SAS Publishing

SAS/Warehouse Administrator® 2.3 Metadata API

Reference
Second Edition

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS / Warehouse Administrator® 2.3 Metadata API Reference, Second Edition*. Cary, NC: SAS Institute Inc.

SAS/Warehouse Administrator® 2.3 Metadata API Reference, Second Edition

Copyright © 2004, SAS Institute Inc., Cary, NC, USA

ISBN 1-59047-222-5

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

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.

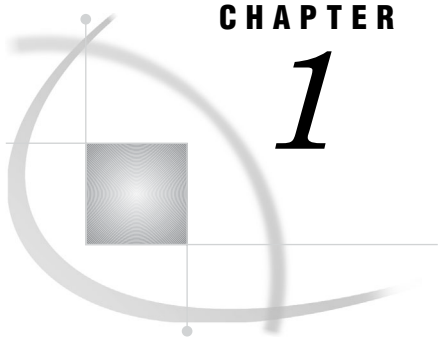
1st printing, January 2004

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/pubs** 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



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?

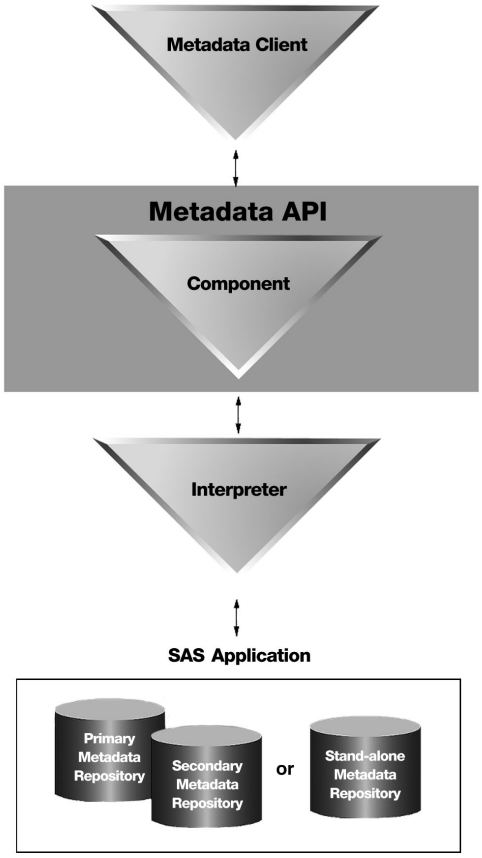
Using the metadata API, you can write programs that read, add, update, or delete the metadata in SAS/Warehouse Administrator—without going through the user interface. You can write SCL applications that

- 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).

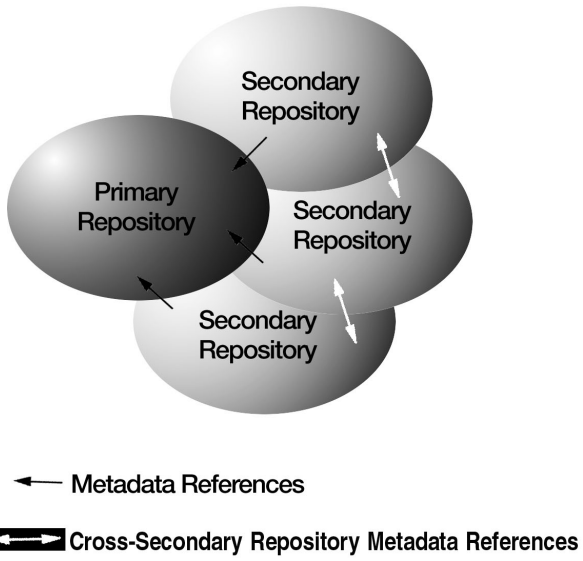
Each physical store of metadata is called a *metadata repository*. There are two main types of metadata repositories—*stand-alone* and *partitioned*.

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.

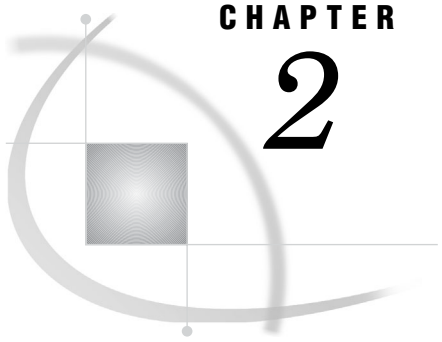
Naming Conventions Used in This Manual

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.



CHAPTER

2

Metadata API Class

Overview of the Metadata API Class

The metadata API class defines a set of methods that read and write metadata types. A metadata client application uses these methods to communicate with an API interpreter. The API interpreter translates the metadata types that are requested by the client to the corresponding metadata in a SAS application's metadata repository.

Parent:
SASHELP.FSP.OBJECT.CLASS

Class:
SASHELP.METAAPI.METAAPI.CLASS

Using the Metadata API Class

Using the metadata API class primarily involves using its methods. To access these methods, instantiate a metadata API object using the INSTANCE and LOADCLASS facilities.

```
i_api=instance(loadclass
('SASHELP.METAAPI.METAAPI.CLASS'));
```

Introduction to Metadata API Methods

Methods that are specific to the metadata API class are described here.

Conventions

All lists and items in those lists that are passed to the API must have the UPDATE list attribute. This applies to both the read and write metadata methods.

Whenever an output list is returned, a list will be created for you if one is not passed. If one is passed, then the output information will be appended to the end of the existing list.

Error Codes

Metadata API methods return error codes in the *l_rc* parameter. If a method returns a nonzero *l_rc*, then the method failed, and *l_rc* is an error list identifier. It is your responsibility as the application programmer to delete this list after interrogating its contents (using PUTLIST, for example). The *l_rc* error list can contain the following named items:

RC
: represents the numeric return code value.

