Engine

The metadata engine constructs a LIBNAME statement for a DBMS SAS/ACCESS engine based on the stored metadata and uses this LIBNAME statement to access your data. The construction is transparent; however, it is important to understand the components of the LIBNAME statement so that you can configure the metadata appropriately with a product like SAS Management Console. For more information about LIBNAME syntax, see the documentation for the specific SAS/ACCESS product.

Note: The metadata must already be registered. If you specify METAOUT=DATA, table metadata is not required. Δ

LIBNAME *libref* SAS/ACCESS-engine-name<SAS/
ACCESS-engine-connection-options> <SAS/ACCESS-LIBNAME-options>;

libref

references the data that the SAS/ACCESS engine will process. This value is obtained from the Libref attribute in the SASLibrary object.

SAS/ACCESS-engine-name

is the name of the underlying engine that will process the data. This value is obtained from the Engine attribute in the SASLibrary object.

SAS/ACCESS-engine-connection-options

are the engine-specific connection options for the LIBNAME statement for the SAS/ACCESS engine. Each option is constructed from the PropertyName, Delimiter, DefaultValue, and UseValueOnly attributes in a Property object.

Property objects can be associated directly with the SASClientConnection object that is associated with the SASLibrary object. PropertySet objects can associate different groups of Property objects with the SASClientConnection object. For more information, see “How the Metadata Engine Constructs Options” on page 47.

Exception: The values for the USERID= and PASSWORD= connection options are obtained from a Login object. The Login object can be associated with the SASLibrary object. If no Login object is associated with the SASLibrary object, a Login object is obtained from the AuthenticationDomain object that is associated with the SASClientConnection object.

SAS/ACCESS-LIBNAME-options

are the LIBNAME statement options for the SAS/ACCESS engine. Each option is constructed from the PropertyName, Delimiter, DefaultValue, and UseValueOnly attributes in a Property object.

Property objects can be associated directly with the SASLibrary object. PropertySet objects can associate different groups of Property objects with the SASLibrary object. For more information, see “How the Metadata Engine Constructs Options” on page 47.

Exception: The value for the SCHEMA= option is obtained from a DatabaseSchema object. The DatabaseSchema object must be associated with the SASLibrary object if the SAS library is processed by a SAS/ACCESS engine.

The Constructed LIBNAME Statement for the REMOTE Engine

The metadata engine constructs a LIBNAME statement for the REMOTE engine based on the stored metadata and uses this LIBNAME statement to access your data. The construction is transparent; however, it is important to understand the components of the LIBNAME statement so that you can configure the metadata appropriately with a product like SAS Management Console. For more information about LIBNAME syntax, see the *SAS/SHARE User's Guide*.

Note: The metadata must already be registered. If you specify METAOUT=DATA, table metadata is not required. △

LIBNAME *libref* SERVER=*serverid* <ACCESS=READONLY> <OUTREP=*format*>
 <USER=*userid*> <PASSWORD=*password*> <SAPW=*server-access-password*>
 <SLIBREF=*server-libref*> ;

libref

references the *local* SAS library. This value is obtained from the Libref attribute in the SASLibrary object for the local SAS library. This value can be the same as the value for the Libref attribute in the SASLibrary object for the *remote* SAS library.

SERVER=*serverid*

specifies a name for the SAS server. This value is constructed from the PropertyName, Delimiter, DefaultValue, and UseValueOnly attributes in a Property object that is associated with the SASClientConnection object for the SAS server.

ACCESS=READONLY

controls read access to a data library (in this case, through the SAS server). This value is obtained from the PropertyName, Delimiter, DefaultValue, and UseValueOnly attributes in a Property object that is associated with the local library's SASLibrary object.

OUTREP=*format*

creates new files in a foreign host format. This value is obtained from the `PropertyName`, `Delimiter`, `DefaultValue`, and `UseValueOnly` attributes in a `Property` object that is associated with the local library's `SASLibrary` object.

`USER=user-name`

specifies the user ID of the accessing client on the server. This value is obtained from the `Userid` attribute in a `Login` object that is associated with the `SASLibrary` object.

`PASSWORD=password`

specifies the password for the accessing client on the server. This value is obtained from the `Password` attribute in a `Login` object that is associated with the `SASLibrary` object.

`SAPW=server-access-password`

specifies a server access password, which is used to gain access to the SAS server. This value is obtained from the `Password` attribute in a `SASPassword` object that is associated with the `SASClientConnection` object for the SAS server.

`SLIBREF=server-libref`

specifies the libref that is used by the server to identify a SAS library. This value is obtained from the `Libref` attribute in the remote library's `SASLibrary` object.

Metadata Objects, Listed by Type

What Is a Metadata Type?

The SAS metadata model includes metadata types for the most commonly used SAS applications. A metadata type is like a template. Each metadata object is a unique instance of a metadata type. For example, the metadata type `Column` models the metadata for a variable in a SAS data set. Each `Column` object has a unique name, but they are all of the metadata type `Column`.

A metadata object is also known as a metadata definition. For more information about the applications that use SAS metadata, see the *SAS Intelligence Platform: System Administration Guide*.

How the Metadata Engine Uses SAS Metadata Types

The following list of SAS metadata types is taken from the SAS Open Metadata Architecture. With the exception of `PhysicalTable` and `Column`, metadata must already be defined in the repository.

AuthenticationDomain

represents the domain that controls user access to the server for the underlying engine. User login information is retrieved from the authentication domain when no default login information has been provided with the SAS library. This user login information is included in the `LIBNAME` statement constructed for the underlying engine. An `AuthenticationDomain` object does not exist if user IDs and passwords are not used to connect to the server, or if a Base SAS engine is the underlying engine.

Attributes:

not applicable

Associations:

Logins (0..n)

This association from the AuthenticationDomain object to Login object(s) is used to retrieve only the user logins (Login objects) that the user is authorized (through SAS Open Metadata Architecture security) to use. The logic for retrieving user login information is in the description for the Login object. The cardinality on this association enables an authentication domain to control several user logins.

Column

represents a column on a library member. Column metadata is retrieved when opening a library member, and when retrieving indexes on an opened library member.

Attributes: The metadata engine uses the following attributes from the Column object:

Desc

The SAS column label field.

SASColumnName

The column name that is used by SAS. This attribute must be populated.

SASColumnType

The column type that is used by SAS. This attribute must be populated.

SASColumnLength

The column length that is used by SAS. This attribute must be populated.

SASFormat

The SAS format that is applied to the column.

SASInformat

The SAS informat that is applied to the column.

Associations:

not applicable

DatabaseSchema

represents a DBMS schema. The SCHEMA= option cannot be represented with a Property object. The DBMS schema must be represented with a DatabaseSchema object. The DatabaseSchema object, if available, must be associated with a SASLibrary object. Only one schema can be associated with the SAS library. If more than one schema is retrieved, the first schema is used.

DBMS schema metadata is retrieved to obtain the name of the DBMS schema that is used in the LIBNAME statement for the underlying engine. The DBMS schema metadata is retrieved after the LIBNAME statement for the metadata engine is executed. The DBMS schema is used as a point of reference when retrieving library member information and when opening a library member.

Attributes:

SchemaName

The name of the DBMS schema. The metadata engine includes this schema name as the value for the SCHEMA= option in the LIBNAME statement for the underlying engine. This attribute must be populated.

Exception: The SAS/ACCESS interfaces for ODBC, OLE DB, Microsoft Access, and Microsoft Excel do not use a schema when their

underlying data source does not support schemas. In these cases, the SchemaName attribute is blank.

Associations:

Tables(0..n)

This association from the DatabaseSchema object to PhysicalTable object(s) is used to obtain all DBMS tables that are associated with the DBMS schema in the repository. The tables that are associated with the schema might differ from the tables that are under the schema in the DBMS. The cardinality on this association enables the DBMS schema to have many DBMS tables.

Directory

represents a file system directory. Directory metadata is retrieved to obtain the physical name that is used as the library specifications in the LIBNAME statement for the underlying engine. Directory metadata is retrieved after the LIBNAME statement for the metadata engine is executed. No library members are associated with a Directory object. The library members are associated with the SASLibrary object that is using this Directory object.

Attributes:

DirectoryName

Directory name, including any delimiters that are specific to the operating environment. The metadata engine treats the value of this attribute as a string, including the total string in the LIBNAME statement for the underlying engine.

Note: For the metadata engine, do not use quotation marks in the directory path. \triangle

IsRelative

Indicates that this directory is a subdirectory and has a parent directory. When this attribute is set to true (1), the DirectoryName attribute does not contain the complete name. The parent directory must be retrieved to complete the name. Several subdirectories can be involved in completing a directory name.

Associations:

Parent(0..1)

This association from the Directory object to another Directory object is used to obtain the remaining name when a directory is relative to another directory. To construct a complete directory name, the value of the DirectoryName attribute in the parent directory is appended to the beginning of the value of the DirectoryName attribute of the subdirectory. The cardinality on this association enables a subdirectory to have (at most) one parent directory. However, many subdirectories and parents might be traversed before the directory name is complete.

Index

represents an index on a library member. When index information is requested for an opened library member, index metadata is retrieved to obtain the name of the index and column(s) that compose the index.

Attributes:

IndexName

The name of the index on the library member. This attribute must be populated.

*Associations:**Columns(0..n)*

This association from the Index object to column object(s) is used to obtain the columns that make up the index. The cardinality on this association enables the index to be a simple index (of one column) or a composite index (made up of more than one column).

Login

represents a user's login information (user ID and password) that is used to connect to a server. This user login information is retrieved after the LIBNAME statement for the metadata engine is executed and is included in the LIBNAME statement constructed for the underlying engine. If no Login object is associated with the SASLibrary object, the user login information is retrieved from the AuthenticationDomain object.

Here is the logic for retrieving user login information from the AuthenticationDomain object:

- If no Login object is retrieved for the user, then no user login information is included in the LIBNAME statement for the underlying engine. It is assumed that either no user login information is required to connect to the server for the underlying engine, or the user login information has been stored in a location according to the setup for the underlying engine. A connection is attempted without user login information, and the user is notified of any connection failures.
- If there is only one Login object retrieved, this user ID and password is included in the LIBNAME statement for the underlying engine as values for the USERID= and PASSWORD= connection options.

*Attributes:**Userid*

A user ID that is used to connect to a server for the underlying engine. This attribute must be populated.

Password

The password that enables a user to access a server for the underlying engine. This attribute must be populated.

Associations:

not applicable

PhysicalTable

represents a member in a SAS library. Metadata for a library member is retrieved when a library member listing is requested, a library member is opened for use, or an index is requested. Metadata for a library member is retrieved to obtain the name and type of the member and, in some cases, the columns of the library member.

Note: To access physical data with METAOUT=ALL, the table's PhysicalTable and Column objects must be defined in the repository. To access physical data with METAOUT=DATA, the PhysicalTable and Column objects are not required. △

*Attributes:**Desc*

The SAS label field.

SASTableName

The SAS name for a library member (for example, DBMS table, SAS data set, and so on). This attribute must be populated.

MemberType

The SAS type for a library member (for example, DATA or VIEW). This attribute must be populated.

Associations:

Columns(0..n)

This association from the PhysicalTable object to Column object(s) is used to obtain the columns of an opened library member. The cardinality on this association enables a library member to have many columns.

Indexes(0..n)

This association from the PhysicalTable object to Index object(s) is used to identify the indexes of an opened library member. The cardinality on this association enables a library member to have several indexes.

Properties(0..n)

This association from the PhysicalTable object to Property object(s) is used to obtain the data set options to be used when accessing a library member. These options are considered the default data set options. They are applied each time the library member is opened unless a PropertySet object is specified in the metadata engine data set option OPTSET=. The cardinality on this association enables a library member to have an unlimited number of data set options.

PropertySets(0..n)

This association from the PhysicalTable object to PropertySet object(s) is used to obtain the data set options to be used when opening a library member. This association is used when a PropertySet object is specified in the metadata engine data set option OPTSET=. If OPTSET= is not specified, any default data set options from the Properties association is applied. The cardinality on this association enables a library member to have many PropertySet objects

SASPasswords(0..n)

This association from the PhysicalTable object to SASPassword object(s) is used to obtain the READ=, WRITE=, ALTER=, and PW= password values for a SAS data set. This association is valid only when a Base SAS engine is the underlying engine for the metadata engine. This association is used each time the SAS data set is opened. The cardinality on this association enables a SAS data set to have more than one SAS password.

TablePackage(1)

This association from the PhysicalTable object to a DatabaseSchema or SASLibrary object is used to identify the DBMS schema or SAS library, respectively, to which this library member belongs. The cardinality on this association enables a library member to belong to one DBMS schema or SAS library.

Property

represents a SAS option. A Property object can be associated with a SASLibrary object for LIBNAME options, and a SASClientConnection object for server connection options. These options are the default options that are included in the LIBNAME statement for the underlying engine. A Property object can be associated with a PhysicalTable object for data set options. In this case, these

options are the default options that are applied when the associated library member is referenced. A Property object can be associated with a PropertySet object. In this case, these options are applied only when the PropertySet object is included as a value in the metadata LIBNAME statement arguments LIBOPTSET=, CONOPTSET=, or the metadata engine data set option OPTSET=. When options within a PropertySet object are used, default options are not used. For more information, see “How the Metadata Engine Constructs Options” on page 47.

The USERID=, PASSWORD=, and SCHEMA= options cannot be represented with a Property object. User IDs and passwords must be represented with a Login object. A DBMS schema must be represented with a DatabaseSchema object. The READ=, WRITE=, ALTER=, and PW= data set password options cannot be represented with a Property object. The SAPW= remote server access password option cannot be represented with a Property object. These passwords must be represented with a SASPassword object.

Attributes:

DefaultValue

Value for the option. This attribute must be populated.

Delimiter

The delimiter between the option name and the option value. For most SAS options, the delimiter is an equal sign (=). This attribute is not used if the UseValueOnly attribute is set to true (1).

PropertyName

Name of the option. This attribute is not used if the UseValueOnly attribute is set to true (1).

UseValueOnly

Indicates a Boolean option. When this attribute is set to true (1), only the DefaultValue attribute is used to specify the option. When this attribute is set to false (0), the PropertyName, Delimiter, and DefaultValue attributes are used to specify the option.

Associations:

not applicable

PropertySet

groups a set of Property objects, representing SAS options, to be used in a particular context. PropertySet objects can be used to group a set of LIBNAME statement, connection, or data set options as follows:

- When you specify a PropertySet name in the LIBOPTSET= argument, the metadata engine includes a set of LIBNAME options in the constructed LIBNAME statement for the underlying engine. The OwningObject for these sets of Property objects is a SASLibrary object.
- When you specify a PropertySet name in the CONOPTSET= argument, the metadata engine includes a set of connection options in the constructed LIBNAME statement for the underlying engine. The OwningObject for these sets of Property objects is a SASClientConnection object.
- When you specify a PropertySet name in the OPTSET= data set option, the metadata engine applies a set of data set options to the specified table. The OwningObject for this set of Property objects is a PhysicalTable object.

Only one set of Property objects is used for an OwningObject, which is either the set of Property objects that are associated directly with the OwningObject, or

the set of Property objects that are associated with the PropertySet object specified for an OwningObject. For more information, see “How the Metadata Engine Constructs Options” on page 47 and “Example: Creating a PropertySet Object for Use with LIBOPTSET=” on page 8.

Attributes:

Name

Name of the PropertySet object. This name is the value specified for the LIBOPTSET= or CONOPTSET= argument or the OPTSET= data set option. This attribute must be populated.

Associations:

SetProperties(0..n)

This association from the PropertySet object to Property object(s) is used to obtain the options defined in the PropertySet. The cardinality of this association enables the PropertySet to contain many options.

SASClientConnection

represents information needed by SAS to connect to a server for the underlying engine.

Attributes:

not applicable

Associations:

Domain(0..1)

This association from the SASClientConnection object to an AuthenticationDomain object is used to access the authorization domain that maintains the user identities (user IDs and passwords) used to access the server. The cardinality on this association enables the client connection to be associated with (at most) one authentication domain.

