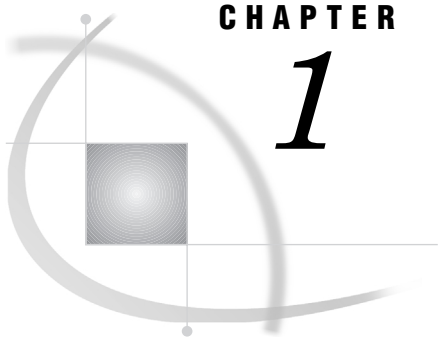


CHAPTER

1

Introduction to the Metadata API

Changes and Enhancements

Enhanced: Metadata API

The Metadata API has been expanded for READ support for jobs and scheduling servers defined in SAS/Warehouse Administrator.

New Feature: New Metadata API Types

The following SAS/Warehouse Administrator metadata API types have been added for Release 2.0:

- WHCOLOLP
- WHEVENT
- WHGRPJOB
- WHGRPOLP
- WHJOB
- WHJOBCAT
- WHJOBFIL
- WHLDOMDD
- WHLDOPRX

- WHLDOTBL
- WHMDDSTR
- WHOLAP
- WHOLPCRS
- WHOLPCUB
- WHOLPDIM
- WHOLPHIR
- WHOLPMDD
- WHOLPSTC
- WHOLPTBL
- WHPOBJCT
- WHROWSEL
- WHSERV
- WHSRVAT
- WHSRVCRN
- WHSRVNUL

Prerequisites

To get the most out of this manual, you should be familiar with

- SCL (SAS Component Language), a programming language that controls SAS/AF applications and provides complete object-oriented programming constructs for creating an entire object-oriented application in SCL
- the SAS/AF software development environment
- SCL applications using FRAME entries
- the SAS application whose metadata you want to read or write.

To use the metadata API, you will need the following SAS products in addition to API software:

- base SAS software, Release 6.12 or above
- SAS/AF software
- SAS/FSP software
- the SAS application whose metadata you want to read or write, such as SAS/Warehouse Administrator, Release 1.2 or above (includes the Job Scheduler).

SCL applications that use the metadata API must run under Release 6.12 or above of the SAS System.

What is Metadata?

Metadata is information that is internal to an application that describes elements in the application, such as tables and columns. Metadata can be divided into two main categories:

Physical metadata
is a set of software instructions that describe an application element.
For example, the physical metadata for a SAS table might specify a certain number of rows and columns, with certain data transformations applied to some columns.

Business metadata
is text that describes the content or purpose of an application element.
For example, the business metadata for a SAS table might describe the purpose of the table and contact information for the person responsible for the accuracy of the information in the table.

Most SAS/Warehouse Administrator metadata contains information about data sources, data stores, and the Jobs that extract, transform, and load source data into the warehouse data stores. SAS/Warehouse Administrator metadata is stored in two or more metadata repositories.

What is a Metadata API?

A metadata application program interface (API) is a set of software tools that enable users to write applications that access metadata.

Using the SAS Metadata API, you will eventually be able to write programs that read and write the metadata that is maintained by a number of SAS products. The current API enables you to write SCL programs to access metadata in two applications:

- SAS/Warehouse Administrator is a product which integrates many of the tools in the SAS System that can be used for data warehousing
- Job Scheduler is an application that is used to schedule batch jobs in SAS products.

What Can I Do with the SAS Metadata API?

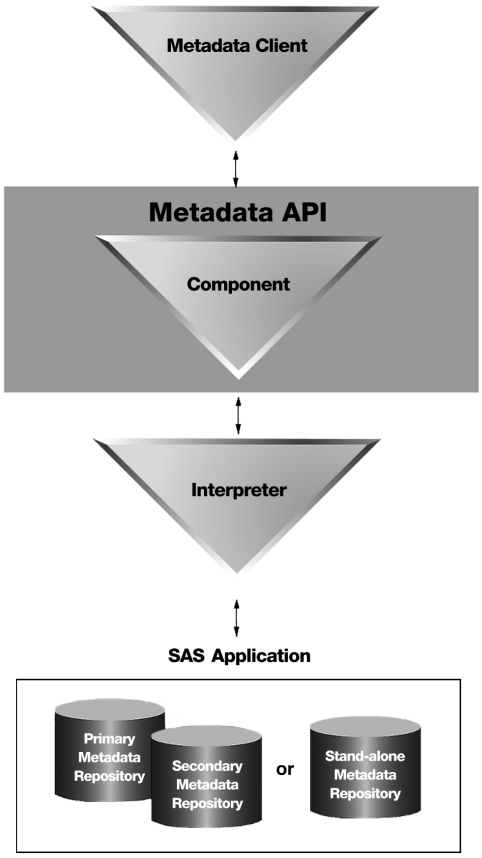
Using the SAS metadata API, you can write programs that read, add, or update the metadata in a SAS application – without going through the application's interface. You can write SCL applications that:

- publish HTML pages containing the current metadata for a SAS application
- change path names in metadata
- copy a table's metadata (in order to create a similar table, for example)
- add columns to a table
- update a column attribute
- add tables and other objects defined by metadata
- use the API in a SAS macro to generate a LIBNAME statement.