MSG
: specifies an optional error message that indicates the type of failure that occurred. The returned string can be a system message or a string that is generated by the API or API interpreters.

Metadata Property List

To read or write a metadata object, you must pass a list of properties for that object to the appropriate metadata API method. Typically, the metadata property list that you pass to a method includes an ID—the unique identifier for a particular metadata object. The list might also include the NAME and DESC properties.

The ID, NAME, and DESC properties are common to all metadata types. In this manual, these properties are often referred to as the general identifying information for a metadata object. For a description of the ID, NAME, and DESC properties, see “Identifying Metadata” on page 7.

A metadata property list is not limited to the ID, NAME, and DESC properties. If you are interested in other properties for a given metadata type, you can often pass those properties as named sublists. The following code sample shows how to use the _GET_METADATA_ method to return the LIBRARY, COLUMNS, and TABLE NAME properties for a detail table:

```
id='A000000E.WHDETAIL.A000002X';
l_meta=clearlist(l_meta,'Y');
l_meta=insertc(l_meta,id,-1,'ID');

   /*
    * Retrieve library, column, and table name
    * properties only.
    */

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);

     /* returns list: */
L_META(
   ID='A000000E.WHDETAIL.A000002X'
   LIBRARY=(
      ID='A0000001.WHLIBRY.A000000U'
      NAME='Warehouse Data Library'
      DESC=''
   )[5]
   COLUMNS=(
      (   ID='A000000E.WHCOLDTL.A0000032'
          NAME='PRODNUM'
          DESC='product number'
       )[9]
      (   ID='A000000E.WHCOLDTL.A0000034'
          NAME='PRODNAME'
          DESC='product name'
       )[11]
      (   ID='A000000E.WHCOLDTL.A0000036'
          NAME='PRODID'
          DESC='product id/abbreviation'
       )[13]
      (   ID='A000000E.WHCOLTIM.A00000FU'
          NAME='_LOADTM'
          DESC='DateTime Stamp of when row
                was loaded'
       )[15]
   )[7]
   TABLE NAME='PRODUCT'
)[3]
```

Not all properties are valid for a given method. To understand which properties for a given type are valid with a given method, see the documentation for each type.

Index to Metadata API Methods

In the method dictionary, metadata API methods are described in alphabetical order. In this section, these methods are listed by category.

Table 2.1 Metadata API Methods

Category	Metadata API Class	Description
Management Methods	"_GET_COMPONENTS_" on page 22	Lists all components that are defined in the metadata API
	"_GET_SUBTYPES_" on page 30	Returns all possible subtypes for a specified metadata type
	"_GET_TYPES_" on page 33	Lists metadata types in the metadata API
	"_GET_TYPE_NAME_" on page 35	Returns metadata type name when passed a type ID
	"_GET_TYPE_PROPERTIES_" on page 36	Returns all possible properties for a metadata type
	"_IS_SUBTYPE_OF_" on page 38	Determines if one metadata type is a subtype of another
Navigation Method	"_GET_METADATA_OBJECTS_" on page 28	Lists metadata objects when passed a repository and type
Read Method	"_GET_METADATA_" on page 25	Reads specified metadata from a repository
Repository Methods	"_CLEAR_SECONDARY_REPOSITORY_" on page 19	Detaches from a secondary repository
	"_GET_CURRENT_REPOSITORIES_" on page 23	Lists all currently active primary metadata repositories
	"_SET_PRIMARY_REPOSITORY_" on page 40	Attaches to a primary metadata repository
	"_SET_SECONDARY_REPOSITORY_" on page 43	Attaches to a secondary metadata repository
Write Methods	"_ADD_METADATA_" on page 16	Adds specified metadata in a repository
	"_DELETE_METADATA_" on page 20	Deletes specified metadata from a repository
	"_UPDATE_METADATA_" on page 46	Updates specified metadata in a repository

_ADD_METADATA_

Adds specified metadata in a repository

Category: Write Methods

Syntax

CALL SEND(*i_api*, '_ADD_METADATA_', *l_rc*, *l_meta*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_meta	L	specifies the passed metadata property list for the object that is to be added. For the general format of this list, see "Metadata Property List" on page 14.

Details

l_meta

specifies the passed metadata property list for the object that is to be added.

To create a new instance of a particular type, the ID value in *l_meta* should be *resposid.typeid*. If an instance ID is passed, it is ignored and replaced with a new instance ID upon successful addition to the repository.

Not all metadata types (type IDs) can be added. The documentation for each metadata type indicates whether it can be added or not. _ADD_METADATA_ will return an error of any type that cannot be added.

Using _ADD_METADATA_

Be sure to check the return code of a write method call. A nonzero return indicates that a failure has occurred while trying to write to the metadata. If a nonzero return code is returned, none of the changes that are indicated by this method call will be made.

Example: Add a New Detail Table

```
l_meta=makelist();

   /*
    *  Set which group to add this new table to.
    */

l_groups=makelist();
l_group=makelist();

l_groups=insertl(l_groups,l_group,-1);

l_group=insertc(l_group,group_id,-1,'ID');

l_meta=insertl(l_meta,l_groups,-1,'GROUP');

   /*
    *  Use the same repository id as the group.
    */
```

```
repos_id=scan(group_id,1,'.');

new_type=repos_id||'.WHDETAIL';

l_meta=insertc(l_meta,new_type,-1,'ID');

   /*
    *  Set the name for the display.
    */

l_meta=insertc(l_meta,
 'NEW TABLE',-1,'NAME');

   /*
    *  Set the desc for the display.
    */

l_meta=insertc(l_meta,'New table added
through API',-1,'DESC');

   /*
    *  Set an icon for the display.
    */

l_meta=insertc(l_meta,
'SASHELP.I0808.ADD.IMAGE',-1,'ICON');
   /*
    *  Define a column.  The COLUMNS property
    * contains a sublist per column.
    */

l_cols=makelist();
l_col=makelist();

l_cols=insertl(l_cols,l_col,-1);

l_meta=insertl(l_meta,l_cols,-1,'COLUMNS');

col_id=repos_id||'.'||'WHCOLUMN';

l_col=insertc(l_col,col_id,-1,'ID');
l_col=insertc(l_col,'CUSTOMER',-1,'NAME');
l_col=insertc(l_col,'Name of Customer',-1,
      'DESC');
l_col=insertc(l_col,'C',-1,'TYPE');
l_col=insertn(l_col,75,-1,'LENGTH');


   /*
    *  Add any additional properties
    *  :
    *  :
    */
```

```
   /*
    *  Add the table.
    */

call send(i_api,'_ADD_METADATA_',l_rc,l_meta);

if l_rc = 0 then do;

   put 'Table Added successfully';

   end; /* if */
else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
  _ADD_METADATA_ FAILED');
   put msg;

   list_rc=dellist(l_rc);

   end; /* else */


l_meta=dellist(l_meta,'Y');
```