Properties(0..n)

This association from the SASClientConnection to Property object(s) is used to obtain the connection options to be included in the LIBNAME statement for the underlying engine. These options are considered the default connection options and are applied each time a LIBNAME statement is constructed for the underlying engine, unless a PropertySet object is specified in the metadata engine LIBNAME statement option CONOPTSET=. The cardinality of this association enables a connection to have many connection options.

PropertySets(0..n)

This association from the SASClientConnection to PropertySet object(s) is used to obtain connection options to be included in the LIBNAME statement for the underlying engine. This association is used when a PropertySet object is specified in the CONOPTSET= argument. If CONOPTSET= is not specified, any default connection options from the Properties association is applied. The cardinality of this association enables a connection to have many sets of connection options.

SAPW(0..1)

This association from the SASClientConnection to a SASPassword object is used to obtain a SAS server access password option to be included in the LIBNAME statement for the underlying engine. The cardinality of this association enables a connection to have (at most) one SAS password.

SASLibrary

represents a SAS library. The SASLibrary object is specified by the LIBID=, LIBRARY=, or LIBURI= argument in the LIBNAME statement. The metadata engine exits with an error if the SASLibrary object is not specified in the metadata LIBNAME statement. Metadata included in this library is used to construct a LIBNAME statement for the underlying engine.

*Attributes:**Engine*

The engine that is used with this library. This attribute tells the metadata engine which engine to assign as the underlying engine. A SAS library is associated with only one engine. This attribute must be populated.

Libref

The libref that is used to assign the underlying engine. This attribute must be populated.

IsDBMSLibname

Indicates whether this library is used to construct a LIBNAME statement for a DBMS engine. If this attribute is set to true (1), then a DatabaseSchema object must be associated with this library (UsingPackages association).

*Associations:**DefaultLogin(0..1)*

This association from the SASLibrary object to a Login object is used to obtain the user login (user ID and password) metadata that is used as values for the user ID and password parameters in the LIBNAME statement. However, many engines do not require user login metadata; therefore, this association might not exist. If this association does not exist, the metadata engine attempts to obtain user login metadata from the AuthenticationDomain object through the SASClientConnection object. If user login metadata cannot be obtained through the AuthenticationDomain object, the metadata engine attempts to assign the underlying engine with a LIBNAME statement without any user login information. The cardinality of this association enables the SAS library to use (at most) one user ID and password for the LIBNAME statement.

LibraryConnection(0..1)

This association from the SASLibrary object to a SASClientConnection object is used to obtain engine connection options. These options are represented as Property objects associated with the SASClientConnection object, or a with PropertySet object that is associated with the SASClientConnection object. This association is used to retrieve user login metadata, where applicable, through the SASClientConnection object and AuthenticationDomain object. The cardinality on this association enables the SAS library to access (at most) one SASClientConnection object.

Properties(0..n)

This association from the SASLibrary object to Property object(s) is used to obtain library options that should be included in the LIBNAME statement for the underlying engine. These options are considered the default options and are applied each time a LIBNAME statement is constructed for the underlying engine unless a PropertySet object is

specified in the metadata engine LIBOPTSET= argument in the metadata LIBNAME statement. The cardinality on this association enables a SAS library to have many library options.

PropertySets(0..n)

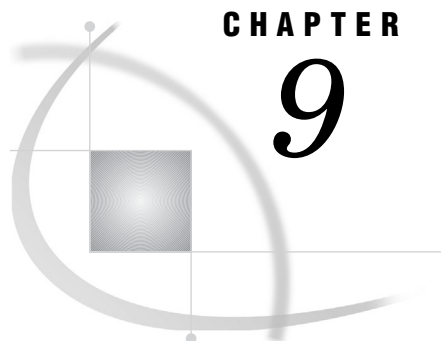
This association from the SASLibrary object to PropertySet object(s) is used to obtain LIBNAME options to be included in the LIBNAME statement for the underlying engine. This association is used when a PropertySet object is specified in the LIBOPTSET= argument in the metadata LIBNAME statement. If no LIBNAME option is specified, any default LIBNAME options from the Properties association will be applied. The cardinality of this association enables a SAS data library to have many sets of options.

UsingPackages(0..1)

This association from the SASLibrary object to either a DatabaseSchema or Directory is used to obtain metadata concerning a DBMS schema or a system directory, respectively. Only one DatabaseSchema or Directory object can be associated with a SASLibrary object.

SASPassword

represents a SAS password. The READ=, WRITE=, ALTER=, and PW= data set password options cannot be represented with a Property object. The SAPW= remote engine server password option cannot be represented with a Property object. These options must be represented with a SASPassword object. SAS passwords are retrieved when the underlying engine for the metadata engine is a Base SAS engine and a password-protected SAS data set is opened. A SASPassword can also be retrieved when the LIBNAME statement is executed for the metadata engine.



CHAPTER

9

Examples for the Metadata Engine

Example: Submitting the LIBNAME Statement 75

Example: Before and After the Metadata Engine 75

Overview 75

Using the SAS/ACCESS Interface to Oracle Engine Directly 76

Using the Metadata Engine 76

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. It also specifies PropertySet objects in the LIBOPTSET= and CONOPTSET= arguments.

```
libname myeng meta library=mylib
  repname=temp metaserver='a123.us.company.com' port=8561
  user=idxyz pw=abcdefg
  liboptset='libset2'
  conoptset='conset2'
;
```

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; ❶

proc datasets library=oralib; ❷
quit;

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

data work.temp;
    set oralib.Sales (dbindex=myindex); ❹
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
    liboptset=myopts;
```

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=, and from other objects that are associated with the SASLibrary object, such as DatabaseSchema, SASClientConnection, Login, or Property objects.

The following table lists objects that are returned from the query. The table does not show the SASClientConnection objects and PropertySet objects.

Table 9.1 SASLibrary Object Query Results

SASLibrary	DatabaseSchema	login	Property	Property	Property
Libref: oralib	SchemaName:	User:	PropertyName:	PropertyName:	PropertyName:
	myschema	myuser	preserve_tab_names	path	Connection
Engine:		Password:	DefaultValue:	DefaultValue:	DefaultValue:
oracle		mypw	yes	ora_dbms	sharedread
isDBMSLibname:			Delimiter: =	Delimiter: =	Delimiter: =
TRUE			UseValueOnly:	UseValueOnly:	UseValueOnly:
			False	False	False

Note: The relational DBMS model in “Diagrams of the SAS Metadata Model” on page 59 can help you determine how to associate these objects with one another. Δ

- From the information embedded in the objects, and from information that is implied in the relational DBMS model, 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;
```

- 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.)

- 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 (optset=myopts);
run;
```

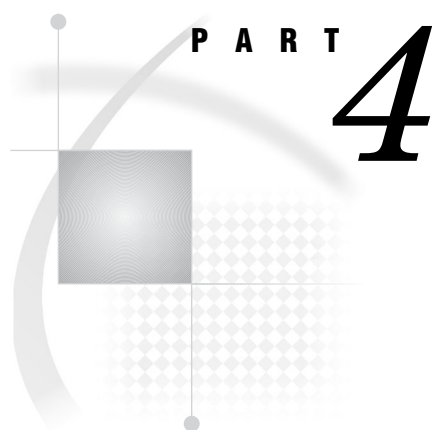
The metadata includes PhysicalTable, Column, and Property object information. The OPTSET= data set option, a metadata engine option, tells the metadata engine to return Property objects that are associated with the MYOPTS PropertySet object. The Property objects contain the values of data set options that have been customized for this table. These data set options are used by the Oracle engine while processing the data.

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 The relational DBMS model allows you to store index information for tables. Each Index object is associated with a table and with the columns that make up the index. 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. In the following statements, the OPTSET= data set option specifies to use indexes to process the table. The metadata engine uses the index information that is stored in the repository:

```
data work.temp;  
    set metaeng.Sales (optset=index_opts);  
run;
```

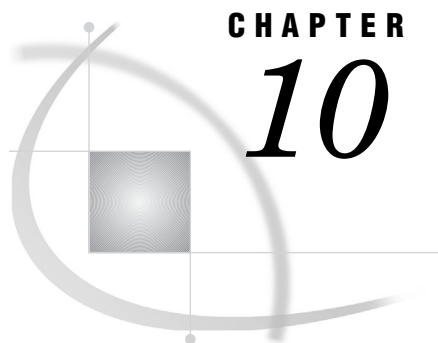
Procedures

Chapter 10.....**Introduction to Procedures for Metadata** 81

Chapter 11.....**METADATA Procedure** 83

Chapter 12.....**METALIB Procedure** 99

Chapter 13.....**METAOPERATE Procedure** 117



CHAPTER

10

Introduction to Procedures for Metadata

Overview of Procedures for Metadata 81

Comparison of the METADATA Procedure and the METAOPERATE Procedure 81

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. For more information, see the administration books in “Recommended Reading” on page 207.

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

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

To submit the procedures, you must establish a connection with the metadata server. You can specify connection information in the procedure, in system options, or in a dialog box. For more information, see “Connection Options” on page 24.

Comparison of the METADATA Procedure and the METAOPERATE Procedure

The METAOPERATE procedure and the METADATA procedure perform some of the same tasks. The benefit of PROC METAOPERATE is simpler syntax. The benefit of PROC METADATA is a broader range of tasks (you can submit many different SAS Open Metadata methods). In addition, PROC METADATA creates XML output that you can use in another program (for example, to run reports). In addition, PROC METADATA supports all parameters of the methods it submits, and some of those parameters are not supported by PROC METAOPERATE.

Here is an example that uses PROC METAOPERATE to check whether the metadata server is paused or running:

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

The 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 query with the following code:

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

The 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.

```
<Status><Metadata><ModelVersion>11.02</ModelVersion>
<PlatformVersion>9.2.0.0</PlatformVersion><ServerState>ONLINE</ServerState>
<PauseComment/><ServerLocale>en_US</ServerLocale></Metadata></Status>
```

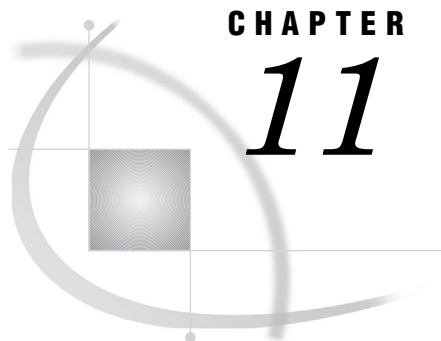
PROC METADATA can submit parameters that are not supported by PROC METAOPERATE. Here is an example:

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

The metadata server returns the journal state:

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

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, another choice is the metadata DATA step functions. For more information, see Chapter 14, “Introduction to DATA Step Functions for Metadata,” on page 131.



CHAPTER

11

METADATA Procedure

Overview: <i>METADATA Procedure</i>	83
Syntax: <i>METADATA Procedure</i>	84
<i>PROC METADATA Statement</i>	84
Concepts: <i>METADATA Procedure</i>	87
<i>Introduction to the Input XML String</i>	87
<i>The Entire Method Is an XML Element</i>	88
<i>A Method's Parameter Is an XML Element</i>	88
<i>A Metadata Object Is an XML Element</i>	88
<i>A Metadata Association Is an XML Element</i>	89
<i>See Also</i>	89
Results: <i>METADATA Procedure</i>	89
Examples: <i>METADATA Procedure</i>	90
<i>Example 1: Change a Repository's State</i>	90
<i>Example 2: Simple Request for Server Status</i>	90
<i>Example 3: Filerefs with the IN= and OUT= Arguments</i>	91
<i>Example 4: Fileref to a Temporary File with the IN= Argument</i>	92
<i>Example 5: HEADER= Argument</i>	93
<i>Example 6: VERBOSE Argument</i>	94
<i>Example 7: Request the Metadata for One Object</i>	95
<i>Example 8: Request the Metadata for One Type of Object</i>	96

Overview: METADATA Procedure

The METADATA procedure sends an XML string to the SAS Metadata Server. The XML string, which you specify with the IN= argument, contains a method call. You form the method call as if you were submitting it with the inMetadata parameter of the DoRequest method. These methods are all part of the SAS Open Metadata Interface. For more information, see “Concepts: METADATA Procedure” on page 87.

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 81.

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, 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 *SAS Metadata Model: Reference*.

Featured in: “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 9

PROC METADATA <server-connection-arguments>

```
IN = "XML-formatted-method-call" | fileref
<OUT = fileref>
<HEADER = NONE | SIMPLE | FULL>
<VERBOSE>;
```

Task	Statement
Send an XML string to the metadata server	“PROC METADATA Statement” on page 84

PROC METADATA Statement

PROC METADATA <server-connection-arguments>

```
IN = "XML-formatted-method-call" | fileref
<OUT = fileref>
<HEADER = NONE | SIMPLE | FULL>
<VERBOSE>;
```

Task	Argument
Connect to the metadata server	<i>server-connection-arguments</i>
Specify an XML-formatted method call, or an XML file that contains the method call	IN=
Save the XML output in a file	OUT=

Task	Argument
Add a character-set encoding tag to the XML output	HEADER=
Write the input XML string to the log	VERBOSE

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 24.

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 33. The maximum length is 512 characters.

Alias: METAPASS=
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 34. The default for the METAPORT= system option is 8561. The range is 1–65535.

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 37. In this release, the only supported value is BRIDGE, which specifies the SAS Bridge protocol.

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 38. The default for the METAREPOSITORY= system option is **Foundation**. The maximum length is 32,000 characters.

Alias: METAREPOSITORY=
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 39. The maximum length is 256 characters.

Alias: `HOST=`
 `IPADDR=`
 `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 41. The maximum length is 256 characters.

Alias: `ID=`
 `METAUSER=`
 `USERID=`

Input Argument

IN= *“XML-formatted-method-call”* | *fileref*

specifies an XML-formatted method call, or specifies an XML file that contains the method call.

You form the method call as if you were submitting it with the `inMetadata` parameter of the `DoRequest` method. The methods are part of the SAS Open Metadata Interface. For more information, see “Concepts: METADATA Procedure” on page 87. See also Example 3 on page 91 and Example 4 on page 92.

Note: Under z/OS, fixed-length records in the XML method call are not supported by PROC METADATA. Specify `RECFM=V` when you create the XML method call. △

Output Arguments

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 89. See also Example 3 on page 91.

Note: 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. △

Note: 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. △

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 on page 93.

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 used when creating the output XML file. In most cases, the encoding is recognized by Web browsers and XML parsers. However, if you encounter errors, you might want to know where SAS finds the encoding, and how SAS uses the encoding.

- 1 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 argument, or from the ENCODING= system option.
- 2 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*.

VERBOSE

specifies to print the input XML string after it has been preprocessed. For more information, see Example 6 on page 94.

Concepts: METADATA Procedure

Introduction to the Input XML String

The IN= argument of PROC METADATA submits an XML-formatted method call 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 class
- the Status method in the IServer class.

The methods are documented with many usage examples in *SAS Open Metadata Interface: Reference*. 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* for help in understanding concepts such as flags, filters, and templates. The following topics provide a brief introduction to the usage with PROC METADATA.

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 Status. The request starts and ends with Status tags. Do not include the DoRequest method in the XML string, because the procedure calls DoRequest for you.

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

A Method's Parameter Is an XML Element

A method's parameters are represented as XML elements, which are nested within the method's XML element. In the following example, the <Metadata> element represents the Status method's InMetadata= parameter. You always use a <Metadata> element to represent an inMetadata= or inMeta= parameter.

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

When you read the documentation about the Status method in *SAS Open Metadata Interface: Reference*, you learn that the inMetadata= parameter can submit an XML string itself. The XML string is represented as XML elements that are nested within the <Metadata> element.

With the Status method, if nothing is submitted within the <Metadata> element, the metadata server returns the default status information. In the previous example, however, an XML string is submitted. The string is an <OMA> element. The documentation for the Status method states that the <OMA> element requests the value for a specified attribute. In this example, the requested attribute is JournalSpaceAvailable. The metadata server returns the information to the SAS log:

```
<Status><Metadata><OMA JournalSpaceAvailable="200000000"/></Metadata></Status>
```

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 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, a `PhysicalTable` object has a `Columns` association to a `Column` object:

```
<PhysicalTable>
  <Columns>
    <Column/>
  </Columns>
</PhysicalTable>
```

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: `<Column/>`.

See Also

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

- See “Examples: METADATA Procedure” on page 90 and “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 9.
- *SAS Open Metadata Interface: Reference* provides the following information:
 - which methods to use for common tasks
 - the `DoRequest` method and other methods in the `IOMI` class
 - the `Status` method in the `IServer` class
- *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. 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 you specify the `VERBOSE` argument, the input XML is written to the SAS log. For example output, see Example 7 on page 95.

If you process the output in a subsequent DATA step, and the processing updates an `_INFILE_` buffer variable in place (for example, `_infile_=tranwrd(_infile_, 'old', 'new')`), the resulting XML string might be truncated at 80 characters. To avoid truncation, create a temporary variable instead of the `_INFILE_` buffer variable. Set the length of the temporary variable (for example, with a `LENGTH` statement within the DATA step).

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.

To use the output XML file (for example, to run reports), submit an `XML LIBNAME` statement that assigns an XML map. The `XML LIBNAME` statement imports the XML file to a SAS data set. Like the output XML file, this data set can be temporary or permanent. For more information, see “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 9. For more information about the XML engine and XML maps, see the *SAS XML LIBNAME Engine: User's Guide*.

Examples: METADATA Procedure

Example 1: Change a Repository's State

Procedure features:

IN= argument

A repository's state is computed from both the repository's registered access mode and the metadata server's state. To change the repository's state, submit the UpdateMetadata method with PROC METADATA. The example code re-registers the repository with a different access mode, 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 can be any of the following integers:

Access= Attribute	Computed Repository State
0	online
1	readonly
2	admin
4	offline

Program

Set the Access= attribute for the repository. This example results in full access for the repository.