How the Metadata API Works

Figure 1.1 on page 4 illustrates how client applications written in SCL use the metadata API to read or write metadata from SAS applications.

Figure 1.1 Metadata API Model



Note: The figure shows how one *component* works with one *interpreter*; however, the metadata API will accommodate multiple components as long as each component has an appropriate interpreter. △

Metadata client
is an application that uses metadata API methods to read or write metadata. For the current release of the SAS Metadata API, metadata clients must be written in SCL.

metadata API
is a set of software tools that enable users to write applications that access metadata.

metadata type
is a template that models the metadata for a particular kind of object in an application. The parameter list for a metadata type matches the items of metadata maintained for the corresponding object.

SAS/Warehouse Administrator metadata types are listed in "Index to SAS/Warehouse Administrator Metadata Types" on page 67.

component
is a group of related metadata types. Each component has an ID (such as WHOUSE) and a name (such as SAS/Warehouse Administrator) which often match the name of the application whose metadata is modeled by the component. The metadata API will accommodate multiple components as long as each has an appropriate interpreter. The components supplied with the current API are WHOUSE (SAS/Warehouse Administrator) and JOBSCH (Job Scheduler).

application program interface (API) interpreter
is a program that translates the API metadata type requested by a client to the corresponding metadata object in a repository. The current API has two interpreters: one for SAS/Warehouse Administrator and the other for Job Scheduler.

API interpreters insulate client applications from the details of metadata repositories. If you use the metadata API, and there is an interpreter for your target repository, client applications do not need to handle the details of that repository in order to read from it or write to it. Also, if the metadata structure in a repository should change, in many cases only the interpreter would have to be updated, not the client applications using the metadata API.

SAS application
is the SAS application whose metadata you want to read or write. The current API supports two applications: SAS/Warehouse Administrator and Job Scheduler.

metadata repository
is a data store that contains an application's metadata. For example, SAS/Warehouse Administrator has multiple metadata repositories: one for each environment and one for each warehouse within an environment. Accordingly, the API provides methods for identifying primary and secondary repositories. Repositories are described in more detail in "Metadata Repositories" on page 8.

Identifying Metadata

Each metadata object in a repository, such as the metadata for a particular column in a SAS table, has a unique identifier. Each object usually has a name and a description as well. For example, here is the ID, name and description for a SAS table column, as returned by the Metadata API's _GET_METADATA_ method.

```
COLUMNS=( ( ID='A000000E.WHCOLDTL.A0000032'
              NAME='PRODNUM'
              DESC='product number'
              )[575]
```

To read or write a metadata object, you must pass a list of properties for that type to the appropriate Metadata API method. (These methods are listed in "Index to Metadata API Methods" on page 14.) The following properties are common to all metadata types. They are often referred to as the "general identifying information" for a metadata object.

ID
is the unique three-level identifier for a metadata object. It takes the form: REPOSID.TYPEID.INSTANCEID. For example, in the previous code example, the ID for the COLUMNS object is A000000E.WHCOLDTL.A0000032.

A000000E is the REPOSID assigned to a particular warehouse repository when it was created in SAS/Warehouse Administrator. A REPOSID (metadata repository

ID) is a unique 8-character string identifying the metadata repository that stores the object. Each application has one or more repositories.

WHCOLDTL is the TYPEID for a column in a SAS/Warehouse Administrator detail table. A TYPEID (metadata type ID) is a maximum 8-character string which defines the type of the metadata object. Each application has its own set of metadata types. For example, SAS/Warehouse Administrator metadata types are listed in "Index to SAS/Warehouse Administrator Metadata Types" on page 67.

A0000032 is the INSTANCEID assigned to a particular column in the detail table when it was created in SAS/Warehouse Administrator. An INSTANCEID (metadata object instance ID) is an 8-character string which distinguishes one metadata object from all other objects of the same type within a given repository.

NAME
is the name of the metadata object, up to 40 characters long. The name is from the context of the component that it comes from. For example, SAS/Warehouse Administrator names are those that appear in the Explorer, the Setup window, the Process Editor, and other frames in that application. In the previous code example, the NAME of the table column is PRODNUM.

DESC
is the description of the metadata object, up to 200 characters long. Not all objects will have a description. In the previous code example, the DESC of the table column is "product number."