See Also

_DELETE_METADATA_, _UPDATE_METADATA_

_CLEAR_SECONDARY_REPOSITORY_

Detaches from a secondary repository

Category: Repository Methods

Syntax

CALL SEND(*i_api*, '_CLEAR_SECONDARY_REPOSITORY_', *l_rc*, *repos_id*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See “Using the Metadata API Class” on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see “Error Codes” on page 14.
repos_id	C	specifies the passed repository ID that specifies the repository that is to be detached from. For details about the *repos_id* parameter, see “Identifying Metadata” on page 7.

Using _CLEAR_SECONDARY_REPOSITORY_

When you only want to be attached to the primary repository, use the _CLEAR_SECONDARY_REPOSITORY_ method to detach from any secondary repositories.

Use the _GET_METADATA_ method to return the list of possible secondary repositories. Specify the REPOSITORIES property in the *l_meta* list, and use the returned metadata identifier from the _SET_PRIMARY_REPOSITORY_ method. See the code examples under _SET_PRIMARY_REPOSITORY_ and _SET_SECONDARY_REPOSITORY_ .

Example: Detach from a Secondary Repository

```
     /* sec_repos_id is the REPOSID of the secondary repository that is
      * to be detached from.
      */
call send(i_api, '_SET_SECONDARY_REPOSITORY_', l_rc, sec_repos_id);
```

See Also

_GET_METADATA_
_SET_SECONDARY_REPOSITORY_
_SET_PRIMARY_REPOSITORY_

_DELETE_METADATA_

Deletes specified metadata from a repository

Category: Write Methods

Syntax

CALL SEND(*i_api*, '_DELETE_METADATA_', *l_rc*, *l_meta*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_meta	L	specifies the passed metadata property list for the object that is to be deleted. For the general format of this list, see "Metadata Property List" on page 14.

Using _DELETE_METADATA

The object whose ID is included in the *l_meta* list will be deleted. Where appropriate, the metadata API will enforce metadata integrity by deleting all other metadata that is associated with the object that is being deleted.

CAUTION:

The _DELETE_METADATA_ method is destructive. Its changes cannot be reversed. When you use this method in an application, verify the delete request before you issue the method call. △

Be sure to check the return code of a write method call. A nonzero return indicates that a failure has occurred while trying to write to the metadata. If a nonzero return code is returned, none of the changes that are indicated by this method call will be made.

Example: Delete Column Definitions

```
    /*
     *  Delete all the current column
     *  definitions for the passed id.
     */

l_meta=makelist();

l_meta=insertc(l_meta,selected_id,-1,'ID');

   /*
    *  Get all of the columns.
    */

l_meta=insertl(l_meta,0,-1,'COLUMNS');

call send(i_api,'_GET_METADATA_',l_rc,l_meta);

   /*
    *  Continue if zero return code
    *  (removed for brevity of example)
    *  :
```

```
    */

l_cols=getniteml(l_meta,'COLUMNS');

num_cols=listlen(l_cols);

do i=1 to num_cols while (l_rc = 0);


   l_col=getiteml(l_cols,i);

      /*
       *  Delete each column.
       */

   call send(i_api,'_DELETE_METADATA_',l_rc,l_col);

   if l_rc ne 0 then do;

      msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
          _DELETE_METADATA_ FAILED');
      put msg;

      list_rc=dellist(l_rc);

      end; /* if */

   end; /* do */

l_meta=dellist(l_meta,'Y');
```

See Also

_ADD_METADATA_, _UPDATE_METADATA_

_GET_COMPONENTS_

Lists all components that are defined in the metadata API

Category: Management Methods

Syntax

CALL SEND(*i_api*, '_GET_COMPONENTS_', *l_rc*, *l_components*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of the METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Conventions" on page 14.
l_components	L	specifies the returned list of all components that are defined in the metadata API. List format: *comp_id=comp_name*. Components are discussed in "How the Metadata API Works" on page 5.

Using _GET_COMPONENTS_

A *component* is a group of related metadata types. One use for the _GET_COMPONENTS_ method is to get a component ID that you can pass to the _GET_TYPES_ method in order to list the metadata types for a particular component.

Example: List All Components Defined for the Metadata API

```
call send(i_api,'_GET_COMPONENTS_',l_rc,
l_components);

   /* A list of components is returned. */
l_components(
   WHOUSE=SAS/Warehouse Administrator
)[3]
```

See Also

_GET_TYPES_

_GET_CURRENT_REPOSITORIES_

Lists all currently active primary metadata repositories

Category: Repository Methods

Syntax

CALL SEND(*i_api*, '_GET_CURRENT_REPOSITORIES_', *l_rc*, *l_reps*, <*type*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of the METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_reps	L	specifies the returned list of repository IDs for the currently active primary metadata repositories. For details about repository IDs, see "Identifying Metadata" on page 7.
type	C	specifies the passed repository type ID. Optional. Limits the *l_reps* list to primary repositories of this type.