```
proc metadata
  in=<UpdateMetadata>
    <Metadata>
      <RepositoryBase id='A0000001.A58LN5R2' Access='0' />
    </Metadata>
    <NS>repos</NS>
    <Flags>268435456</Flags>
  </UpdateMetadata>;
run;
```

Example 2: Simple Request for Server Status

Procedure features:

IN= argument

This PROC METADATA example queries the metadata server for its status information. The procedure uses the Status method in the IServer class in the SAS Open Metadata Interface. You can also submit the STATUS action with PROC METAOperate.

Program

Submit the Status method. Always use the <Metadata> element to represent the inMetadata= or inMeta= parameter of a method.

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

SAS Log

NOTE: Response XML:

```
<Status><Metadata><ModelVersion>8.01/<ModelVersion>
<PlatformVersion>9.2.0.0/<PlatformVersion>
<ServerState>ONLINE</ServerState><PauseComment/></Metadata></Status>
```

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

Procedure features:

Connection arguments

IN= argument

OUT= argument

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

Program

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

Procedure features:

IN= argument

Other features:

DATA _NULL_ and DATALINES statements

DATA _NULL_ and PUT statements

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.

Program

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. 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;
```

The following code performs 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;
```

Example 5: HEADER= Argument

Procedure features:

HEADER= argument

This example shows how the HEADER=SIMPLE and HEADER=FULL arguments are used.

Program

Insert a header at the top of the output XML. This code inserts the static header `<?xml version="1.0" ?>` in the output XML file that is identified by the fileref MYOUTPUT.

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

```

        <Options/>
    </GetTypes>";
run;

```

Insert a header at the top of the output XML. This code inserts the header `<?xml version="1.0" encoding="ebcdic1047"?>` in the output XML file that is identified by the fileref MYOUTPUT. `ENCODING= EBCDIC1047` in the `FILENAME` statement indicates the encoding for Western EBCDIC. For a list of encoding values, see the **SAS National Language Support (NLS): Reference Guide**.

```

filename myoutput "u:\out.xml" encoding=ebcdic1047;
proc metadata
    header=full
    out=myoutput
    in="<GetTypes>
        <Types/>
        <Ns>SAS</Ns>
        <Flags/>
        <Options/>
        </GetTypes>";
run;

```

Example 6: VERBOSE Argument

Procedure features:

VERBOSE argument

IN= argument

Program

Issue PROC METADATA with VERBOSE. In this example, PROC METADATA issues a `GetMetadataObjects` method to list all of the objects of type `PhysicalTable` that are defined in the active repository. The active repository identifier is substituted where `$METAREPOSITORY` appears in the XML string.

```

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

```


SAS Log

The VERBOSE argument returns the preprocessed input XML, which includes the repository identifier referenced by \$METAREPOSITORY. The XML output shows two objects: a table that is named INVENTORY, and another table that is named LOCATIONS.

NOTE: Input XML:

```
<GetMetadataObjects>    <Reposid>A0000001.A5K2EL3N</Reposid>
<Type>PhysicalTable</Type>    <Objects/>    <Ns>SAS</Ns>    <Flags/>
<Options/>    </GetMetadataObjects>
```

NOTE: Response XML:

```
<GetMetadataObjects><Reposid>A0000001.A5K2EL3N</Reposid><Type>PhysicalTable</Type>
<Objects><PhysicalTable Id="A58LNR2.AR000001" Name="INVENTORY"><PhysicalTable
Id="A58LNR2.AR00000RT" Name="LOCATIONS"></Objects><Ns>SAS</Ns><Flags/><Options/>
</GetMetadataObjects>
```

Example 7: Request the Metadata for One Object

Procedure features:

IN= argument

Other features:

<Flags> element

Program

Request all properties of a specific table. This code submits a GetMetadata method for a table whose object identifier is A58LN5R2.AR000001. The <Flags> value 1 represents the OMI_ALL flag. For more information about flags, see **SAS Open Metadata Interface: Reference**.

```
proc metadata
  in='<GetMetadata>
    <Metadata>
      <PhysicalTable Id="A58LN5R2.AR000001"/>
    </Metadata>
    <Ns>SAS</Ns>
    <Flags>1</Flags>
    <Options/>
  </GetMetadata>';

run;
```

SAS Log

NOTE: Response XML:

```
<GetMetadata><Metadata><PhysicalTable Id="A58LN5R2.AR000001" ChangeState=""
DBMSType="" Desc="" IsCompressed="0" IsDBMSView="0" IsEncrypted="0" LockedBy=""
MemberType="DATA" MetadataCreated="17Oct2006:20:54:16"
MetadataUpdated="17Oct2006:20:54:17" Name="INVENTORY" NumRows="-1" PublicType=""
SASTableName="INVENTORY" TableName="INVENTORY" UsageVersion="0"><AccessControls/>
<Aggregations/><AnalyticTables/><AssociatedXMLMap/><Changes/><Columns>
<Column Id="A58LN5R2.AS000001" Name="Product" Desc=""/><Column Id="A58LN5R2.AS000002"
Name="Test1" Desc=""/><Column Id="A58LN5R2.AS000003" Name="Test2" Desc=""/>
<Column Id="A58LN5R2.AS000004" Name="Final" Desc=""/></Columns><CustomAssociations/>
<Documents/><Extensions/><ExternalIdentities/><ForeignKeys/><Groups/><Implementors/>
<Indexes/><Keywords/><LocalizedAttributes/><ModelResults/><Notes/>
<PrimaryPropertyGroup/><Prompts/><Properties/><PropertySets/><ReachThruCubes/>
<ReferencedObjects/><ResponsibleParties/><Roles/><SASPasswords/>
<SourceClassifierMaps/><SourceTransformations/><SpecSourceTransformations/>
<SpecTargetTransformations/><TableCollections/><TablePackage>
<SASLibraryId="A58LN5R2.AP000001" Name="blue" Desc="This is in the c:/blue directory
and its SAS libname is MYFILES"/></TablePackage><TargetClassifierMaps/>
<TargetTransformations/><Timestamps/><TrainedModelResults/><Trees/>
<TSObjectNamespace/><UniqueKeys/><UsedByPrototypes/><UsingPrototype/><Variables/>
<XPaths/></PhysicalTable></Metadata></Ns>SAS</Ns><Flags>1</Flags><Options/>
</GetMetadata>
```

Example 8: Request the Metadata for One Type of Object

Procedure features:

IN= argument

Other features:

<XMLSelect> element

Program

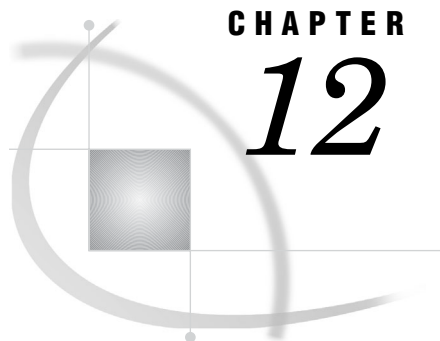
Submit the GetMetadataObjects method. The <XMLSelect> element searches for InformationMap, which is a specific TransformRole of a Transformation object. The metadata server returns metadata for all of the SAS Information Maps that are registered in the specified repository. For details about information maps, see the **Base SAS Guide to Information Maps**. For more information about using the <XMLSelect> element, see **SAS Open Metadata Interface: Reference**.

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

data _null_;
  file myinput;
  input;
  put _infile_ ' ';
  datalines;
<GetMetadataObjects>
<Reposid>A0000001.A57DQR88</Reposid>
<Type>Transformation</Type>
<Objects/>
<NS>SAS</NS>
```

```
<Flags>401</Flags>
<Options>
  <XMLSelect search="Transformation[@TransformRole='InformationMap']"/>
</Options>
</GetMetadataObjects>
;;
run;

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

CHAPTER

12

METALIB Procedure

Overview: <i>METALIB</i> Procedure	99
Syntax: <i>METALIB</i> Procedure	100
<i>PROC METALIB</i> Statement	101
<i>OMR</i> Statement	101
<i>EXCLUDE</i> or <i>SELECT</i> Statement	104
<i>FOLDER</i> or <i>FOLDERID</i> Statement	105
<i>IMPACT_LIMIT</i> Statement	106
<i>NOEXEC</i> Statement	106
<i>PREFIX</i> Statement	107
<i>REPORT</i> Statement	107
<i>UPDATE_RULE</i> Statement	108
Concepts: <i>METALIB</i> Procedure	108
How <i>PROC METALIB</i> Works	108
What Metadata Is Updated?	109
Results: <i>METALIB</i> Procedure with the <i>REPORT</i> Statement	109
Introduction	109
Output Delivery System (ODS) Reports	110
Details in the Report	110
Examples: <i>METALIB</i> Procedure	111
Example 1: Synchronizing Metadata with the Data Source	111
Example 2: Selecting Tables for Processing	112
Example 3: Generating an Impact Analysis	113
Example 4: Adding a Prefix to New Metadata Names	114

Overview: METALIB Procedure

The METALIB procedure supports SAS data sets (data files and data views), DBMS data, and SAS Information Maps. The data source or information map 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. A SAS library is defined in the metadata by a SASLibrary object. A table in a SAS library is defined in the metadata by several objects that are collectively called a *table definition*. All of the table definitions that are associated with a SASLibrary object are tracked in an association list.

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, unique keys, foreign keys, and key associations

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
- 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 an impact analysis for Job and Transformation objects
- 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 is obsolete or duplicated

For more information, see “How PROC METALIB Works” on page 108. For details about information maps, see *Base SAS Guide to Information Maps*.

Syntax: METALIB Procedure

Requirement: The metadata server must be running. The SAS library must already be defined in the metadata by using a product such as SAS Management Console.

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

PROC METALIB;

```

OMR <=> (LIBID = <">identifier<"> | LIBRARY = <">name<"> | LIBRARY = "/"
  folder-pathname/name" | LIBURI = "URI-format"
  <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>);>

```

Task	Statement
Update metadata in the metadata server to match the tables in a library	“PROC METALIB Statement” on page 101
Specify the data source and connection parameters for the SAS Metadata Server	“OMR Statement” on page 101
Exclude or select a table, or a list of tables, for processing	“EXCLUDE or SELECT Statement” on page 104

Task	Statement
Specify where new metadata is stored in SAS folders	“FOLDER or FOLDERID Statement” on page 105
Specify the maximum number of Job or Transformation objects that can be affected by updates to table definitions	“IMPACT_LIMIT Statement” on page 106
Suppress the metadata changes from being made	“NOEXEC Statement” on page 106
Specify a text string to add to the beginning of all new metadata object names	“PREFIX Statement” on page 107
Create a report that summarizes metadata changes	“REPORT Statement” on page 107
Override default update behavior	“UPDATE_RULE Statement” on page 108

PROC METALIB Statement

PROC METALIB;

Task	Statement
Update metadata to match the data source	PROC METALIB

OMR Statement

Specifies the data source and connection parameters for the SAS Metadata Server.

OMR <=> (LIBID=<">identifier<"> | LIBRARY=<">name<"> | LIBRARY="/
folder-pathname/name" | LIBURI="*URI-format*"
 <*server-connection-arguments*>);

Required Argument

LIBID=<">identifier<"> | **LIBRARY**=<">name<"> | **LIBRARY**="/*folder-pathname/*
name" | **LIBURI**="*URI-format*"

specifies a SASLibrary object, which defines a SAS library. This SAS library contains the tables whose metadata is updated.

Restriction: 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 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 will 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.

LIBID=<">identifier<">

specifies the 8-character metadata identifier of the SASLibrary object that represents the library. The 8-character identifier is the second half of the 17-character identifier. For more information, see Chapter 2, “Metadata Object Identifiers and URIs,” on page 5.

LIBRARY=<">name<">

specifies the value in the SASLibrary object’s Name= attribute. If you have more than one library with the same name, but they are stored in different SAS folders, then you must use the *folder-pathname* syntax.

LIBRARY="/folder-pathname/name"

specifies the folder pathname and the value in the SASLibrary object’s Name= attribute. The pathname is the object’s location in a SAS folder. The pathname begins with a forward slash. An example is **library="/Users/Dmitri/My Folder/test/mylib"**. For more information about how PROC METALIB uses folders, see “FOLDER or FOLDERID Statement” on page 105.

LIBURI="URI-format"

specifies a URI, which is a standard from SAS Open Metadata Architecture. For more information, see Chapter 2, “Metadata Object Identifiers and URIs,” on page 5. The following URI formats are supported:

LIBURI="identifier.identifier"

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

LIBURI="SASLibrary/identifier.identifier"

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

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

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

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

Note: SAS Data Integration Studio can process Work tables that exist temporarily in the Work library. The metadata type is WorkTable. Usually, Work tables are not assigned to a library and have no library metadata, but they do have table and column metadata. A Work table that results from a generated transformation can be dynamic in nature. In other words, its structure might be modified by the transformation. PROC METALIB can be used to 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="A507HLNB");
  select ("A507HLNB.A9000001");
run;
△
```

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 24.

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 33. The maximum length is 256 characters.

Alias: METAPASS=
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 34. The default for the METAPORT= system option is 8561. The range is 1–65535.

Alias: METAPORT=

Requirement: The value must be enclosed 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 37. In this release, the only supported value is BRIDGE, which specifies the SAS Bridge protocol.

Alias: METAPROTOCOL=

Requirement: Do not enclose the value in quotation marks.

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 38. The default for the METAREPOSITORY= system option is **Foundation**.

REPID=<">identifier<">

specifies an 8-character identifier. This identifier is the first half of the SASLibrary’s 17-character identifier, and is the second half of the repository’s identifier. For more information, see Chapter 2, “Metadata Object Identifiers and URIs,” on page 5.

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 39. The maximum length is 256 characters.

Alias: **HOST=**

IPADDR=

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, and 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 41. The maximum length is 256 characters.

Alias: **ID=**

METAUSER=

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 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.