CAUTION:
It is strongly recommended that you avoid coding the literal identifier of a particular metadata object in a client application. Instead, use the _GET_METADATA_OBJECTS_ method or other Metadata API methods to return an SCL list of the unique object identifiers, names, and descriptions for objects of a particular type. △

Reading Metadata: A Simple Example

The following steps illustrate how to use the API to select and display the metadata for a particular detail table in a particular data warehouse created by SAS/Warehouse Administrator. For the sake of simplicity, assume that you have already attached to the relevant metadata repositories, that the metadata you want is in the A000000E repository, and that the TYPEID for the SAS/Warehouse Administrator detail table is WHDETAIL.

1 Concatenate the DW_REPOS_ID (A000000E)with the metadata type ID (WHDETAIL) and store them in the variable TYPE.

```
type=dw_repos_id||'.WHDETAIL';
```

2 Define a list (L_OBJS) to hold the results of a read operation in the next step.

```
l_objs=makelist();
```

3 Call the _GET_METADATA_OBJECTS_ method, which accepts the REPOSID.TYPEID assigned to the TYPE variable. It then loads the L_OBJS list with the instance IDs and names of WHDETAIL objects in repository A000000E .

```
call send(i_api,'_GET_METADATA_OBJECTS_',rc,
type,l_objs);
```

4 Use the PUTLIST function to display the list in the Message Window or SASLOG.

```
 call putlist(l_objs,'WAREHOUSE OBJECTS',2);
WAREHOUSE OBJECTS
( A000000E.WHDETAIL.A000001L='Customer detail table'
  A000000E.WHDETAIL.A000002X='Product detail table'
  A000000E.WHDETAIL.A000003M='Customer detail table'
  A000000E.WHDETAIL.A000004H='Sales fact table'
  A000000E.WHDETAIL.A000005U='Oracle
  A000000E.WHDETAIL.A000006Q='Sybase'
  A000000E.WHDETAIL.A000007L='Remote Detail Table'
  A000000E.WHDETAIL.A000008I='Suppliers'
  )[421]
```

5 Search the list for the unique ID of the Product detail table and pass it to _GET_METADATA_ to retrieve information about that table.

If you are interested in particular properties for a given metadata type, you can pass those properties to the _GET_METADATA_ method as named items. For example, in the code that follows, the LIBRARY, COLUMNS, and TABLE NAME properties for the detail table metadata type are inserted in the metadata property list (*l_meta*) that is passed to the _GET_METADATA_ method.

```
index=searchc(l_objs,'Product',1,1,'Y','Y');

id=nameitem(l_objs,index);
rc=clearlist(l_meta,'D');
l_meta=insertc(l_meta,id,-1,'ID');
l_lib=makelist();
l_meta=insertl(l_meta,l_lib,-1,'LIBRARY');
l_cols=makelist();
l_meta=insertl(l_meta,l_cols,-1,'COLUMNS');
l_meta=insertc(l_meta,' ',-1,'TABLE NAME');
call send(i_api,'_GET_METADATA_',l_rc,l_meta);
rc=putlist(l_meta,'PRODUCT table',2);
```

6 The method populates these sublists with the requested information.

```
PRODUCT table( ID='A000000E.WHDETAIL.A000002X'
   LIBRARY=( ID='A0000001.WHLIBRY.A000000U'
             NAME='Warehouse Data Library'
             DESC=''
             )[405]
   COLUMNS=( ( ID='A000000E.WHCOLDTL.A0000032'
               NAME='PRODNUM'
               DESC='product number'
               )[575]
             ( ID='A000000E.WHCOLDTL.A0000034'
               NAME='PRODNAME'
               DESC='product name'
              )[643]
             ( ID='A000000E.WHCOLDTL.A0000036'
               NAME='PRODID'
               DESC='product id/abbreviation'
              )[619]
             ( ID='A000000E.WHCOLTIM.A00000FU'
               NAME='_LOADTM'
```

```
      DESC='DateTime Stamp of when row was
            loaded'
    )[621]
  )[407]
```

The API enables you to read and write many metadata objects using techniques similar to those used in these steps.

Metadata Repositories

An application's metadata can be divided into different physical stores based on a number of criteria:

- different storage locations (such as separate repositories for local and remote metadata)
- different intended users (such as separate repositories for business users and IT staff)
- different levels of access control (such as separate repositories for testing and production)