Details

l_reps
specifies the returned list of currently active primary metadata repositories, in the format:

```
l_reps=(type=repository id
        type=repository id
        )
```

For details about primary and secondary repositories, see "Metadata Repositories" on page 10.

type
specifies the passed repository type ID. Optional. Limits the *l_reps* list to primary repositories of this type. If a type ID is not passed, all primary repositories will be returned (a value of _ALL_) is passed for *type*).

Each component has one or more metadata repository types. See the metadata type documentation for a particular component for details. For example, for SAS/Warehouse Administrator metadata repository types, see Appendix 1, "Sample Metadata API Code," on page 273.

Using _GET_CURRENT_REPOSITORIES_

To return the list of active secondary repositories, use the _GET_METADATA_ method with the appropriate primary repository ID from the returned *l_reps* list. See the Usage notes under _GET_METADATA_ for details.

See Also

_GET_METADATA_

_GET_METADATA_

Reads specified metadata from a repository

Category: Read Method

Syntax

CALL SEND(*i_api*, '_GET_METADATA_', *l_rc*, *l_meta*, <*all*>, <*expand*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_meta	L	specifies the passed metadata property list for the object that is to be read. For the general format of this list, see "Metadata Property List" on page 14.
all	N	specifies the passed indicator. Optional. Specifies whether the method should get all associated metadata for the object.
expand	N	specifies the passed indicator. Optional. Specifies whether references to dependent objects should be expanded.

Details

l_meta

specifies the passed metadata property list for the object that is to be read. At a minimum, you must supply a fully qualified ID in the *l_meta* list: *reposid.typeid.instanceid*.

You can supply a formatted *l_meta* list as input to the _GET_METADATA_ call, as shown in Example 1. In this case, only the sublists (properties) whose names have been passed in the formatted list will be returned. This will allow for selective retrieval of pieces of metadata about an object.

Alternatively, you could pass an *l_meta* list with only the ID property filled in and supply the *all* parameter to indicate to return all information about the requested object, as shown in Example 2. Getting all properties could take much longer than getting a select few.

all

specifies the passed indicator. Optional. Specifies whether the method should get all associated metadata for the object.

0 — (default) return only the metadata that is specified in *l_meta*.

1 — return all information known about the object that is specified in *l_meta*. However, if a sublist is returned that references another object, only the general identifying information for the referenced object will be returned.

Note that it takes longer to return a query if you ask for more information.

expand
specifies the passed indicator. Optional. Specifies that any references to dependent objects should be expanded to include all properties for the referenced object (not only its general identifying information). For an explanation of dependent objects, see "Independent and Dependent Metadata Objects" on page 53.
0 — (default) return all property lists unexpanded.
1 — expand all dependent object references.
Note that it takes longer to return a query if you ask for more information.

Note: To understand which properties of a given metadata type will be expanded, see the property tables for each type in "Using the Metadata Type Dictionary" on page 73. △

Using _GET_METADATA_

It is possible that a sublist that is returned might contain identifiers of different types of objects, each with its own properties list format. Use the _IS_SUBTYPE_OF_ method to determine the type of the metadata identifier and thus the appropriate properties list format.

CAUTION:
The performance of this method is directly related to the number and content of the properties that are requested. The *all* and *expand* parameters can have an adverse effect on the performance of this method and should be used accordingly. △

In addition to reading metadata objects in a repository, you can use _GET_METADATA_ to return a list of secondary metadata repositories. Specify the REPOSITORIES property in the *l_meta* list, and use the returned metadata identifier from the _SET_PRIMARY_REPOSITORY_ method. See "Example: Set a Secondary Repository" on page 45 and Example 2 on page 42.

Examples

Example 1: Return Table Information

```
id='A000000E.WHDETAIL.A000002X';
l_meta=makelist();
l_meta=insertc(l_meta,id,-1,'ID');

    /*
     *  For now, retrieve only table properties.
     */
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);

   /* returns list: */
L_META(
   ID='A000000E.WHDETAIL.A000002X'
   LIBRARY=(
      ID='A0000001.WHLIBRY.A000000U'
      NAME='Warehouse Data Library'
      DESC=''
```

```
   )[5]
   COLUMNS=(
      (   ID='A000000E.WHCOLDTL.A0000032'
          NAME='PRODNUM'
          DESC='product number'
       )[9]
      (   ID='A000000E.WHCOLDTL.A0000034'
          NAME='PRODNAME'
          DESC='product name'
       )[11]
      (   ID='A000000E.WHCOLDTL.A0000036'
          NAME='PRODID'
          DESC='product id/abbreviation'
       )[13]
      (   ID='A000000E.WHCOLTIM.A00000FU'
          NAME='_LOADTM'
          DESC='DateTime Stamp of when row
               was loaded'
       )[15]
   )[7]
   TABLE NAME='PRODUCT'
)[3]
```

Example 2: Return Information about One Column

```
l_cols=getniteml(l_meta,'COLUMNS');
   l_col=getiteml(l_cols,4);
      /*
       * Get all information about column
       * (note get_all=1 parameter)
       */
    call send(i_api,'_GET_METADATA_',l_rc,l_col,1);

   /* returns list: */
   L_COL=(
   ID='A000000E.WHCOLTIM.A00000FU'
   DESC='DateTime Stamp of when row was loaded'
   NOTE=()[4083]
   INDEXES=()[4085]
   INPUT OBJECTS=()[4087]
   OUTPUT OBJECTS=()[4089]
   EXTENDED ATTRIBUTES=()[4096]
   TABLE=(
         ID='A000000E.WHDETAIL.A000002X'
         NAME='Product detail table'
         DESC='Contains information about all
                products'
         )[4091]
   FORMAT='DATETIME.'
   INFORMAT='DATETIME.'
   INPUT SOURCES=()[4093]
   LENGTH=8
   OUTPUT TARGETS=()[4095]
   TYPE='N'
   CVALUE=''
```

```
METADATA CREATED='  04MAR1997:15:29:29'
METADATA UPDATED='  04MAR1997:15:29:29'
NAME='_LOADTM'
NVALUE=.
)[3719]
```