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

Arguments

table-specification

<">*table-name*<">

is the SAS name of a PhysicalTable 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 Chapter 2, “Metadata Object Identifiers and URIs,” on page 5. 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 101. Δ

read-password

is the READ password, if any, that was previously assigned to the table. For information about file protection, see *SAS Language Reference: Dictionary*. The following example specifies a READ password for tab1:

```
select ("tab1" read=myspw "tab2" "tab3" "Table4");
```

FOLDER or FOLDERID Statement

Specifies where new metadata is stored in SAS folders.

FOLDER `<=>` `"/pathname"` | **FOLDERID** `<=>` `"identifier.identifier"`;

Arguments

When you specify FOLDER= or FOLDERID=, you add or update the table definition in the specified SAS folder. The SASLibrary object remains in its original folder. Column, ForeignKey, Index, KeyAssociation, and UniqueKey objects are added or updated in the same folder as the specified PhysicalTable object.

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 the 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 will be in different SAS folder locations. Every table definition has a unique metadata identifier.

If you do not specify a folder or if the folder specification is not valid, 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 or if the folder specification is not valid, and if a table is new, PROC METALIB adds the new table definition to the SASLibrary object's folder.

FOLDER `<=>` `"/pathname"`

is a folder pathname in a SAS folder. The pathname begins with a forward slash. Here is an example:

```
folder="/Users/MyUserID/My Folder/Test";
```

FOLDERID $\langle = \rangle$ "*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.

IMPACT_LIMIT Statement

Specifies the maximum number of Job or Transformation objects that can be affected by updates to table definitions.

See also: Example 3 on page 113

IMPACT_LIMIT=*n*;

Arguments

n

maximum number (an integer) of Job or Transformation objects that can be affected by changes to a table. For each table that is analyzed, if the specified number is exceeded, the table's metadata is not added, updated, or deleted. You can specify IMPACT_LIMIT with TYPE=DETAIL in the REPORT statement to generate an impact analysis. The default is no impact limit and no impact analysis.

With this argument, the procedure identifies potential impact only. It does not verify that a Job or Transformation object will be affected, only that it could be affected.

The procedure identifies only the Job or Transformation objects that can be directly affected by the changes. These objects can contain many other objects that could also be affected 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.

NOEXEC;

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 also: Example 4 on page 114

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

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.

See also: “Results: METALIB Procedure with the REPORT Statement” on page 109

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

Arguments

TYPE=DETAIL | SUMMARY

DETAIL

specifies that the report includes all of the information generated by TYPE=SUMMARY, and a list of Job and Transformation objects that are directly related to the table that is being processed. To get the list of Job and Transformation objects (also known as an *impact analysis*), you must specify IMPACT_LIMIT (see “IMPACT_LIMIT Statement” on page 106) . If you do not specify IMPACT_LIMIT, the report defaults to the TYPE=SUMMARY report.

SUMMARY

specifies that the report includes information about any metadata changes that were (or would be) made to the table that is being processed. This is the default.

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.

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.

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

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 tables that have 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 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 the column name in the data source but 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.

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

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. △

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, ForeignKey, Index, KeyAssociation, and UniqueKey. 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 additionally 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 due to large impact, and a list of 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 log and output 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 107.

Output Delivery System (ODS) Reports

The default report destination is a SAS output listing. In addition, you can choose to format the report with an Output Delivery System (ODS) destination, such as HTML or RTF. ODS reports are easier to read than the SAS output listing. To omit the SAS output listing, submit the ODS LISTING CLOSE statement.

For example code that produces ODS output, see “Examples: METALIB Procedure” on page 111.

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 an attribute name (which

indicates that the attribute's value was modified). It can also be a "Column" or a "Column Order" message, followed by the name of the column that was affected by 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: Synchronizing Metadata with the Data Source

Procedure features:

- OMR statement with LIBID value
- Default connection properties
- UPDATE_RULE statement with DELETE argument
- REPORT statement with MATCHING argument

Other features:

- ODS
-

This example adds, updates, and deletes metadata to match the physical tables in a SAS library.

Program

Submit an ODS HTML statement. ODS sends the output to an HTML file. In addition, the ODS LISTING CLOSE statement suppresses the default listing destination.

```
ods listing close;
ods html body="c:\test\updatereport.html";

proc metalib;
```

Specify the data source and connect to the metadata server. Specify a SAS library that is already defined in the metadata. Because this example does not specify connection arguments for the metadata server, the procedure uses the values of the METAPASS, METAPORT, METAREPOSITORY, METASERVER, and METAUER system options.

```
omr (libid="A58LN5R2.AP000001" );
```

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.

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

Close the HTML destination. Reopen the listing destination.

```
ods html close;
ods listing;
```

SAS Output: HTML

Metadata Summary Statistics							
Total tables analyzed	10						
Tables Updated	3						
Tables Deleted	1						
Tables Added	2						
Tables matching data source	4						
Tables not processed	0						

Tables Updated							
Table			Updates				
Metadata Name	Metadata ID	SAS Name	Metadata Name	Metadata ID	SAS Name	Metadata Type	Change
CENSUS	A52UFLFW.BB00000N	CENSUS	AdjRate	A52UFLFW.BC00003O	AdjRate	Column	Added
CENSUS1980	A52UFLFW.BB00000T	CENSUS1980	State	A52UFLFW.BC00003M	State	Column	IsDiscrete
			State	A52UFLFW.B0000002	State	Index	Added
			CENSUS1980.ic_id	A52UFLFW.B0000001	ic_id	UniqueKey	Added
ENERGY	A52UFLFW.BB00000Q	ENERGY	State	A52UFLFW.B0000003	State	Index	Added

Tables Deleted		
Metadata Name	Metadata ID	SAS Name
GROC	A52UFLFW.BB00000R	GROC

Tables Added		
Metadata Name	Metadata ID	SAS Name
CENSUSSUB	A52UFLFW.BB00000U	CENSUSSUB
GROCERY	A52UFLFW.BB00000V	GROCERY

Tables matching data source		
Metadata Name	Metadata ID	SAS Name
AUTO	A52UFLFW.BB00000M	AUTO
CENSUS1990	A52UFLFW.BB00000O	CENSUS1990

Example 2: Selecting Tables for Processing

Procedure features:
SELECT statement

Program

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='test'");
  select (mytable);
  report;
run;
```

Program

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.

Specify a table ID. 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='test'");
  select (A7892350.B00265DX);
  report;
run;
```

Example 3: Generating an Impact Analysis

Procedure features:

- IMPACT_LIMIT statement
- REPORT(TYPE=DETAIL) statement
- NOEXEC statement

To generate an impact analysis, specify IMPACT_LIMIT and REPORT(TYPE=DETAIL). Use the NOEXEC statement if you want to examine the changes before you commit to making the changes. The generated impact analysis shows how the data source differs from metadata, and how making those changes will affect the Job and Transformation objects.

Program

Submit the procedure. The impact limit is set to zero, so any impact on Job or Transformation objects results in a "limit exceeded" entry in the output.

```
ods listing close;
ods html body="C:\test\update.html";
proc metalib;
  omr (library="mydifiles");
  update_rule(delete);
```

```

report (type=detail);
impact_limit=0;
noexec;
run;
ods html close;
ods listing;

```

SAS Output: HTML

The SAS System

The METALIB Procedure

Detail Report of Potential Changes for Library mydifiles

Repository Foundation

03SEP2008

Metadata Summary Statistics	
Total tables analyzed	8
Tables not processed due to large impact	2
Tables to be Updated	0
Tables to be Deleted	0
Tables to be Added	0
Tables matching data source	6
Tables not processed	0

Tables Exceeding Impact Limit of 0			
Metadata Name	Metadata ID	SAS Name	Total Impact
AUTOSORT	A56KQRB0.B600000H	AUTOSORT	1
AUTO	A56KQRB0.B600000A	AUTO	1

Potential-Impact Analysis						
Table			Impact			
Metadata Name	Metadata ID	Action	Metadata Name	Metadata ID	Metadata Type	Transformation Role
AUTOSORT	A56KQRB0.B600000H	Limit Exceeded	My Test Job	A56KQRB0.BR000001	Job	
AUTO	A56KQRB0.B600000A	Limit Exceeded	My Test Job	A56KQRB0.BR000001	Job	

Example 4: Adding a Prefix to New Metadata Names

Procedure features:
 PREFIX= statement

To add a prefix to the names of new metadata objects during an update, specify the PREFIX statement.

Program

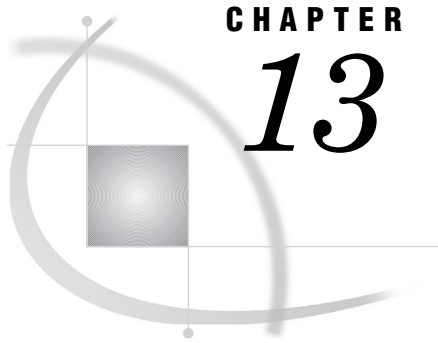
In this example, the user runs an update on November 30 and wants to add that date to any new metadata. A new table has been added to the SAS library. In the data source, the table is named **Q3Sales**. In the metadata, the table definition is named **November30Q3Sales**.

Submit the PREFIX statement with PROC METALIB. If any new metadata objects are defined, the metadata name (the Name= attribute) begins with the specified prefix.

```
proc metalib;  
  omr (library="mylibrary");  
  prefix="November30";  
  report;  
run;
```

SAS Output: Listing

Metadata Name	Metadata ID	SAS Name
November30Q3sales	A58LN5R2.AR00066A	Q3sales



CHAPTER

13

METAOPERATE Procedure

<i>Overview: METAOPERATE Procedure</i>	117
<i>Syntax: METAOPERATE Procedure</i>	117
<i>PROC METAOPERATE Statement</i>	118
<i>Concepts: METAOPERATE Procedure</i>	123
<i>How PROC METAOPERATE Works</i>	123
<i>How PAUSE, REFRESH, and RESUME Affect Repositories</i>	123
<i>Examples: METAOPERATE Procedure</i>	124
<i>Example 1: Submitting ACTION=STATUS</i>	124
<i>Example 2: Submitting ACTION=PAUSE with a Pause Comment</i>	125
<i>Example 3: Submitting ACTION=REFRESH with ARM Logging</i>	126
<i>Example 4: Submitting ACTION=REFRESH with Journaling</i>	126
<i>Example 5: Submitting ACTION=REFRESH to Pause and Resume the Metadata Server</i>	126
<i>Example 6: Submitting ACTION=RESUME</i>	127
<i>Example 7: Submitting ACTION=EMPTY</i>	127

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

The METADATA procedure and the metadata DATA step functions perform some of the same tasks as PROC METAOPERATE. For more information, see “Comparison of the METADATA Procedure and the METAOPERATE Procedure” on page 81.

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>;

```

Task	Statement
Perform administrative tasks associated with the metadata server	"PROC METAOPERATE Statement" on page 118

PROC METAOPERATE Statement

PROC METAOPERATE <server-connection-arguments>

```

ACTION = PAUSE | REFRESH | RESUME | DELETE | EMPTY | UNREGISTER
        | STATUS | STOP
<NOAUTOPAUSE>
<OPTIONS = "XML-string">
<OUT = SAS-data-set>;

```

Task	Argument
Connect to the metadata server	<i>server-connection-arguments</i>
Specify an action for the metadata server	ACTION=
Omit the automatic pause and resume when you pass an action to the metadata server	NOAUTOPAUSE
Submit an XML string to the metadata server	OPTIONS=
Create an output data set	OUT=

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 24.

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 33. The maximum length is 512 characters.

Alias: METAPASS=
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 34. The default for the METAPORT= system option is 8561. The range is 1–65535.

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 37. In this release, the only supported value is BRIDGE, which specifies the SAS Bridge protocol.

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 38. The default for the METAREPOSITORY= system option is **Foundation**. The maximum length is 32,000 characters.

Alias: METAREPOSITORY=
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 39. The maximum length is 256 characters.

Alias: HOST=
IPADDR=
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 41. The maximum length is 256 characters.

Alias: ID=
METAUSER=
USERID=

Action Arguments

ACTION=

specifies the action that you want to perform. ACTION is a required argument.

Requirement: You must have the appropriate SAS Administrator role on the metadata server to execute all actions except STATUS. For more information, see the *SAS Intelligence Platform: System Administration Guide*.

Tip: Specifying more than one XML element in a PROC METAOPERATE statement might cause undesired results. Use more than one XML element only when specified in the documentation.

Tip: If you use PROC METAOPERATE to delete, empty, or unregister a *project* repository, you must first issue an UndoCheckoutMetadata method call with the METADATA procedure to unlock any checked-out objects. Alternatively, 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.

DELETE

removes the specified repository, and removes the repository's registration from the repository manager. 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. 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 7 on page 127.

PAUSE

limits the availability of the metadata server by setting the metadata server's state to admin or offline. Beginning in SAS 9.2, 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 123.

Issue ACTION=PAUSE and use the OPTIONS= argument to specify a <Server STATE=ADMIN/> or a <Server STATE=OFFLINE/> 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 to an offline state.

The <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 on page 125.

Here are some examples of situations that can require you to pause the metadata server to an admin or offline state:

- ☐ to close repositories while you perform a system backup
- ☐ to troubleshoot system errors
- ☐ to install or upgrade software

REFRESH

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

- With the `<Server/>` XML element and no other XML elements specified, the **REFRESH** action pauses and resumes (in a single step) the metadata server. Do not specify the **STATE=** parameter. The **REFRESH** action recovers memory on the metadata server and reloads authorization inheritance rules. For more information, see Example 5 on page 126. After the refresh, all repositories return to the same pause state they were in before the refresh. For more information, see “How **PAUSE**, **REFRESH**, and **RESUME** Affect Repositories” on page 123.
- With the `<ARM/>` XML element specified, the **REFRESH** action enables or disables ARM logging and specifies a pathname for the ARM log. The `<Server/>` XML element is required. For more information, see Example 3 on page 126.
- With the `<OMA/>` XML element and the **JOURNALPATH=** parameter specified, the **REFRESH** action specifies a new filename for metadata server journaling. The `<Server/>` XML element is required. For more information, see Example 4 on page 126.

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 123 and Example 6 on page 127.

Any text that was specified in the `<PauseComment/>` XML element during the **PAUSE** action is cleared.

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 on page 124.

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. 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.

NOAUTOPAUSE

omits the automatic pause and resume of the metadata server when **PROC METAOPERATE** passes an action to the metadata server. **NOAUTOPAUSE** is required for the **DELETE**, **EMPTY**, and **UNREGISTER** actions.

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" "/>"
```

Δ

The XML strings include the following:

<ARM parameter-name="parameter-value"/>

is one or more <ARM/> XML elements that specify system options to enable or disable ARM logging. The <Server/> XML element is required. REFRESH is the most appropriate action to specify the <ARM/> XML element, but the PAUSE and RESUME actions can also specify it. If the metadata server is refreshed, or stopped and started, the ARM parameters return to the values in the configuration file. For more information, see Example 3 on page 126 and the *SAS Intelligence Platform: System Administration Guide*, as well as the ARMSUBSYS= and ARMLOC= system options in *SAS Language Reference: Dictionary*. 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.

<OMA JOURNALPATH="filename"/>

when submitted with the REFRESH action, stops writing journal entries to the metadata server journal file in the current location, and resumes writing in a new journal file in the specified physical location. The <Server/> XML element is required. Journaling must be enabled on the metadata server before you submit this parameter. If the metadata server is refreshed, or stopped and started, this parameter returns to the value in the configuration file. For more information, see Example 4 on page 126 and the *SAS Intelligence Platform: System Administration Guide*.

<PauseComment>text</PauseComment>

when submitted 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 on page 125. When you submit the RESUME action, you clear the text in <PauseComment>.

<Server STATE="ADMIN | OFFLINE"/>

specifies that the action applies to the metadata server. The <Server/> XML element has the following uses:

- ☐ It is required for the REFRESH action, with no parameters. For more information, see Example 5 on page 126.
- ☐ It is optional for the PAUSE action, and its STATE= parameter is optional. With the PAUSE action, if you do not specify <Server/> 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.

