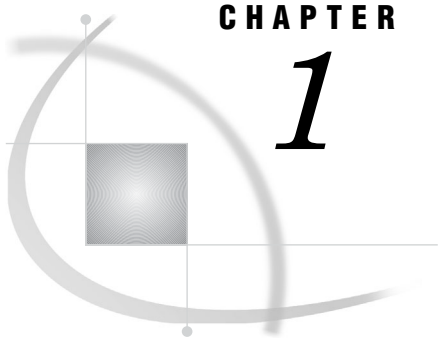


CHAPTER

1

Introduction to the Metadata API

Changes and Enhancements

This section describes changes to the SAS/Warehouse Administrator metadata API after Release 2.0.

- You can add and update the PATH property for the WHEFILE type.
- You can now use the metadata API to add, update, and delete process objects. For example, you can write a metadata API program that creates a data store and also creates all of the processes that are required to extract, transform, and load information into that data store. The following metadata types have been updated to support this feature:
  - WHCOLUMN
    - WHCOLDTL
    - WHCOLDAT
    - WHCOLODD
    - WHCOLOLP
    - WHCOLTIM
  - WHCTRNFM
  - WHEFILE
  - WHEXTATR
  - WHINDEX
  - WHOLAP
    - WOLPDIM

- WOLPHIR
- WOLPCRS
- WOLPCUB

- WHPHYSTR
  - WHDMSST
  - WHSASSTR

- WHPOBJECT
  - WHJOB
  - WHGRPJOB
  - WHEVENT

- WHTFILE
  - WHTXTFIL
  - WHSCRFIL

- WHTXTCAT
  - WHNOTE
  - WHSRCCAT
    - WHJOBCAT

- WHDW
- WHDWENV
- WHINFO
- WHINFOFL
- WHTABLE
  - WHDATTBL
  - WHDETAIL
  - WHLDETL
  - WHODDTBL
  - WHODTTBL
  - WHSUMTBL
  - WHOLPSTC
    - WHGRPOLP
    - WHOLPTBL
    - WHOLPMDD
  - WHTBLPRC
    - WHTBLMAP
    - WHTBLREC
    - WHTBLUSR
    - WHTBLXFR

- WHPROCES
  - WHPRCMAN
    - WHPRCMAP
    - WHPRCREC
    - WHPRCUSR

- WHPRCXFR
- WHPRCLDR
  - WHLDRDAT
  - WHLDRDTL
  - WHLDREXT
  - WHLDRINF
  - WHLDRIMF
  - WHLDRLDT
  - WHLDRMDB
  - WHLDRODD
  - WHLDRODT
  - WHLDRSUM
  - WHLDOTBL
  - WHLDOMDD
    - WHLDOPRX

- WHPRCSPR
  - WHPRCPST
  - WHSUBSET
  - WHROWSEL

- The TABLE OPTIONS property of the WHDBMSST type has a new sublist—the APPEND sublist. The APPEND sublist contains any SAS/ACCESS LIBNAME data set options that are used to create or load the table, such as BULKLOAD=yes.
- Load process options for warehouse tables, such as GENERATION LEVEL and DROP INDEXES, are now surfaced through the WHPRCLDR type and all of its subtypes. For example, you can write a SAS/Warehouse Administrator add-in that reads the load options that are specified in a table's load process and uses these options to load the corresponding table.
- The operating system and SAS version that are associated with a given host are now available through the WHHOST property. For example, you can write a SAS/Warehouse Administrator add-in that reads the host metadata that is associated with a given data store and then uses these values to generate code that is appropriate for the operating system and SAS version.
- You can now write OLAP objects through the metadata API. The following types have been updated:
  - WHMDDSTR
  - WHOLPSTC
  - WHGRPOLP
  - WHOLPTBL
  - WHOLPMDD
  - WHCOLOLP
  - WHOLPDIM
  - WHOLPHIR
  - WHOLPCRS
  - WHOLPCUB.

- Metadata for columns that are selected using point and click in the Expression Builder and that are used in either a WHERE clause or a row selector is now surfaced through the WHSUBSET and WHROWSEL types. For example, you can write a SAS/Warehouse Administrator add-in that reads the column metadata that is associated with a WHERE clause or a row selector and uses this metadata to generate the appropriate code.
- You can now update the EXTENDED ATTRIBUTES property and other properties in the WHCOLTIM type. For example, you can use an add-in tool to add data mining attributes to a _LOADTM column, export the metadata for the table to Enterprise Miner and analyze the _LOADTM column in Enterprise Miner.
- The usage notes for the _UPDATE_METDATA_ method have been expanded. For details, see "Using _UPDATE_METADATA_" on page 46.

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 that use 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 later
- SAS/AF software
- SAS/GRAPH software—if you need to modify or write API software that includes a GUI
- the SAS application whose metadata you want to read or write, such as SAS/Warehouse Administrator, Release 1.2 or later.

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

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
specifies 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
specifies 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 the SAS/Warehouse Administrator Metadata API?

It is a set of software tools that enable programmers to write applications that access metadata in SAS/Warehouse Administrator.

What Can I Do with the SAS/Warehouse Administrator Metadata API?