_GET_METADATA_OBJECTS_

Lists metadata objects when it is passed repository and type

Category: Navigation Method

Syntax

CALL SEND(*i_api*, '_GET_METADATA_OBJECTS_', *l_rc*, *reposid.typeid*, *l_objs*, <*search_secondary*>, <*include_subtypes*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
reposid.typeid	C	specifies the passed *reposid* and *typeid* of metadata that is to be returned. For the general format of these IDs, see "Identifying Metadata" on page 7.
l_objs	L	spcifies the returned list of metadata objects. List format: *id=name*.
search_ secondary	N	specifies the passed indicator. Optional. Specifies whether the returned list should include any objects of the specified type from all active secondary repositories.
include_subtypes	N	specifies the passed indicator. Optional. Specifies whether the returned list should include subtypes.

Details

search_secondary
specifies the passed indicator. Optional. Specifies whether the returned list should include any objects of the specified type from all active secondary repositories.
0 — return objects from the passed repository only.

1 — (default) if passed repository ID is that of the primary repository, then return all from the secondary repository, too. If passed repository ID is that of a secondary repository, this parameter is ignored.

include_subtypes
specifies the passed indicator. Optional. Specifies whether the returned list should include subtypes.
0 — return objects of the specified type only.
1 — (default) return objects of the specified type and their subtypes, if any.

Examples

Example 1: Returning a List of Entries for a Specific Type (Detail Tables)

```
type=dw_repos_id||'.WHDETAIL';
call send(i_api,'_GET_METADATA_OBJECTS_',l_rc,
type,l_objs);

   /* returns list: */

L_objs(
   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'

)[3]
```

Example 2: Get All Tables Registered in Primary and Secondary Repositories

```
   /*
    * Get all Tables registered in Primary and
    * Secondary repositories
    * Primary Repos ID = A0000001
    * Secondary Repos ID = A0000003
    */

type=primary_repos_id||'.WHTABLE';

l_objs=makelist();

call send(i_api, '_GET_METADATA_OBJECTS_', l_rc, type, 1, 1);

num_objs=listlen(l_objs);

if l_rc = 0 then do;
```

```
   call putlist(l_objs);

   end; /* if */

else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
   _GET_METADATA_OBJECTS_ Failed.');
   list_rc=dellist(l_rc);

   end; /* else */

l_objs=dellist(l_objs);


/* returns list */

L_OBJS(
  A0000003.WHDETAIL.A0000069='Product detail table'
  A0000003.WHDETAIL.A00000QI='Sales fact table'
  A0000003.WHLDETL.A000004R='Customer detail'
  A0000003.WHSUMTBL.A00000TG='Monthly summary'
  A0000001.WHODDTBL.A0000049='Services'
  A0000001.WHODDTBL.A00000FP='Payment File'

 )
```

_GET_SUBTYPES_

Returns all possible subtypes for a specified metadata type

Category: Management Methods

Syntax

CALL SEND(*i_api*, '_GET_SUBTYPES_', *l_rc*, *type_id*, *l_types*, <*expand*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See “Using the Metadata API Class” on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see “Error Codes” on page 14.
type_id	C	specifies the passed type ID of metadata to be returned. For the general format of this ID, see “Identifying Metadata” on page 7.
l_types	L	specifies the returned list of all possible subtypes for the specified type. List format: *type_id=type_name*.
expand	N	specifies the passed indicator. Optional. Specifies whether returned list will expand all possible subtypes of all subtype branches.

Details

expand
specifies the passed indicator. Optional. Specifies whether returned list will expand all possible subtypes of all subtype branches.
0 — (default) return all subtypes unexpanded.
1 — expand all possible subtypes of all subtype branches.

Examples

Example 1: Get Subtypes—Unexpanded

```
    /*
     *  Get all Immediate Subtypes of Tables
     */
l_types=makelist();

call send(i_api,'_GET_SUBTYPES_',l_rc,'WHTABLE',l_types,0);

num_types=listlen(l_types);

if l_rc = 0 then do;

   call putlist(l_types);

   end; /* if */

else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
    _GET_SUBTYPES_ Failed.');
   list_rc=dellist(l_rc);

   end; /* else */
```

```
l_types=dellist(l_types);

    /* returns list */
 l_types(
  WHDETAIL='Detail Table'
  WHLDETL='Detail Logical Table'
  WHSUMTBL='Summary Table'
  WHODDTBL='Operational Data Definition'
  WHODTTBL='Operational Data Table'
  WHTBLPRC='Process Output Tables'
  WHDATTBL='Data Table'
  WHOLPSTR='OLAP Structure'
  WHTBLPRC='Process Output Table' )[47]
```

Example 2: Get Subtypes—Expanded

```
    /*
     *  Get all Subtypes of Tables
     */
l_types=makelist();

call send(i_api,'_GET_SUBTYPES_',l_rc,'WHTABLE',l_types,1);

num_types=listlen(l_types);

if l_rc = 0 then do;

   call putlist(l_types);

   end; /* if */

else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
   _GET_SUBTYPES_ Failed.');
   list_rc=dellist(l_rc);

   end; /* else */

l_types=dellist(l_types);

    /* returns list */

 L_types(
 WHDETAIL='Detail Table'
 WHLDETL='Detail Logical Table'
 WHSUMTBL='Summary Table'
 WHODDTBL='Operational Data Definition'
 WHODTTBL='Operational Data Table'
 WHTBLPRC='Process Output Tables'
 WHDATTBL='Data Table'
 WHTBLMAP='Mapping Process Table'
 WHTBLUSR='User Exit Process Table'
```

```
WHTBLXFR='Data Transfer Process Table'
WHTBLREC='Record Selector Process Table'
WHOLPSTR='OLAP Structure'
WHGRPOLP='OLAP Group'
WHOLPTBL='OLAP Table'
WHOLPMDD='OLAP MDDB'
)[47]
```

_GET_TYPES_

Lists metadata types that are in the metadata API

Category: Management Methods

Syntax

CALL SEND(*i_api*, '_GET_TYPES_', *l_rc*, *l_types*, <*component*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_types	L	specifies the returned list of metadata types. List format: *type_id=type_name*.
component	C	specifies the passed component name. Optional. Can limit the returned list to the metadata types within a specific component.