For the PAUSE action, 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.

OUT=SAS-data-set

names the output data set. This argument is used with the STATUS action. Other actions do not create output.

Concepts: METAOPERATE Procedure

How PROC METAOPERATE Works

The administrator of the metadata server can perform two kinds of maintenance with PROC METAOPERATE:

- control the metadata server by calling methods in the IServer class of SAS Open Metadata Architecture: PAUSE, REFRESH, RESUME, STATUS, STOP
- control a repository by calling methods in the IOMI class of SAS Open Metadata Architecture: DELETE, EMPTY, UNREGISTER

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. Alternatively, 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 Chapter 11, "METADATA Procedure," on page 83.
- 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*, especially the large section about backing up and restoring.

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 OMS_READONLY access mode and an ADMIN server state has an admin(readonly) pause state.

	Online Server State	Admin Server State	Offline Server State
OMS_FULL Access Mode	online	admin	offline
OMS_READONLY Access Mode	readonly	admin(readonly)	offline
OMS_ADMIN Access Mode	admin	admin	offline
OMS_OFFLINE Access Mode	offline	offline	offline

Examples: METAOPERATE Procedure

Example 1: Submitting ACTION=STATUS

Procedure features:

Connection arguments
ACTION=STATUS argument

These examples request the status of the metadata server and show the arguments that connect to the metadata server.

Program

Specify connection arguments and query the metadata server for its status. This example specifies all connection arguments for the metadata server.

```
proc metaoperate
  server="a123.us.company.com"
  port=8561
```

```

userid="myuserid"
password="mypassword"
action=status;
run;

```

Specify connection arguments and query the metadata server for its status.

LOCALHOST specifies the metadata server that is running on the same host as the SAS session.

```

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

```

SAS Log

```

NOTE: Server al23.us.company.com SAS Version is 9.02.02B0P012308.
NOTE: Server al23.us.company.com SAS Long Version is 9.02.02B0P01232008.
NOTE: Server al23.us.company.com Operating System is XP_PRO.
NOTE: Server al23.us.company.com Operating System Family is WIN
NOTE: Server al23.us.company.com Operating System Version is Service Pack 2.
NOTE: Server al23.us.company.com Client is janedoe.
NOTE: Server al23.us.company.com Metadata Model is Version 11.02.
NOTE: Server al23.us.company.com is RUNNING on 11Aug08:15:54:15.

```

Example 2: Submitting ACTION=PAUSE with a Pause Comment

Procedure features:

ACTION=PAUSE argument

OPTIONS= argument with <PauseComment>

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

Program

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

Procedure features:

ACTION=REFRESH argument

OPTIONS= argument with <ARM/>

Program

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""/>
           <Server/>" ;
run;
```

Example 4: Submitting ACTION=REFRESH with Journaling

Procedure features:

ACTION=REFRESH argument

OPTIONS= argument with <OMA/>

Program

Change the pathname. The JOURNALPATH= value specifies a pathname for metadata server journaling.

```
proc metaoperate
    action=refresh
    options="<OMA JOURNALPATH='c:/MyServers/journal/test.dat' />
           <Server/>";
run;
```

Example 5: Submitting ACTION=REFRESH to Pause and Resume the Metadata Server

Procedure features:

ACTION=REFRESH argument

OPTIONS= argument with <Server/> only

Program

Refresh the metadata server. To recover memory and reload inheritance rules, submit the REFRESH action with the <Server/> element.

```
proc metaoperate
    action=refresh
    options="<Server/>";
run;
```

Example 6: Submitting ACTION=RESUME

Procedure features:

ACTION=RESUME argument

Program

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 7: Submitting ACTION=EMPTY

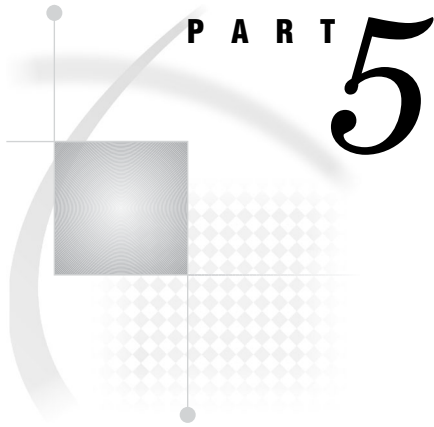
Procedure features:

ACTION=EMPTY argument

Program

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;
```

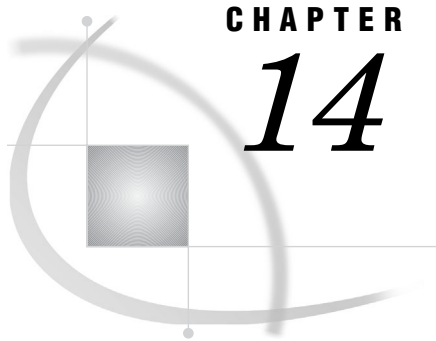
DATA Step Functions

Chapter 14.....**Introduction to DATA Step Functions for Metadata** 131

Chapter 15.....**DATA Step Functions for Reading and Writing Metadata** 133

Chapter 16.....**DATA Step Functions for Metadata Security
Administration** 169

Appendix 1.....**Recommended Reading** 207



Introduction to DATA Step Functions for Metadata

Overview of DATA Step Functions for Metadata 131

Best Practices 131

Array Parameters 132

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:

- Chapter 15, “DATA Step Functions for Reading and Writing Metadata,” on page 133
- Chapter 16, “DATA Step Functions for Metadata Security Administration,” on page 169

Before you can use the metadata DATA step functions, you must issue the metadata system options to establish a connection with the metadata server. For more information, see “Connection Options” on page 24.

For help in forming your DATA step, see the following references:

- For information about metadata objects, see *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 Language Reference: Dictionary*.
- 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 *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.

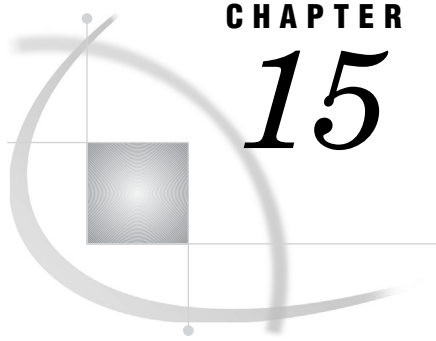
For best performance, always resolve your URI into an ID instance. For example, if you make several function calls on the object “`OMSOBJ:LogicalServer?@Name='foo'`”, first use the `METADATA_RESOLVE` or `METADATA_GETNOBJ` function to convert the object to “`OMSOBJ:LogicalServer\A57DQR88.AU000003`”. URIs 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 Metadata Security Administration” on page 196.
- 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

DATA Step Functions for Reading and Writing Metadata

<i>Introduction to DATA Step Functions for Reading and Writing Metadata</i>	135
<i>What Are the DATA Step Functions for Reading and Writing Metadata?</i>	135
<i>Referencing a Metadata Object with a URI</i>	136
<i>Comparison of DATA Step Functions to Metadata Procedures</i>	137
<i>METADATA_DELASSN Function</i>	137
Syntax	137
Arguments	137
Return Values	137
Example	138
Related Functions	139
<i>METADATA_DELOBJ Function</i>	139
Syntax	139
Arguments	139
Return Values	140
Example	140
Related Functions	140
<i>METADATA_GETATTR Function</i>	140
Syntax	140
Arguments	141
Return Values	141
Example	141
Related Functions	142
<i>METADATA_GETNASL Function</i>	142
Syntax	142
Arguments	142
Return Values	142
Example	143
Related Functions	143
<i>METADATA_GETNASN Function</i>	143
Syntax	143
Arguments	144
Return Values	144
Example	144
<i>METADATA_GETNATR Function</i>	145
Syntax	145
Arguments	145
Return Values	145
Examples	146
Example 1: Using an Object URI	146
Example 2: Using a Repository URI	146
Related Functions	147

<i>METADATA_GETNOBJ Function</i>	147
Syntax	147
Arguments	147
Return Values	148
Examples	148
Example 1 : Determining How Many Machine Objects Exist	148
Example 2 : Looping Through Each Repository on a Metadata Server	149
Related Functions	149
<i>METADATA_GETNPRP Function</i>	149
Syntax	149
Arguments	149
Return Values	150
Example	150
Related Functions	151
<i>METADATA_GETNTYP Function</i>	151
Syntax	151
Arguments	151
Return Values	151
Example	151
<i>METADATA_GETPROP Function</i>	152
Syntax	152
Arguments	152
Return Values	152
Example	153
Related Functions	153
<i>METADATA_NEWOBJ Function</i>	153
Syntax	153
Arguments	153
Return Values	154
Details	154
Example	154
Related Functions	155
<i>METADATA_PATHOBJ Function</i>	155
Syntax	155
Arguments	155
Return Values	156
Examples	156
Example 1: The Path Contains a (deftype) Suffix	156
Example 2: The defeype Is Passed as a Separate Parameter	157
<i>METADATA_PAUSED Function</i>	157
Syntax	158
Return Values	158
Example	158
<i>METADATA_PURGE Function</i>	158
Syntax	158
Arguments	159
Return Values	159
Details	159
Example	159
<i>METADATA_RESOLVE Function</i>	160
Syntax	160
Arguments	160
Return Values	160
Examples	161

<i>Example 1: Using an Object URI</i>	161
<i>Example 2: Using a Repository URI</i>	161
<i>METADATA_SETASSN Function</i>	162
<i>Syntax</i>	162
<i>Arguments</i>	162
<i>Return Values</i>	164
<i>Example</i>	164
<i>Related Functions</i>	165
<i>METADATA_SETATTR Function</i>	165
<i>Syntax</i>	165
<i>Arguments</i>	165
<i>Return Values</i>	165
<i>Example</i>	165
<i>Related Functions</i>	166
<i>METADATA_SETPROP Function</i>	166
<i>Syntax</i>	166
<i>Arguments</i>	166
<i>Return Values</i>	166
<i>Example</i>	167
<i>Related Functions</i>	167
<i>METADATA_VERSION Function</i>	167
<i>Syntax</i>	167
<i>Return Values</i>	167
<i>Example</i>	168

Introduction to 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.

Name	Description
METADATA_DELASSN	Deletes all objects that make up the specified association
METADATA_DELOBJ	Deletes the first object that matches the specified URI
METADATA_GETATTR	Returns the value of the specified attribute for specified object
METADATA_GETNASL	Returns the <i>n</i> th association of the specified object
METADATA_GETNASN	Returns the <i>n</i> th associated object of the specified association
METADATA_GETNATR	Returns the <i>n</i> th attribute on the object specified by the URI
METADATA_GETNOBJ	Returns the <i>n</i> th object that matches the specified URI
METADATA_GETNPRP	Returns the <i>n</i> th property of the specified object

Name	Description
METADATA_GETNTYP	Returns the <i>n</i> th object type on the metadata server
METADATA_GETPROP	Returns the specified property of the specified object
METADATA_NEWOBJ	Creates a new metadata object
METADATA_PATHOBJ	Returns the Id and Type attributes of the specified folder object
METADATA_PAUSED	Determines whether the metadata server is paused
METADATA_PURGE	Purges the specified URI
METADATA_RESOLVE	Resolves a URI into an object on the metadata server
METADATA_SETASSN	Modifies an association list for an object
METADATA_SETATTR	Sets the specified attribute for the specified object
METADATA_SETPROP	Sets the specified property for the specified object
METADATA_VERSION	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 Chapter 2, “Metadata Object Identifiers and URIs,” on page 5. 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 Language Reference: Dictionary*.

Comparison of DATA Step Functions to Metadata Procedures

The METAOPERATE procedure and the METADATA procedure perform some of the same tasks as the DATA step functions for reading and writing metadata. These language elements can query metadata for reports, or make changes to specified objects.

PROC METAOPERATE has much simpler syntax and is easy to use. However, it supports a narrow range of tasks.

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 kind of ODS reports as you can with PROC METADATA. Instead of writing the output to an XML data set, you execute the DATA step within a macro, and assign macro variables to the returned values. Those variables become the columns in an output data set. See how these two examples create similar reports from their output: “Example: Creating a Report with the DATA Step” on page 14 and “Example: Creating a Report with the METADATA Procedure and the XML Engine” on page 9.

METADATA_DELASSN Function

Deletes all objects that make up the specified association

Syntax

```
rc = METADATA_DELASSN(uri,asn);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>asn</i>	in	Association name

Return Values

Value	Description
0	Successful completion
-1	Unable to connect to the metadata server

Value	Description
-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;

```

Related Functions

- “METADATA_SETASSN Function” on page 162
- “METADATA_GETNASN Function” on page 143

METADATA_DELOBJ Function

Deletes the first object that matches the specified URI.

Syntax

```
rc = METADATA_DELOBJ(uri);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier

Return Values

Value	Description
0	Successful completion
-1	Unable to connect to the metadata server
-2	The deletion was unsuccessful; see the SAS log for details
-3	No objects match the 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;
```

Related Functions

- “METADATA_DELASSN Function” on page 137
 - “METADATA_GETNOBJ Function” on page 147
 - “METADATA_GETNTYP Function” on page 151
 - “METADATA_NEWOBJ Function” on page 153
-

METADATA_GETATTR Function

Returns the value of the specified attribute for the specified object.

Syntax

```
rc = METADATA_GETATTR(uri, attr, value);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>attr</i>	in	Attribute of the metadata object
<i>value</i>	out	Value of the specified attribute

Return Values

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

```

Related Functions

- “METADATA_GETNATR Function” on page 145
- “METADATA_SETATTR Function” on page 165

METADATA_GETNASL Function

Returns the *n*th association for the specified object.

Syntax

```
rc = METADATA_GETNASL(uri, n, asn);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>n</i>	in	Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 132
<i>asn</i>	out	Association name

Return Values

Value	Description
n	The number of objects that match the URI
-1	Unable to connect to the metadata server
-3	No objects match the 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;

```

Related Functions

- “METADATA_GETNASN Function” on page 143
- “METADATA_SETASSN Function” on page 162

METADATA_GETNASN Function

Returns the n th associated object of the specified association.

Syntax

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

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>asn</i>	in	Association name
<i>n</i>	in	Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 132
<i>nuri</i>	out	URI of the <i>n</i> th associated object

Return Values

Value	Description
<i>n</i>	The number of associated objects
-1	Unable to connect to the metadata server
-3	No objects match the URI
-4	<i>n</i> 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);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>n</i>	in	Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 132
<i>attr</i>	out	Attribute of the metadata object
<i>value</i>	out	Value of the specified attribute

Return Values

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

Related Functions

- “METADATA_GETATTR Function” on page 140
- “METADATA_SETATTR Function” on page 165

METADATA_GETNOBJ Function

Returns the *n*th object that matches the specified URI.

Syntax

```
rc = METADATA_GETNOBJ(uri, n, nuri);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>n</i>	in	Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 132
<i>nuri</i>	out	URI of the <i>n</i> th object that matches the input URI or matches a subtype object of the input URI

Return Values

Value	Description
<i>n</i>	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	<i>n</i> 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;

```

Related Functions

- “METADATA_DELOBJ Function” on page 139
- “METADATA_NEWOBJ Function” on page 153

METADATA_GETNPRP Function

Returns the *n*th property of the specified object.