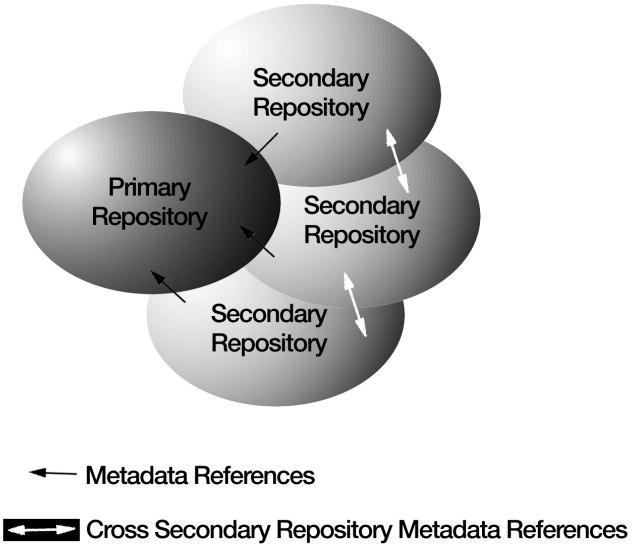
Each physical store of metadata is called a *metadata repository*. There are two main types of metadata repositories: *stand-alone* and *partitioned*.

A stand-alone repository is a single metadata store, such as repository in SAS/EIS software. Once a stand-alone repository has been accessed, all metadata is accessible. Figure 1.2 on page 8 illustrates a stand-alone repository.

Figure 1.2 Stand-Alone Metadata Repository

A partitioned repository has one or more *primary* repositories, each of which has one or more *secondary* repositories. Figure 1.3 on page 9 illustrates the relationship between a primary repository and its secondary repositories.

Figure 1.3 Partitioned Metadata Repository



Partitioning allows different kinds of metadata to be stored in different locations, in different formats, and so on. The amount of metadata that can be accessed is controlled by setting which repositories are active. Each repository in a partitioned repository has a unique repository identifier (REPOSID).

SAS/Warehouse Administrator has a partitioned metadata repository. Each primary repository stores metadata shared by all warehouses in an environment. Each secondary repository stores metadata for an individual warehouse within an environment.

Metadata that is stored in each repository may be associated with metadata in other repositories. The secondary repositories may contain references to metadata in the primary repository, but the primary repository *cannot* contain references to metadata in any of the secondary repositories (as indicated by the solid arrow in Figure 1.3 on page 9). Some partitioned repositories also support secondary repositories containing metadata references into other secondary repositories, referred to as cross-secondary repository references.

Note: The current SAS/Warehouse Administrator metadata repository does not support cross-secondary repository references. Also, it supports only a single secondary repository (metadata for one warehouse) to be active at one time. △

Setting the Active Metadata Repository

To use the metadata API, your SCL programs must attach to the repository that contains the metadata you want to read or write. This is done with the _SET_PRIMARY_REPOSITORY_ method and the _SET_SECONDARY_REPOSITORY_ method.

In the context of the "set repository" methods, *primary* refers to either a stand-alone repository or a primary repository of a partitioned repository. If the metadata you want is in a stand-alone repository, or if it is in a primary portion of a partitioned repository, there is no need to set the secondary repository.

To identify the repository where a given type of metadata resides, you could use the _GET_METADATA_OBJECTS_ method (perhaps with the SEARCH_SECONDARY

parameter). This method returns a list of all metadata objects of a given type. The REPOSID field for each object identifies the repository where the object is stored.

Learning to Use the Metadata API

Here are some steps you can take to learn the metadata API.

1. Become familiar with the elements of the metadata API: primary repository, secondary repository, types, subtypes, type names, type IDs, and so on.
2. Study the "Read Metadata Code Sample" on page 203 and the "Write Metadata Code Sample" on page 207.
3. Learn how to initialize the metadata API by executing simple API method calls that do not read any actual metadata. For example, list all the object-types that are available in the API. List the properties for a given object in the API.
4. Try some simple queries against the metadata of a well-known metadata object. Since this is just a test program, you can code the literal identifier of the object in your client application. For example, list all the detail-tables that are defined in a warehouse.
5. Try a more realistic task, perhaps using the codes samples in Appendix 1, "Sample Metadata API Code," on page 203 as a starting point.
   a. Decide what information you need.
   b. Translate this information into metadata types and attributes.
   c. Determine how the different metadata types you need are related, so that you will know how to access the metadata you want.

      For example, if you want to list all of the owners defined for a given data warehouse, and list all of the detail tables for which each owner is responsible, you must first get a list of all detail tables. Then you can list the owner of each detail table. For details about SAS/Warehouse Administrator metadata relationships, see "Relationships Among Metadata Types" on page 49.
   d. Write the client application.
   e. Run the application and compare the returned metadata with the actual metadata that you can view through the application.

Naming Conventions Used in This Manual

The following conventions have been used in the examples for this book:

- any variable that begins with *i_* is an object (an instance of a class)
- any variable that begins with *l_* is an SCL list identifier
- method names and SCL list item names appear in uppercase letters.

Where Metadata API Classes and SLISTS are Stored

The default classes and SLISTS for the Metadata API are stored in the SASHELP.METAAPI catalog.