Details

component
specifies the passed component name, such as WHOUSE for the SAS/Warehouse Administrator component. Optional. Can limit the returned list to the metadata types within a specific component. The default is _ALL_ (all components).

Components are discussed in "How the Metadata API Works" on page 5. Use _GET_COMPONENTS_ to return a list of components that are available at your site.

Example: Get All Types for the WHOUSE Component

```
l_types=makelist();
```

```
call send(i_api,'_GET_TYPES_',l_rc,l_types,'WHOUSE');

num_types=listlen(l_types);

if l_rc = 0 then do;

   call putlist(l_types);

   end; /* if */

else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
   _GET_TYPES_ Failed.');
   list_rc=dellist(l_rc);

   end; /* else */

l_types=dellist(l_types);

   /* returns list - types
    * (removed for brevity of example)
    */

 l_types(
 (WHROOT='Warehouse Root Metadata Type'
 WHDWENV='Warehouse Environment'
 WHDW='Data Warehouse'
 WHTFILE='Text File'
 WHTXTCAT='Catalog Text File'
 WHTXTFIL='External Text File'
 WHSCRFIL='SAS/Connect Script File'
 WHEFILE='External File'
 WHTABLE='Table'
 WHDETAIL='Detail Table'
 WHLDETL='Detail Logical Table'
 :
 WHTBLPRC='Process Output Tables'
 WHTBLMAP='Mapping Process Table'
 WHTBLUSR='User Exit Process Table'
 :

 ) [47]
```

See Also

_GET_COMPONENTS_

_GET_TYPE_NAME_

Returns metadata type name when it is passed a type ID

Category: Management Methods

Syntax

CALL SEND(*i_api*, '_GET_TYPE_NAME_', *l_rc*, *type_id*, *type_name*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
type_id	C	specifies the passed type ID of the metadata type for which you want the name. For more information on type ID, see "Identifying Metadata" on page 7.
type_name	C	specifies the returned name of the specified metadata type.

Details

type_id
specifies the passed type ID of the metadata type for which you want the name. For example, SAS/Warehouse Administrator has the WHDETAIL type. If an invalid type ID is passed, the returned type name is set to blank, and a nonzero return code is returned.

type_name
specifies the returned name of the specified metadata type. For example, if you pass the type ID WHDETAIL, the name "Detail Table" would be returned. A blank value is returned if an invalid metadata type name is supplied.

Example: Get Type Name for WHDETAIL

```
type_id='WHDETAIL';
type_name = _blank_;

call send(i_api,'_GET_TYPE_NAME_',
l_rc,type_id,type_name);

if l_rc = 0 then do;

   put type_id= type_name=;
```

```
   end; /* if */

else do;

   msg=getnitemc(l_rc,'MSG',1,1,
   'ERROR: _GET_TYPE_NAME_ Failed.');
   list_rc=dellist(l_rc);

   /* Output in log */
WHDETAIL=Detail Table
```

_GET_TYPE_PROPERTIES_

Returns all possible properties for a metadata type

Category: Management Methods

Syntax

CALL SEND(*i_api*, '_GET_TYPE_PROPERTIES_', *l_rc*, *type_id*, *l_props*, <format>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
type_id	C	specifies the passed type ID of the metadata type for which you want a list of properties. For more information on type IDs, see "Identifying Metadata" on page 7.
l_props	L	specifies the returned list of all possible properties for the specified metadata type.
format	C	specifies the passed indicator. Optional. Specifies the format of the returned property list.

Details

type_id
specifies the passed type ID of the metadata type for which you want a list of properties. For example, SAS/Warehouse Administrator has the WHDETAIL type. If an invalid type ID is passed, the returned list is blank.

l_props
specifies the returned list of all possible properties for the specified metadata type. The returned list includes all possible properties. Some properties might not be populated in a given instance of this type.

format
specifies the passed indicator. Optional. Specifies the format of the returned property list.

S — (default) returns the property list in skeleton format; the property names are item names in the list. This format is suitable for passing to the _GET_METADATA_ method. See Example 1.

D — returns the property list in display format; the property names are character values in the list. This format is suitable for display to the user. See Example 2.

Examples

Example 1: Return Skeleton Properties List for a Given Type ID

```
call send(i_api, _GET_TYPE_PROPERTIES_', l_rc, 'WHDETAIL', l_props2,'S');

/*    Returns list: */
     l_props2=(
      (DESC=' '
       NOTE=()[291]
       ADMINISTRATOR=()[292]
       GROUP=()[293]
       MEMBERS=()[294]
       OWNER=()[295]
       COLUMNS=()[296]
       HOST=()[297]
       INPUT OBJECTS=()[298]
       INPUT SOURCES=()[299]
       LIBRARY=()[300]
       OUTPUT OBJECTS=()[301]
       OUTPUT TARGETS=()[302]
       PHYSICAL STORAGE=()[303]
       PROCESS=()[304]
       EXTENDED ATTRIBUTES=()[305]
       ACCESS SAME AS PHYSICAL=.
       CREATING JOB=()[306]
       TABLE NAME=' '
       USING JOBS=()[307]
       ICON=' '
       CVALUE=' '
       ID=' '
       METADATA CREATED=' '
       METADATA UPDATED=' '
       NAME=' '
       NVALUE=.
      )[290]
```