Syntax

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

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>n</i>	in	Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 132

Argument	Direction	Description
<i>prop</i>	out	Abstract property string
<i>value</i>	out	Value of the specified property string

Return Values

Value	Description
<i>n</i>	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	<i>n</i> 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;
```

Related Functions

- “METADATA_GETPROP Function” on page 152
- “METADATA_SETPROP Function” on page 166

METADATA_GETNTYP Function

Returns the n th object type on the server.

Syntax

```
rc = METADATA_GETNTYP( $n$ ,  $type$ );
```

Arguments

Argument	Direction	Description
n	in	Numeric index value that indicates which row to return from the array; see “Array Parameters” on page 132
$type$	out	Metadata type

Return Values

Value	Description
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);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>prop</i>	in	Abstract property string
<i>value</i>	out	Value of the specified property string
<i>propuri</i>	out	URI of the property object that is associated with the input URI

Return Values

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

Related Functions

- “METADATA_GETNPRP Function” on page 149
- “METADATA_SETPROP Function” on page 166

METADATA_NEWOBJ Function

Creates a new metadata object.

Syntax

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

Arguments

Argument	Direction	Description
<i>type</i>	in	Metadata type
<i>uri</i>	out	Uniform Resource Identifier
<i>name</i>	in	Name attribute for the new metadata object

Argument	Direction	Description
<i>repos</i>	in	Repository identifier of an existing repository; by default, the new object is created in the default repository
<i>parent</i>	out	Parent of the new metadata object
<i>asn</i>	in	Association name

Return Values

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

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 *SAS Metadata Model: Reference*.

Example

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

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

  /* Create a PhysicalTable object. */
```

```

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

put uri=;

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

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

put curi=;

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

put curi=;

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

put curi=;
run;

```

Related Functions

- “METADATA_DELOBJ Function” on page 139
- “METADATA_GETNOBJ Function” on page 147

METADATA_PATHOBJ Function

Returns the Id and Type attributes of the specified folder object.

Syntax

```
rc = METADATA_PATHOBJ(proj, path, deftype, type, ID);
```

Arguments

Argument	Direction	Description
<i>proj</i>	in	Not currently used. Set this argument to null by submitting an empty string “ ”
<i>path</i>	in	Pathname of the object in SAS folders. Pathname begins with a forward slash. Can include the deftype in parentheses as a suffix.
<i>deftype</i>	in	Optional public object type. Can be omitted if deftype is specified as a suffix in the path argument. The deftype is not the same as the type. For example, If you submit a deftype of StoredProcess, the returned type will be ClassifierMap. For more information, see <i>SAS Metadata Model: Reference</i> .
<i>type</i>	out	Metadata object type of the returned ID
<i>ID</i>	out	Unique identifier for the object

Return Values

Value	Description
<i>n</i>	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: The Path Contains a (deftype) Suffix

```
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: The deftype Is Passed as a Separate Parameter

```

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

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

Example

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


Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier; if no argument is specified, the entire connection is purged from the cache

Return Values

Value	Description
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);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>type</i>	out	Metadata type
<i>ID</i>	out	Unique identifier for the first object (or subtype object) that matches the input URI

Return Values

Value	Description
<i>n</i>	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_getnattr("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>);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>asn</i>	in	Association name

Argument	Direction	Description
<i>mod</i>	in	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
<i>auri-1<,...auri-n></i>	in	List of the URIs of the associated objects; see “Array Parameters” on page 132.

*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*.

Return Values

Number of objects matching the input URI.

Value	Description
0	Successful completion
-1	Unable to connect to the metadata server
-3	No objects match the input URI
-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_setasn("omsobj:Machine?@Name='bluedog'",
                    "Notes",
                    "Append",
                    uri);

  put rc=;

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

  put rc=;

run;
```

Related Functions

- “METADATA_DELASSN Function” on page 137
- “METADATA_GETNASN Function” on page 143

METADATA_SETATTR Function

Sets the specified attribute for the specified object.

Syntax

```
rc = METADATA_SETATTR(uri, attr, value);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>attr</i>	in	Attribute of the metadata object
<i>value</i>	in	Value of the specified attribute

Return Values

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

```

Related Functions

- “METADATA_GETATTR Function” on page 140
- “METADATA_GETNATR Function” on page 145

METADATA_SETPROP Function

Sets the specified property for the specified object.

Syntax

```
rc = METADATA_SETPROP(uri, prop, value, propuri);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Uniform Resource Identifier
<i>prop</i>	in	Abstract property string
<i>value</i>	in	Value of the specified property string
<i>propuri</i>	out	URI of the property object that is associated with the input URI

Return Values

Value	Description
1	New property was created and set
0	Existing property was successfully set
-1	Unable to connect to the metadata server
-2	Unable to set the attribute
-3	No objects match the 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;
```

Related Functions

- “METADATA_GETPROP Function” on page 152
- “METADATA_GETNPRP Function” on page 149

METADATA_VERSION Function

Returns the metadata server’s model version number.

Syntax

```
ver = METADATA_VERSION();
```

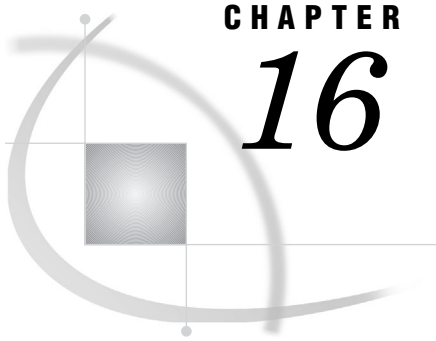
Return Values

Value	Description
<i>ver</i>	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;
```



CHAPTER

16

DATA Step Functions for Metadata Security Administration

<i>Introduction to DATA Step Functions for Metadata Security Administration</i>	171
<i>What Are the DATA Step Functions for Metadata Security Administration?</i>	171
<i>Transaction Contexts and URIs</i>	172
<i>Using the %MDSECCON() Macro</i>	173
METASEC_APPLYACT Function	173
Syntax	173
Arguments	173
Return Values	174
Details	174
Example	174
Related Functions	174
METASEC_BEGTRAN Function	174
Syntax	174
Arguments	175
Return Values	175
Details	175
Example	175
Related Functions	175
METASEC_DELACT Function	176
Syntax	176
Arguments	176
Return Values	176
Details	176
Example	176
Related Functions	177
METASEC_ENDTRAN Function	177
Syntax	177
Arguments	177
Return Values	177
Details	178
Example	178
Related Functions	178
METASEC_GETACTA Function	178
Syntax	178
Arguments	178
Return Values	179
Details	179
Example	179
Related Functions	180
METASEC_GETNACT Function	180
Syntax	180

<i>Arguments</i>	180
<i>Return Values</i>	181
<i>Details</i>	181
<i>Example</i>	182
<i>Related Functions</i>	182
<i>METASEC_GETNACTA Function</i>	182
<i>Syntax</i>	182
<i>Arguments</i>	182
<i>Return Values</i>	182
<i>Details</i>	183
<i>Example</i>	183
<i>Related Functions</i>	183
<i>METASEC_GETNAUTH Function</i>	183
<i>Syntax</i>	184
<i>Arguments</i>	184
<i>Authorizations and the %MDSECCON() Macro</i>	185
<i>About the in/out Arguments</i>	186
<i>Return Values</i>	186
<i>Details</i>	187
<i>Example</i>	187
<i>Related Functions</i>	187
<i>METASEC_GETNID Function</i>	187
<i>Syntax</i>	187
<i>Arguments</i>	187
<i>Return Values</i>	188
<i>Details</i>	189
<i>Example</i>	189
<i>Related Functions</i>	189
<i>METASEC_NEWACT Function</i>	189
<i>Syntax</i>	189
<i>Arguments</i>	189
<i>Return Values</i>	190
<i>Details</i>	190
<i>Example</i>	191
<i>Related Functions</i>	191
<i>METASEC_REMACT Function</i>	191
<i>Syntax</i>	191
<i>Arguments</i>	191
<i>Return Values</i>	192
<i>Details</i>	192
<i>Example</i>	192
<i>Related Functions</i>	192
<i>METASEC_SETACTA Function</i>	192
<i>Syntax</i>	193
<i>Arguments</i>	193
<i>Return Values</i>	193
<i>Details</i>	193
<i>Example</i>	194
<i>Related Functions</i>	194
<i>METASEC_SETAUTH Function</i>	194
<i>Syntax</i>	194
<i>Arguments</i>	195
<i>Return Values</i>	196
<i>Details</i>	196

<i>Example</i>	196
<i>Related Functions</i>	196
<i>Examples: DATA Step Functions for Metadata Security Administration</i>	196
<i>Overview</i>	196
<i>Example: Begin and End Transaction Context</i>	197
<i>Example: Working with ACTs</i>	198

Introduction to 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:

Task	Functions	Example
Transaction context control	“METASEC_BEGTRAN Function” on page 174	“Example: Begin and End Transaction Context” on page 197
	“METASEC_ENDTRAN Function” on page 177	
Access control definition	“METASEC_APPLYACT Function” on page 173	“Example: Working with ACTs” on page 198
	“METASEC_GETNACT Function” on page 180	
	“METASEC_GETNAUTH Function” on page 183	
	“METASEC_GETNID Function” on page 187	
	“METASEC_REMACT Function” on page 191	
	“METASEC_SETAUTH Function” on page 194	
ACT manipulation	“METASEC_DELACT Function” on page 176	“Example: Working with ACTs” on page 198
	“METASEC_GETACTA Function” on page 178	
	“METASEC_GETNACTA Function” on page 182	
	“METASEC_NEWACT Function” on page 189	
	“METASEC_SETACTA Function” on page 192	

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 132.

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 6.

- 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.

METASEC_APPLYACT Function

Applies ACT to the access controls on an object.

Syntax

```
rc = METASEC_APPLYACT(tc,uri,act_uri<,flags>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>uri</i>	in	Character variable or constant that contains the URI of the object to which you are applying the ACT

Argument	Direction	Description
<i>act_uri</i>	in	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”
<i>flags</i>	in	Not currently used; set to 0 (zero)

Return Values

Value	Description
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*.

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_GETNACT Function” on page 180
- “METASEC_GETNAUTH Function” on page 183
- “METASEC_GETNID Function” on page 187
- “METASEC_REMACT Function” on page 191
- “METASEC_SETAUTH Function” on page 194

METASEC_BEGTRAN Function

Begins the TC.

Syntax

```
rc = METASEC_BEGTRAN(uri,flags,tc);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Character variable or constant that contains the URI of the object to be manipulated by the transaction
<i>flags</i>	in	Not currently used; set to 0 (zero)
<i>tc</i>	out	Character variable that contains the handle of the new TC; must be at least \$16

Return Values

Value	Description
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 (<i>flag</i>) 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*.

Example

- “Example: Begin and End Transaction Context” on page 197
-

Related Functions

- “METASEC_ENDTRAN Function” on page 177

METASEC_DELACT Function

Deletes ACT from the metadata server.

Syntax

```
rc = METASEC_DELACT(tc,act_uri);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context; if <i>tc</i> is returned from the METASEC_BEGTRAN function, then <i>tc</i> references an existing ACT
<i>act_uri</i>	in	Character variable or constant that contains the URI of the ACT that you are deleting; use the following form of URI: "omsobj:AccessControlTemplate/xxxxxxx.yyyyyyyy"

Return Values

Value	Description
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*.

Example

- "Example: Working with ACTs" on page 198

Related Functions

- “METASEC_GETACTA Function” on page 178
- “METASEC_GETNACTA Function” on page 182
- “METASEC_NEWACT Function” on page 189
- “METASEC_SETACTA Function” on page 192

METASEC_ENDTRAN Function

Ends the TC.

Syntax

```
rc = METASEC_ENDTRAN(uri,tc,flags);
```

Arguments

Argument	Direction	Description
<i>uri</i>	in	Character variable or constant that contains the URI of the object to be manipulated by the transaction
<i>tc</i>	in	Character variable that contains the handle of the TC to be ended
<i>flags</i>	in	Integer bit field that specifies whether the transaction should be committed; use one of the following macro variables from %MDSECCON() _SECAD_COMMIT_TC Commit transaction _SECAD_DISCARD_TC Do not commit transaction For more information, see “Using the %MDSECCON() Macro” on page 173.

Return Values

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

Value	Description
-4	Numeric value (<i>flag</i>) 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*.

Example

- “Example: Begin and End Transaction Context” on page 197

Related Functions

- “METASEC_BEGTRAN Function” on page 174

METASEC_GETACTA Function

Returns an ACT attribute.

Syntax

```
rc = METASEC_GETACTA(tc,act_uri,attr,attr_value);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	TC handle; can be an empty string " " to invoke with a temporary context; if <i>tc</i> is returned from the METASEC_BEGTRAN function, then <i>tc</i> references an existing ACT
<i>act_uri</i>	in	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/xxxxxxx.yyyyyyy”

Argument	Direction	Description
<i>attr</i>	in	Character variable that specifies the ACT attribute whose value you are requesting; see “Details” on page 179.
<i>attr_value</i>	out	Character variable that contains the value of the ACT attribute; see “Details” on page 179.

Return Values

Value	Description
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*.

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.

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

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_DELECT Function” on page 176
- “METASEC_GETNACTA Function” on page 182
- “METASEC_NEWACT Function” on page 189
- “METASEC_SETACTA Function” on page 192

METASEC_GETNACT Function

Returns the *n*th ACT.

Syntax

```
rc = METASEC_GETNACT(tc,uri,n,act_uri,name,desc,use<,flags>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>uri</i>	in	<p>Character variable or constant that contains the URI of the object from which you want to return the ACTs.</p> <p>If the URI is for a repository (in the form "ReposID:xxxxxxx"), then the metadata server function GetAccessControlTemplateList() is called to obtain the ACT information.</p> <p>If the URI is for an object (in the form "omsobj:ObjectType/xxxxxxx.yyyyyyy"), then GetACTsOnObj() is called.</p> <p>Search syntax is supported, such as "omsobj:ObjectType?@Name='My Object' "</p>
<i>n</i>	in	One-based numeric index value that indicates which row to return from the array; for information, see “Array Parameters” on page 132
<i>act_uri</i>	out	Character variable that contains the URI of the ACT that is requested; this URI is in the following form: "omsobj:AccessControlTemplate/xxxxxxx.yyyyyyy"
<i>name</i>	out	Character variable that contains the name of the <i>n</i> th ACT
<i>desc</i>	out	Character variable that contains the description of the <i>n</i> th ACT

Argument	Direction	Description
<i>use</i>	out	Character variable that contains the value of the USE attribute of the <i>n</i> th ACT. If the ACT is for a repository, the returned value is "REPOS". Otherwise the returned value is an empty string.
<i>flags</i>	in	<p>Optional integer bit field; if the <i>uri</i> argument is in the form ReposID:yyyyyyyyy (that is, GetAccessControlTemplateList() is called), you can use the following macro variable from %MDSECCON();</p> <p style="padding-left: 40px;">_SECAD_REPOS_DEPENDENCY_USES</p> <p style="padding-left: 40px;">Return all ACTs in all SAS Metadata Repositories that are not of type PROJECT</p> <p>For more information see “Using the %MDSECCON() Macro” on page 173.</p>

Return Values

Value	Description
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 (<i>n</i>) exceeds the maximum usable value
-5	<i>n</i> 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*.

Example

- “Example: Working with ACTs” on page 198
-

Related Functions

- “METASEC_APPLYACT Function” on page 173
 - “METASEC_GETNAUTH Function” on page 183
 - “METASEC_GETNID Function” on page 187
 - “METASEC_REMACT Function” on page 191
 - “METASEC_SETAUTH Function” on page 194
-

METASEC_GETNACTA Function

Returns the *n*th attribute for an ACT.

Syntax

```
rc = METASEC_GETNACTA(tc,act_uri,n,attr,attr_value);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context; if <i>tc</i> is returned from the METASEC_BEGTRAN function, then <i>tc</i> references an existing ACT
<i>act_uri</i>	in	Character variable that contains the URI of the ACT that is requested; this URI is in the following form: “omsobj:AccessControlTemplate/xxxxxxx.yyyyyyyy”
<i>n</i>	in	One-based numeric index value that indicates which row to return from the array; for more information, see “Array Parameters” on page 132
<i>attr</i>	out	Character variable that contains the name of the <i>n</i> th attribute found on the ACT; see “Details” on page 183.
<i>attr_value</i>	out	Character variable that contains the value of the <i>n</i> th attribute found on the ACT; see “Details” on page 183.