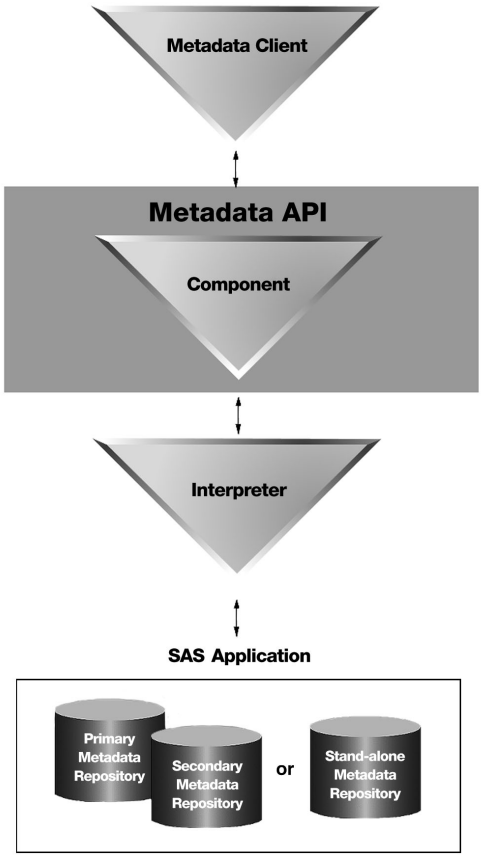
Using the metadata API, you can write programs that read, add, update, or delete the metadata in SAS/Warehouse Administrator—without going through the user interface. You can write SCL applications that

- publish HTML pages that contain the current metadata for a SAS/Warehouse Administrator group or data store
- 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 that are 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 6 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 accommodates multiple components as long as each component has an appropriate interpreter. △

metadata client
specifies 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
specifies a set of software tools that enables users to write applications that access metadata.

metadata type
represents 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 that are maintained for the corresponding object.

SAS/Warehouse Administrator metadata types are listed in "Index to SAS/Warehouse Administrator Metadata Types" on page 70.

component
specifies a group of related metadata types. Each component has an ID (such as WHOUSE) and a name (such as SAS/Warehouse Administrator) that often match the name of the application whose metadata is modeled by the component. The component that is supplied with the current API is WHOUSE (SAS/Warehouse Administrator).

application program interface (API) interpreter
represents a program that translates the API metadata type that is 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 the Job Scheduler utility.

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 and not the client applications that use the metadata API.

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

metadata repository
specifies 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 10.

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 can have 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]
            ) [671]
```

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 16.) The following properties are common to all metadata types. They are often referred to as the *general identifying information* for a metadata object.

ID
specifies the unique three-level identifier for a metadata object. It takes the following form: *reposid.typeid.instanceid*. For example, in the previous code example, the ID for the COLUMNS object is A000000E.WHCOLDTL.A0000032.

A000000E is the repository ID that is 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 that identifies the metadata repository that stores the object. Each application has one or more repositories.

WHCOLDTL is the type ID for a column in a SAS/Warehouse Administrator detail table. A *typeid* (metadata type ID) is a maximum 8-character string that 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 70.

A0000032 is the instance ID that is 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 that distinguishes one metadata object from all other objects of the same type within a given repository.

NAME
specifies 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
describes 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 that is created by SAS/Warehouse Administrator. For the sake of simplicity, assume that you have already attached to the relevant metadata repositories, that the metadata that you want is in the A000000E repository, and that the type ID 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 that is 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_ in order 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,'Y');
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 that are similar to those used in these steps.

Metadata Repositories

You can divide an application's metadata into different physical stores based on the following 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).

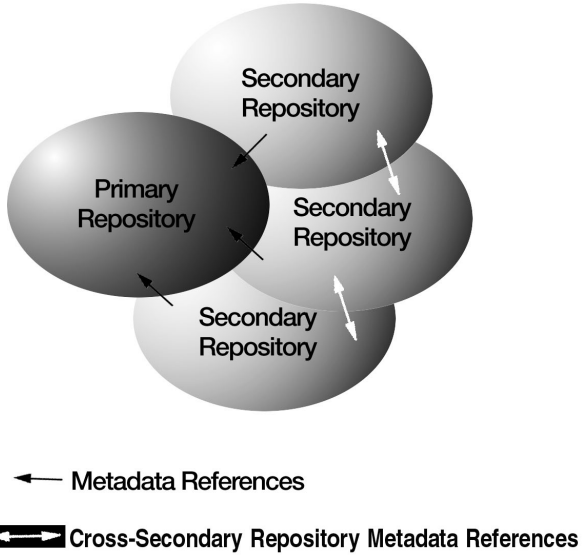
Each physical store of metadata is called a *metadata repository*. There are two main types of metadata repositories—*stand-alone* and *partitioned*.

A stand-alone repository is a single metadata store, such as a SAS/EIS respository. Once you access a stand-alone repository, all metadata is accessible. Figure 1.2 on page 10 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 11 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 you can access 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 that is 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 can be associated with metadata in other repositories. The secondary repositories can 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 11). Some partitioned repositories also support secondary repositories that contain metadata references into other secondary repositories, which are 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 that 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 that 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 (with the SEARCH_SECONDARY parameter).

This method returns a list of all metadata objects of a given type. The *reposid* for each object identifies the repository where the object is stored.

Learning to Use the Metadata API

The following 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 273 and the "Write Metadata Code Sample" on page 277.
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. Because 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 by using the code samples in Appendix 1, "Sample Metadata API Code," on page 273 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 that you want.

      For example, if you want to list all of the owners that are 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 53.
   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

This document uses the following conventions in the examples:

- 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.