Example 2: Return List of Property Names for a Given Type ID

```
call send(i_api, _GET_TYPE_PROPERTIES_', l_rc,'WHDETAIL', l_props2,'D');
```

```
    /*    Returns list: */
  L_PROPS2=(
    ('DESC'
     'NOTE'
     'ADMINISTRATOR'
     'GROUP'
     'MEMBERS'
     'OWNER'
     'COLUMNS'
     'HOST'
     'INPUT OBJECTS'
     'INPUT SOURCES'
     'LIBRARY'
     'OUTPUT OBJECTS'
     'OUTPUT TARGETS'
     'PHYSICAL STORAGE'
     'PROCESS'
     'EXTENDED ATTRIBUTES'
     'ACCESS SAME AS PHYSICAL'
     'CREATING JOB'
     'TABLE NAME'
     'USING JOBS'
     'ICON'
     'CVALUE'
     'ID'
     'METADATA CREATED'
     'METADATA UPDATED'
     'NAME'sas
     'NVALUE'
   )[290]
```

_IS_SUBTYPE_OF_

Determines if one metadata type is a subtype of another

Category: Management Methods

Syntax

CALL SEND(*i_api*, '_IS_SUBTYPE_OF_', *l_rc*, *type_id*, *super_type_id result*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.

Parameter	Type	Description
type_id	C	specifies the passed type ID of the metadata type that might be a subtype of *super_type_id*. Type IDs are discussed in "Identifying Metadata" on page 7.
super_type_id	C	specifies the passed type ID of the metadata type that might be a supertype of *type_id*.
result	C	specifies the returned indicator. Indicates whether *type_id* is a subtype of *super_type_id*.

Details

result
specifies the returned indicator. Indicates whether *type_id* is a subtype of *super_type_id*.
0 — *type_id* is not a subtype of *super_type_id*.
1 — *type_id* is a subtype of *super_type_id*.

Example: Is WHOLPTBL a Subtype of WHTABLE?

```
type_id='WHOLPTBL';
super_type_id='WHTABLE';

call send(i_api,'_IS_SUBTYPE_OF_',l_rc,type_id,
    super_type_id,a_subtype);

if l_rc = 0 then do;

   if a_subtype then
           put type_id 'is a subtype of ' super_type_id;
   else
           put type_id 'is not a subtype of 'super_type_id;

   end; /* if */

else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
       _IS_SUBTYPE_OF_ Failed.');
   list_rc=dellist(l_rc);

   end; /* else */
call send(i_api, '_GET_METADATA_OBJECTS_', l_rc, type, 1, 1);

   /*    Output to Log: */
WHOLPTBL is a subtype of WHTABLE
```

_SET_PRIMARY_REPOSITORY_

Attaches to a primary metadata repository

Category: Repository Methods

Syntax

CALL SEND(*i_api*, '_SET_PRIMARY_REPOSITORY_', *l_rc*, *l_meta*, *repos_type*, *repos_id*, *l_meta2*, <*already*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_meta	L	specifies the passed metadata property list for the primary repository that is to be set.
repos_type	C	specifies the passed repository type ID of the primary metadata repository that is to be set.
repos_id	C	specifies the returned repository ID for the primary metadata repository that was set. For details about the repository ID parameter, see "Identifying Metadata" on page 7.
l_meta2	L	specifies the returned metadata property list for the repository that was set.
already	N	specifies the passed indicator. Optional. Specifies whether the repository to be set is already accessible.

Details

l_rc
specifies the return codes for the method.

Note: Be sure to check the return code for this method. Do not continue if the method fails. △

l_meta
specifies the passed metadata property list for the primary repository that is to be set.

From the list of all possible properties for the metadata repository type that is specified in the *repos_type* parameter, you need only to pass those properties that are required to allocate a SAS libname for the repository. See the following Usage notes.

repos_type
specifies the passed repository type ID of the primary metadata repository that is to be set.

To identify the primary repository type ID for a given SAS application, see its metadata type documentation. For example, in SAS/Warehouse Administrator, the primary repository type ID is WHDWENV for the environment repository.

repos_id
specifies the returned repository ID for the primary metadata repository that was set. Use this ID as the REPOSID part of the metadata identifier in subsequent methods that access metadata in this repository.

l_meta2
specifies the returned metadata property list for the repository that was set. Includes the general, identifying information for that repository. Use this list with subsequent calls, such as _GET_METADATA_, to retrieve more information about the primary repository. See Example 2.

already
specifies the passed indicator. Optional. Specifies whether the repository that is to be set is accessible.

0 — (default) repository is not accessible. Perform the process to gain access to the repository.

1 — repository is accessible.

A repository might already be accessed for several reasons. If you know that the repository is already accessed, this indicator can be set to 1 to indicate that fact. Note that you should use this parameter with caution because possible future changes to a metadata repository structure might cause incorrect results.

Note: Mixed usage (both **0** and **1**) of the *already* parameter during a single execution of an application is strongly discouraged. All calls to _SET_PRIMARY_REPOSITORY_ during a single execution should use a single value, either 0 or 1. △

Using _SET_PRIMARY_REPOSITORY_

Primary and secondary repositories are discussed in "Metadata Repositories" on page 10.

The example shows how to attach to the sample primary repository (a warehouse environment called *Sample Demo*) that is shipped with SAS/Warehouse Administrator.

In the example, *repos_type* is set equal to WHDWENV because WHDWENV is the type ID of a SAS/Warehouse Administrator environment repository.