Return Values

Value	Description
0	Successful completion, but no ACTs are found
-1	Unable to connect to the metadata server
-4	Numeric value (<i>n</i>) exceeds the maximum usable value
-5	<i>n</i> 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*.

The following table provides more information about the *attr* and *attr_value* arguments.

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

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_DELACT Function” on page 176
- “METASEC_GETACTA Function” on page 178
- “METASEC_NEWACT Function” on page 189
- “METASEC_SETACTA Function” on page 192

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>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>uri</i>	in	Character variable or constant that contains the URI of the object that is requested; can be an empty string " " if <i>tc</i> is specified; you can optimize performance by using the following form of URI: omsobj:metatype/identifier.identifier
<i>n</i>	in	One-based numeric index value that indicates which row to return from the array; for more information, see “Array Parameters” on page 132
<i>type</i>	in/out	Character variable that contains the identity type. The variable should be large enough to store the two available values, IdentityGroup or Person, probably at least \$13. 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 <i>name</i> argument; for more information, see “About the in/out Arguments” on page 186.
<i>name</i>	in/out	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 <i>type</i> argument; for more information, see “About the in/out Arguments” on page 186.
<i>auth</i>	out	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 185.
<i>perm</i>	in/out	For input, it is an optional, comma-delimited list of permission names for which authorizations are requested;for more information, see “About the in/out Arguments” on page 186. If this argument is empty, all available permissions are returned. For output, it is a character variable that contains the name of the permission whose grant or deny state is specified in the <i>auth</i> argument.

Argument	Direction	Description
<i>cond</i>	out	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
<i>flags</i>	in	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 <i>tc</i> or <i>uri</i> argument references an ACT _SECAD_DO_NOT_RETURN_PERMCOND Do not return any available values for the <i>cond</i> argument For more information see “Using the %MDSECCON() Macro” on page 173.
<i>display</i>	out	Optional; character variable that contains the value of the DisplayName attribute if the identity has a DisplayName attribute

Authorizations and the %MDSECCON() Macro

As noted in “Arguments” on page 184, 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 173. For suggested usage, see “Example: Working with ACTs” on page 198.

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

Authorization Type	Macro Variable	Description
ACT grant	<code>_SEC_PERM_ACTG</code>	Grant that originates from an ACT other than the default ACT
ACT mask	<code>_SEC_PERM_ACTM</code>	Mask to extract indirect value that originates from an ACT other than the default ACT
Indirect deny	<code>_SEC_PERM_NDRD</code>	Indirect deny that originates from an IdentityGroup membership, through inheritance, or from the default ACT; an indirect value is always returned
Indirect grant	<code>_SEC_PERM_NDRG</code>	Indirect grant that originates from an IdentityGroup membership, via inheritance, or from the default ACT; an indirect value is always returned
Indirect mask	<code>_SEC_PERM_NDRM</code>	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);
```

Return Values

Value	Description
0	Successful completion
-1	Unable to connect to the metadata server

Value	Description
-2	Error parsing <i>type</i> or <i>name</i> input list
-3	No objects match the specified URI
-4	Numeric value (flag) exceeds the maximum usable value
-5	<i>n</i> 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*.

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_APPLYACT Function” on page 173
- “METASEC_GETNACT Function” on page 180
- “METASEC_GETNID Function” on page 187
- “METASEC_REMACT Function” on page 191
- “METASEC_SETAUTH Function” on page 194

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>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>uri</i>	in	Character variable or constant that contains the URI of the object to be manipulated by the transaction; can be an empty string " " if <i>tc</i> is specified
<i>n</i>	in	One-based numeric index value that indicates which row to return from the array; for more information, see “Array Parameters” on page 132
<i>type</i>	out	Character variable that contains the identity type; the variable should be large enough to store the two available values, IdentityGroup or Person, probably at least \$13
<i>name</i>	out	Character variable that contains the identity name, which must be unique for every identity of that type on the SAS Metadata Server
<i>flags</i>	in	Optional integer bit field; you can use one of the following macro variables from %MDSECCON() <p>_SECAD_ACT_CONTENTS</p> <p>If the <i>uri</i> argument references an ACT, returns the identities that define the ACT and not the identities from the access controls that protect the ACT.</p> <p>_SECAD_RETURN_ROLE_TYPE</p> <p>Return roles as the type for IdentityGroups that are acting as roles.</p> <p>For more information see “Using the %MDSECCON() Macro” on page 173.</p>
<i>display</i>	out	Optional; character variable that contains the value of the DisplayName attribute if the identity has a DisplayName attribute
<i>origin</i>	out	Optional; indicates where the identity originates for the object in security: <p>D</p> <p>The identity originates from an ACT or ACE that is directly attached to the object</p> <p>I</p> <p>The identity originates from inheritance</p> <p>DI</p> <p>The identity originates from inheritance but is also involved with direct access controls on the object</p>

Return Values

Value	Description
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	<i>n</i> 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*.

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_APPLYACT Function” on page 173
- “METASEC_GETNACT Function” on page 180
- “METASEC_GETNAUTH Function” on page 183
- “METASEC_REMACT Function” on page 191
- “METASEC_SETAUTH Function” on page 194

METASEC_NEWACT Function

Creates a new ACT.

Syntax

```
rc = METASEC_NEWACT(tc, repos_uri<,attr,attr_value>...<,attr_3,attr_value_3>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>repos_uri</i>	in	Character variable or constant that contains the URI of the repository where you are creating the ACT; use the following form of URI: "Reposid:xxxxxxx"
<i>attr</i>	in	Character variable or constant that specifies an ACT attribute. You must pair this argument with an <i>attr_value</i> argument, and you can specify up to three <i>attr</i> and <i>attr_value</i> pairs. See "Details" on page 190.
<i>attr_value</i>	in	Character variable or constant that contains the value of an ACT attribute; you must pair this argument with an <i>attr</i> argument, and you can specify up to three <i>attr</i> and <i>attr_value</i> pairs. See "Details" on page 190.

Return Values

Value	Description
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*.

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.

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. △

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_DELACT Function” on page 176
- “METASEC_GETACTA Function” on page 178
- “METASEC_GETNACTA Function” on page 182
- “METASEC_SETACTA Function” on page 192

METASEC_REMACT Function

Removes an ACT from the object's access controls.

Syntax

```
rc = METASEC_REMACT(tc,uri,act_uri<,flags>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>uri</i>	in	Character variable or constant that contains the URI of the object from which you want to remove the ACT
<i>act_uri</i>	in	Character variable or constant that contains the URI of the ACT that you are removing; use the following form of URI: "omsobj:AccessControlTemplate/xxxxxxx.yyyyyyy"
<i>flags</i>	in	Not currently used; set to 0 (zero)

Return Values

Value	Description
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*.

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_APPLYACT Function” on page 173
- “METASEC_GETNACT Function” on page 180
- “METASEC_GETNAUTH Function” on page 183
- “METASEC_GETNID Function” on page 187
- “METASEC_SETAUTH Function” on page 194

METASEC_SETACTA Function

Sets an ACT attribute.

Syntax

```
rc = METASEC_SETACTA(tc,act_uri,attr,attr_value);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context; if <i>tc</i> is returned from the METASEC_BEGTRAN function, then <i>tc</i> references an existing ACT
<i>act_uri</i>	in	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/xxxxxxx.yyyyyyy"
<i>attr</i>	in	Character variable that specifies the ACT attribute that you are setting; see "Details" on page 193.
<i>attr_value</i>	out	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" on page 193.

Return Values

Value	Description
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*.

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.

Attribute	Attribute Value	Maximum Length of Attribute Value	Notes
NAME	Character string	60	Optional; the attribute value must be unique within the repository.
DESC	Character string	200	Optional
USE	REPOS or empty string	5	Optional; when you specify REPOS , the ACT becomes the new default repository ACT. When you specify an empty string, the ACT is removed from being the default repository ACT. See the following caution.

CAUTION:

Passing in an empty string for the USE attribute is not recommended. Passing in an empty string has an effect only when the ACT already has **USE=REPOS**. However, setting a repository ACT's USE attribute to a blank leaves the repository in a default mode where all permissions are granted. If you want to change the default ACT, it is recommended that you set USE=REPOS on the ACT that you want to use as the repository ACT. The metadata server automatically removes the USE=REPOS attribute from the previous repository ACT. Thus the repository is not left in a mode with no repository ACT. \triangle

Example

- \square “Example: Working with ACTs” on page 198

Related Functions

- \square “METASEC_DELACT Function” on page 176
- \square “METASEC_GETACTA Function” on page 178
- \square “METASEC_GETNACTA Function” on page 182
- \square “METASEC_NEWACT Function” on page 189

METASEC_SETAUTH Function

Sets authorization for an object.

Syntax

```
rc = METASEC_SETAUTH(tc,uri,type,name,auth,perm,cond<,flags>);
```

Arguments

Argument	Direction	Description
<i>tc</i>	in	Transaction context handle; can be an empty string " " to invoke with a temporary context
<i>uri</i>	in	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
<i>type</i>	in	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
<i>name</i>	in	Character variable that contains the identity name
<i>auth</i>	in	Character variable that indicates the authorization to set for the permission and identity (which are specified in the <i>perm</i> and <i>name</i> arguments, respectively); specify one of the following values: <div style="margin-left: 40px;"> G Grant D Deny R Remove </div>
<i>perm</i>	in	Character variable that contains the name of the permission whose grant, deny, or remove state is specified in the <i>auth</i> argument
<i>cond</i>	in	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.
<i>flags</i>	in	Optional integer bit field; you can use one of the following macro variables from %MDSECCON() <div style="margin-left: 40px;"> _SECAD_ACT_CONTENTS Return the authorizations that define the contents of an ACT when the <i>tc</i> or <i>uri</i> argument references an ACT For more information see “Using the %MDSECCON() Macro” on page 173. </div>

Return Values

Value	Description
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*.

Example

- “Example: Working with ACTs” on page 198

Related Functions

- “METASEC_APPLYACT Function” on page 173
- “METASEC_GETNACT Function” on page 180
- “METASEC_GETNAUTH Function” on page 183
- “METASEC_GETNID Function” on page 187
- “METASEC_REMACT Function” on page 191

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 administration books in “Recommended Reading” on page 207. △

Example: Begin and End Transaction Context

```

options metaserver="myserver"
        metaport=8561
        metauser="myuser"
        metapass="mypwd"
        metarepository="Foundation";

/* Get macro variable bit flags. */
%mdsecon();

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. */
%mdsecon();

/*-----*/
/* 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;

```



```

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. */
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;

```

```

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;

```

```
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.

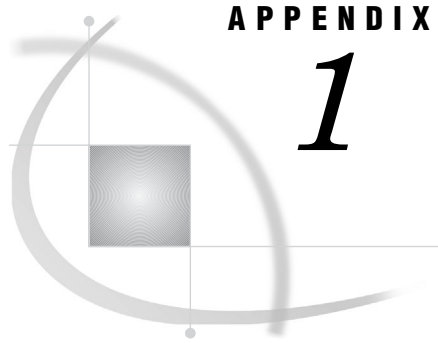
The following code deletes the table, the ACT, and the person by name, with the search form of URI. The calls in the code are submitted without a transaction context, so the changes are made immediately.

```
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;
```

APPENDIX

1

Recommended Reading

Recommended Reading 207

Recommended Reading

Here is the recommended reading list for this title:

- *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 Metadata Model: Reference*
- *SAS Open Metadata Interface: Reference*
- *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 at:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: 1-800-727-3228
Fax: 1-919-531-9439
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/bookstore**

Customers outside the United States and Canada, please contact your local SAS office for assistance.

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.

ARM (Application Response Measurement)

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.

attribute

a characteristic that is part of the standard metadata for an object. Examples of attributes include the object's name, creation date, and modification date.

authentication

the process of verifying the identity of a person or process within the guidelines of a specific authorization policy.

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 method of running SAS programs in which you prepare a file that contains SAS statements plus any necessary operating system control statements and submit the file to the operating system. Execution is completely separate from other operations at your terminal. Batch mode is sometimes referred to as running in the background.

column

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

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 only the intended recipient can read or use.

engine

a component of SAS software that reads from or writes to a file. Each engine enables SAS to access files that are in a particular file format.

job

a metadata object that specifies processes that create output.

libref

a short name for the full physical name of a SAS library. In the context of the SAS Metadata Repository, a libref is associated with a SAS library when the library is defined in the metadata repository.

localhost

a 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

data about data. For example, metadata typically describes resources that are shared by multiple applications within an organization. These resources can include software, servers, data sources, network connections, and so on. Metadata can also be used to define application users and to manage users' access to resources. Maintaining metadata in a central location is more efficient than specifying and maintaining the same information separately for each application.

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 provides metadata management services to one or more client applications. A SAS Metadata Server is an example.

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.

row

in relational database management systems, the horizontal component of a table. A row is analogous to a SAS observation.

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. See also SAS data set and SAS data view.

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 that are stored 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. SAS data views can be created by the SAS DATA step and by the SAS SQL procedure.

SAS library

a collection of one or more files that are recognized by SAS and that are referenced and stored as a unit. SAS libraries can be defined in a SAS Metadata Repository to provide centralized definitions for SAS applications.

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 Repository

one or more files that store metadata about application elements. Users connect to a SAS Metadata Server and use the SAS Open Metadata Interface to read metadata from or write metadata to one or more SAS Metadata Repositories. The metadata types in a SAS Metadata Repository are defined by the SAS Metadata Model.

SAS Metadata Server

a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories. The SAS Metadata Server uses the Integrated Object Model (IOM), which is provided with SAS Integration Technologies, to communicate with clients and with other servers.

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. See also SAS data set.

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 SAS Data Integration Studio, a metadata object that specifies how to extract data, transform data, or load data into data stores. Each transformation that you specify in a process flow diagram generates or retrieves SAS code. You can specify user-written code in the metadata for any transformation in a process flow diagram.

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.

Index

A

- access controls
 - applying ACT to 173
 - removing ACT from 191
- accessibility features 4
- ACT
 - applying to access controls on an object 173
 - attributes 178, 182, 192
 - creating 189
 - deleting from metadata server 176
 - removing from object's access controls 191
 - returning nth ACT 180
 - working with 198
- ACTION= argument
 - METAOPERATE procedure 120
- action arguments
 - METAOPERATE procedure 120
- ACTION= EMPTY argument
 - METAOPERATE procedure 127
- ACTION=PAUSE argument
 - METAOPERATE procedure 125
- ACTION=REFRESH argument
 - METAOPERATE procedure 126
- ACTION=RESUME argument
 - METAOPERATE procedure 127
- ACTION=STATUS argument
 - METAOPERATE procedure 124
- add actions
 - overriding 108
- address
 - of metadata server 39
- administration tasks
 - pausing metadata server for 7
- ARM logging 126
- array parameters 132
- association lists 162
- attributes
 - ACT 178, 182, 192
 - nth attribute of specified object 145
 - setting 165
 - value of 140
- AuthenticationDomain metadata type 66
- authorization
 - metadata engine and 47
 - nth authorization for an object 183
 - setting for objects 194

B

- Base SAS engine
 - LIBNAME statement constructed for 63
- best practices 131

C

- code testing 92
- Column metadata type 67
- columns 3
- configuration files
 - invoking connection profile 25
 - specifying connection options in 24
- connection options 24
 - specifying directly 24
 - specifying in configuration file 24
 - specifying stored connection profiles 25
 - specifying with OPTIONS statement 25
- connection parameters
 - specifying 101
- connection profiles
 - for connecting to metadata server 29
 - invoking 25
 - specifying stored profiles 25
 - XML document containing 36
- CONOPTSET= argument
 - constructing a LIBNAME statement 48
 - LIBNAME statement, metadata engine 55
 - obtaining option values from metadata objects 48

D

- data access
 - Oracle engine versus metadata engine 75
- data set options 4, 55
 - METAOUT= 55
 - OPTSET= 50, 56
- data sources 3
 - accessing tables in 55
 - output processing of tables in 54
 - specifying 101
 - synchronizing metadata with 111
- DATA step
 - creating reports with 14
- DATA step functions 4, 131
 - array parameters 132
 - best practices 131
 - compared with metadata procedures 137

- for reading and writing metadata 135
- for security administration 171
- for security administration, examples 196
- DatabaseSchema metadata type 67
- DBMS
 - Relational DBMS Model 60
 - Remote Relational DBMS Model 60
- DBMS data 99
- DBMS SAS/ACCESS engine
 - LIBNAME statement constructed for 64
- delete actions 108
- DELETE argument
 - UPDATE_RULE statement (METALIB) 108
- Directory metadata type 68

E

- EMPTY action 127
- encryption 30
 - level of 32
- encryption options 25
- engines
 - See also* metadata engine
 - DBMS SAS/ACCESS engine 64
 - Oracle engine versus metadata engine 75
 - REMOTE 65
 - underlying 45
- examples
 - creating property set object for LIBOPTSET= 8
 - creating reports with DATA step 14
 - creating reports with METADATA procedure and XML engine 9
 - data access with Oracle versus metadata engine 75
 - functions for security administration 196
 - METADATA procedure 90
 - METALIB procedure 111
 - METAOPERATE procedure 124
 - pausing metadata server for administration tasks 7
 - submitting LIBNAME statement 75
- EXCLUDE statement
 - METALIB procedure 104

F

- filerefs
 - to temporary file with IN= argument 92
 - with IN= and OUT= arguments 91
- folder objects
 - Id and Type attributes 155
- FOLDER statement
 - METALIB procedure 105
- FOLDERID statement
 - METALIB procedure 105
- folders
 - storing metadata in 105
- functions
 - See* DATA step functions

H

- HEADER= argument
 - PROC METADATA statement 86, 93
- host name or address 39

I

- Id attribute
 - of folder objects 155
- identifiers 5
 - See also* URI
 - obtaining 5
- impact analysis 107, 113
- IMPACT_LIMIT statement
 - METALIB procedure 106
- IN= argument
 - fileref to temporary file 92
 - filerefs with 91
 - METADATA procedure 86
- Index metadata type 68
- information maps 99
- input argument
 - METADATA procedure 86
- input XML string 87

J

- Job objects
 - maximum number for table definition updates 106
- journaling 126

L

- language elements 3
 - metadata objects listed by 63
 - when to use 3
- LIBID= argument
 - OMR statement (METALIB) 101
- LIBNAME statement, metadata engine 3, 51
 - CONOPTSET= argument 55
 - constructed for Base SAS engine 63
 - constructed for DBMS SAS/ACCESS engine 64
 - constructed for REMOTE engine 65
 - constructing 47, 48
 - LIBOPTSET= argument 55
 - METAOUT= argument 54
 - overview 45
 - required arguments 52
 - server connection arguments 53
 - submitting 75
 - syntax 51
- LIBOPTSET= argument
 - constructing a LIBNAME statement 48
 - creating PropertySet object for 8
 - LIBNAME statement, metadata engine 55
 - obtaining option values from metadata objects 48
- libraries 3
- librefs 45
- logging, ARM 126
- Login metadata type 69

M

- MATCHING argument
 - REPORT statement (METALIB) 108
- %MDSECCON() macro 173
- METAAUTORESOURCES system option 27
- METABROWSE command 3
- METACON command 3
- METACONNECT= system option 29

- metadata
 - adding prefix to names 114
 - adding prefixes to names 107
 - DATA step functions for reading and writing 135
 - report summarizing changes in 107
 - requesting, for one object 95
 - requesting, for one type of object 96
 - setting defaults for 3
 - storing in folders 105
 - suppressing changes 106
 - synchronizing with data source 111
 - updating 109
- metadata associations
 - deleting all objects 137
 - modifying association list 162
 - nth associated object 143
 - nth association for specified object 142
 - represented as XML element 89
- metadata definitions 66
- metadata engine 45
 - See also* LIBNAME statement, metadata engine
 - advantages of 46
 - authorization and 47
 - constructing a LIBNAME statement 47
 - constructing options 47
 - data set options for 55
 - librefs 45
 - metadata types and 66
 - output processing of tables 54
 - process 45
 - requirements for 59
 - SAS Metadata Model 59
 - supported features 46
 - versus Oracle engine 75
- metadata identifiers 5
 - obtaining 5
- metadata language elements 3
 - metadata objects listed by 63
 - when to use 3
- metadata LIBNAME statement
 - See* LIBNAME statement, metadata engine
- metadata modeling 59
- metadata names 5
- metadata objects 63
 - adding prefix to names 107
 - applying ACT to access controls 173
 - creating 153
 - deleting first object matching the URI 139
 - deleting objects making up an association 137
 - listed by language element 63
 - listed by type 66
 - nth associated object 143
 - nth association for specified object 142
 - nth attribute 145
 - nth authorization for 183
 - nth identity for 187
 - nth object matching specified URI 147
 - nth object type on metadata server 151
 - nth property of 149
 - obtaining option values from 48
 - referencing with URI 136
 - removing ACT from access controls 191
 - represented as XML element 88
 - requesting metadata for one 95
 - requesting metadata for one type of 96
 - resolving URI into 160
 - setting attributes 165
 - setting authorization for 194
 - setting properties 166
 - URI of specified property for 152
 - value of specified attribute 140
- METADATA procedure 83, 84
 - changing a repository's state 90
 - compared with METAOPERATE procedure 81
 - concepts 87
 - creating reports with 9
 - examples 90
 - filerefs 91
 - filerefs to temporary files 92
 - input argument 86
 - input XML string 87
 - metadata association represented as XML element 89
 - metadata object represented as XML element 88
 - method parameter represented as XML element 88
 - method represented as XML element 88
 - output arguments 86
 - requesting metadata for one object 95
 - requesting metadata for one type of object 96
 - results 89
 - server connection arguments 85
 - server status request 91
 - syntax 84
 - task tables 84
- metadata resources
 - assigned at startup 27
- metadata server
 - connection options for 24
 - connection profile for connecting to 29
 - deleting ACT from 176
 - determining if paused 81, 157
 - encryption level 32
 - encryption type 30
 - host name or address 39
 - model version number 167
 - network protocol for connecting to 37
 - nth object type 151
 - password for 33
 - pausing and resuming 126, 127
 - pausing for administration tasks 7
 - resolving URI into objects 160
 - SAS Metadata Repository for 38
 - sending XML strings to 83
 - specifying connection parameters 101
 - specifying data source 101
 - SPN for 40
 - status request 91, 124
 - TCP port for 35
 - user ID for 42
 - XML document containing connection profiles for 36
- metadata types 66
 - metadata engine and 66
- METADATA_DELASSN function 137
- METADATA_DELOBJ function 139
- METADATA_GETATTR function 140
- METADATA_GETNASL function 142
- METADATA_GETNASN function 143
- METADATA_GETNATR function 145
- METADATA_GETNOBJ function 147
- METADATA_GETNPRP function 149
- METADATA_GETNTYP function 151
- METADATA_GETPROP function 152
- METADATA_NEWOBJ function 153

- METADATA_PATHOBJ function 155
 - METADATA_PAUSED function 157
 - METADATA_PURGE function 158
 - METADATA_RESOLVE function 160
 - METADATA_SETASSN function 162
 - METADATA_SETATTR function 165
 - METADATA_SETPROP function 166
 - METADATA_VERSION function 167
 - METAENCRYPTALG system option 30
 - METAENCRYPTLEVEL system option 32
 - METAFIND command 3
 - METALIB procedure 99, 100
 - adding prefix to metadata names 114
 - concepts 108
 - examples 111
 - EXCLUDE statement 104
 - FOLDER statement 105
 - FOLDERID statement 105
 - how it works 108
 - impact analysis 107, 113
 - IMPACT_LIMIT statement 106
 - NOEXEC statement 106
 - ODS reports 110
 - OMR statement 101
 - PREFIX statement 107
 - PROC METALIB statement 101
 - REPORT statement 107, 109
 - results 109
 - SELECT statement 104
 - selecting tables for processing 104, 112
 - server connection arguments 103
 - synchronizing metadata with data source 111
 - syntax 100
 - task tables 100, 101
 - UPDATE_RULE statement 108
 - updating metadata 109
 - METAOPERATE procedure 117
 - action arguments 120
 - ACTION=EMPTY 127
 - ACTION=PAUSE with pause comment 125
 - ACTION=REFRESH to pause and resume metadata server 126
 - ACTION=REFRESH with ARM logging 126
 - ACTION=REFRESH with journaling 126
 - ACTION=RESUME 127
 - actions and repositories 123
 - ACTION=STATUS 124
 - compared with METADATA procedure 81
 - concepts 123
 - examples 124
 - how it works 123
 - server connection arguments 118
 - syntax 118
 - task tables 118
 - METAOUT= argument
 - LIBNAME statement, metadata engine 54
 - METAOUT= data set option 55
 - METAPASS= system option 33
 - METAPORT= system option 35
 - METAPROFILE system option 36
 - METAPROTOCOL= system option 37
 - METAREPOSITORY= system option 38
 - METASEC_APPLYACT function 173
 - METASEC_BEGTRAN function 174
 - METASEC_DELACT function 176
 - METASEC_ENDTRAN function 177
 - METASEC_GETACTA function 178
 - METASEC_GETNACT function 180
 - METASEC_GETNACTA function 182
 - METASEC_GETNAUTH function 183
 - METASEC_GETNID function 187
 - METASEC_NEWACT function 189
 - METASEC_REMACT function 191
 - METASEC_SETACTA function 192
 - METASEC_SETAUTH function 194
 - METASERVER= system option 157, 39
 - METASPN= system option 40
 - METAUSER= system option 42
 - methods
 - parameter represented as XML element 88
 - represented as XML element 88
 - model version number
 - metadata server 167
 - modeling 59
- ## N
- names 5
 - adding prefix to metadata names 114
 - host name of metadata server 39
 - network protocol
 - for connecting to metadata server 37
 - NOADD argument
 - UPDATE_RULE statement (METALIB) 108
 - NOAUTOPAUSE argument
 - METAOPERATE procedure 121
 - NODELDUP argument
 - UPDATE_RULE statement (METALIB) 108
 - NOEXEC statement
 - METALIB procedure 106
 - NOUPDATE argument
 - UPDATE_RULE statement (METALIB) 108
- ## O
- objects
 - See also* metadata objects
 - folder objects 155
 - Job objects 106
 - maximum number for updates 106
 - PropertySet 8
 - Transformation 106
 - observations 3
 - ODS reports 110
 - OMR statement
 - METALIB procedure 101
 - options
 - constructed by metadata engine 47
 - obtaining values from metadata objects 48
 - OPTSET= data set option 50
 - OPTIONS= argument
 - METAOPERATE procedure 121
 - OPTIONS statement
 - specifying connection options with 25
 - OPTSET= data set option 50, 56
 - Oracle engine
 - versus metadata engine 75
 - OUT= argument
 - filerefs with 91
 - METADATA procedure 86
 - METAOPERATE procedure 123

- output arguments
 - METADATA procedure 86
- output processing
 - of tables in data source 54
- overriding
 - add actions 108
 - update actions 108

P

- PASSWORD= argument
 - METADATA procedure 85
 - METAOPERATE procedure 118
 - OMR statement (METALIB) 103
- passwords
 - for metadata server 33
- PAUSE action 123, 125
- pause comments 125
- pausing and resuming metadata server 126, 157
- pausing metadata server 7, 81
- PhysicalTable metadata type 69
- port
 - TCP port for metadata server 35
- PORT= argument
 - METADATA procedure 85
 - METAOPERATE procedure 119
 - OMR statement (METALIB) 103
- PREFIX statement
 - METALIB procedure 107
- prefixes
 - adding to metadata names 107, 114
- PROC METADATA statement 84
 - input argument 86
 - output arguments 86
 - server connection arguments 85
- PROC METALIB statement 101
- PROC METAOPERATE statement 118
 - action arguments 120
 - server connection arguments 118
- procedures 4, 81
 - See also* METADATA procedure
 - compared with DATA step functions 137
- properties
 - nth property of specified object 149
 - setting 166
 - URI of specified property 152
- Property metadata type 70
- PropertySet metadata type 71
- PropertySet object
 - creating for LIBOPTSET= argument 8
- PROTOCOL= argument
 - METADATA procedure 85
 - METAOPERATE procedure 119
 - OMR statement (METALIB) 103
- purging URIs 158

R

- reading metadata 135
- REFRESH action 123
 - pausing and resuming metadata server 126
 - with ARM logging 126
 - with journaling 126
- Relational DBMS Model 60
- REMOTE engine
 - LIBNAME statement constructed for 65

- Remote Relational DBMS Model 60
- Remote SAS Data Set Model 60
- REPID= argument
 - OMR statement (METALIB) 103
- REPORT statement
 - METALIB procedure 107, 109
- reports
 - creating with DATA step 14
 - creating with METADATA procedure and XML engine 9
 - details in 110
 - ODS 110
 - summarizing metadata changes 107
- repository
 - changing state of 90
 - effect of PAUSE, REFRESH, and RESUME actions 123
 - SAS Metadata Repository for metadata server 38
- REPOSITORY= argument
 - METADATA procedure 85
 - METAOPERATE procedure 119
- resolving URIs 160
- resource option 26
- RESUME action 123, 127
- resuming metadata server 126, 127
- rows 3

S

- SAS/ACCESS Interface to Oracle engine
 - versus metadata engine 75
- SAS Data Set Model 60
- SAS Metadata Model 59
- SAS Metadata Repository
 - for metadata server 38
- SASClientConnection metadata type 72
- SASLibrary metadata type 73
- Section 508 4
- security administration
 - DATA step functions for 171
 - DATA step functions for, examples 196
- SELECT statement
 - METALIB procedure 104
- SERVER= argument
 - METADATA procedure 85
 - METAOPERATE procedure 119
 - OMR statement (METALIB) 104
- server connection arguments
 - LIBNAME statement, metadata engine 53
 - METADATA procedure 85
 - METALIB procedure 103
 - METAOPERATE procedure 118
- SPN (service principal name) 40
- state of repository 90
- STATUS action 124
- status of metadata server 91
- status request for metadata server 124
- stored connection profiles 25
- synchronizing metadata 111
- system options 3
 - by category 23
 - connection options 24
 - encryption options 25
 - overview 23
 - resource option 26
 - viewing settings 23

T

- table definitions 99
 - maximum number of objects for updates 106
- tables 3, 99
 - accessing in data source 55
 - excluding for processing 104
 - output processing of 54
 - selecting for processing 104, 112
- TCP port
 - for metadata server 35
- temporary files
 - fileref to, with IN= argument 92
- terminology 3
- testing code 92
- transaction contexts 172
 - beginning and ending 174, 177, 197
- Transformation objects
 - maximum number for table definition updates 106
- TYPE= argument
 - REPORT statement (METALIB) 107
- Type attribute
 - of folder objects 155

U

- underlying engine 45
- Uniform Resource Identifier
 - See* URI
- update actions
 - overriding 108
- UPDATE_RULE statement
 - METALIB procedure 108
- updates
 - maximum number of objects for 106
- updating metadata 109
- URI 6
 - DATA step functions for security administration 172
 - deleting first object that matches 139

- formats 6
- nth object matching 147
- of specified property for specified object 152
- purging 158
- referencing metadata objects with 136
- resolving into metadata objects 160
- USER= argument
 - METADATA procedure 86
 - METAOPERATE procedure 119
 - OMR statement (METALIB) 104
- user ID
 - for metadata server 42

V

- variables 3
- VERBOSE argument
 - METADATA procedure 87, 94

W

- writing metadata 135

X

- XML documents
 - containing connection profiles 36
- XML elements
 - metadata association represented as 89
 - metadata object represented as 88
 - method parameter represented as 88
 - methods represented as 88
- XML engine
 - creating reports with 9
- XML strings
 - input XML string 87
 - sending to SAS Metadata Server 83

Your Turn

We welcome your feedback.

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