The documentation for the WHDWENV type describes only one property that is needed for a SAS libname statement—LIBRARY. The LIBRARY property, in turn, has the properties of the WHLIBRY metadata type. The documentation for the WHLIBRY type describes several properties that might be needed in a SAS libname statement—ENGINE, LIBREF, OPTIONS, and PATH.

To access a local metadata repository of type WHDWENV, the *l_meta* list only needs to include the ENGINE, PATH, and OPTIONS properties of the LIBRARY property, as shown in the example.

Once you have attached to the primary repository, you can use the _GET_METADATA_ method to return the list of possible secondary repositories. Specify the REPOSITORIES property in the *l_meta* list, and use the returned metadata identifier from the _SET_PRIMARY_REPOSITORY_ method, as shown in Example 2 on page 42.

Examples

Example 1: Access a Primary Metadata Repository

```
    /*
     * Access the sample primary metadata repository
     * that is shipped with SAS/Warehouse Administrator, which is
     * a warehouse environment called 'Sample Demo.'
     */
path="!SASROOT\whouse\dwdemo\_master";
repos_type='WHDWENV';
   /*
    *  Insert the Location information into
    *  the metadata list with a name of LIBRARY.
    */
l_inmeta=makelist();
l_lib=makelist();
l_inmeta=insertl(l_inmeta,l_lib,-1,'LIBRARY');
   /*
    *  Use the default Libname Engine
    *  to access a Local Path.
    */
l_lib=insertc(l_lib,' ',-1,'ENGINE');
l_path=makelist();
l_lib=insertl(l_lib,l_path,-1,'PATH');
l_opts=makelist();
l_lib=insertl(l_lib,l_opts,-1,'OPTIONS');
l_path=insertc(l_path,path,-1);
   /*
    *  Set the primary repository. If a bad return code
    *  is returned, then you cannot continue.
    */
call send(i_api,'_SET_PRIMARY_REPOSITORY_',
l_rc,l_inmeta,repos_type,primary_repos_id,l_meta);
l_inmeta=dellist(l_inmeta,'Y');
if l_rc = 0 then do;
   /*
    *  Accessed the primary repository correctly.
    */
```

Example 2: Using _GET_METADATA_ to list secondary repositories

```
    /*
     * Get the list of all repositories under
     * the primary repository
     * l_meta is l_meta returned from
     *  _SET_PRIMARY_REPOSITORY_.
     */
 l_reps=makelist();
 l_meta=setniteml(l_meta,l_reps,'REPOSITORIES');
 call send(i_api,'_GET_METADATA_',l_rc,l_meta);

   /* returns list: */
 l_meta=(ID='A0000001.WHDWENV.A0000001'
             NAME='Warehouse Administrator'
             DESC='Sample Demo Warehouse Environment.'
```

```
REPOSITORIES=(
    (ID='A0000001.WHDW.A000000E'
     NAME='Marketing Campaigns Data Warehouse'
     DESC='Sample Data Warehouse.'
     )[2455]
   )[2201]
 )[2271]
```

See Also

_SET_SECONDARY_REPOSITORY_

_SET_SECONDARY_REPOSITORY_

Attaches to a secondary metadata repository

Category: Repository Methods

Syntax

CALL SEND(*i_api*, '_SET_SECONDARY_REPOSITORY_', *l_rc*, *l_meta*, *repos_id*, <*already*>);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_meta	L	specifies the passed metadata property list for the secondary repository that is to be set.
repos_id	C	specifies the returned repository ID for the secondary metadata repository that was set. For details about the repository ID parameter, see "Identifying Metadata" on page 7.
already	N	specifies the passed indicator. Optional. Specifies whether the repository that is to be set is accessible.

Details

l_rc
specifies the return codes for the method.

Note: Be sure to check the return code for this method. Do not continue if the method fails. △

l_meta
specifies the passed metadata property list for the secondary repository that is to be set. You must pass the ID property in this list. See the following Usage notes.

repos_id
specifies the returned repository ID for the secondary metadata repository that was set. Use this ID as the *reposid* part of the metadata identifier in subsequent methods that access metadata in this repository.

already
specifies the passed indicator. Optional. Specifies whether the repository that is to be set is accessible.

0 — (default) repository is not accessible. Perform the process to gain access to the repository.

1 — repository is accessible.

A repository might already be accessed for several reasons. If you know that the repository is already accessed, this indicator can be set to 1 to indicate that fact. Note that you should use this parameter with caution because possible future changes to a metadata repository structure might cause incorrect results.

Note: Mixed usage (both **0** and **1**) of the *already* parameter during a single execution of an application is strongly discouraged. All calls to _SET_SECONDARY_REPOSITORY_ during a single execution should use a single value, either **0** or **1**. △

Using _SET_SECONDARY_REPOSITORY_

Primary and secondary repositories are discussed in “Metadata Repositories” on page 10. To set a secondary repository, you must pass the ID property for this method, and you can pass the LIBRARY property.

ID
specifies the metadata identifier of the secondary repository. Normally, the information that is needed to access the secondary repository is stored as a piece of metadata within the primary repository. If the metadata that is stored is sufficient for your application to access the secondary repository, then this is all that is required.

The example shows how to use the ID property in the *l_meta* list to set a secondary metadata repository.

LIBRARY
specifies an optional property that allows the stored metadata information to be overridden with the information that is specified here. An example of this might be if the administrator’s metadata libname definition does not gain you proper access to a secondary repository (for example, SAS/SHARE access versus access other than SAS/SHARE).

“Example: Set a Secondary Repository” on page 45 shows how to use the _GET_METADATA_ method to return the list of possible secondary repositories. Specify the REPOSITORIES property in the *l_meta* list, and use the returned metadata identifier from the _SET_PRIMARY_REPOSITORY_ method.

Note: The current interpreter for SAS/Warehouse Administrator allows only one secondary repository to be active at any given time. If you are attached to a secondary repository and you attach to another secondary repository, you will be detached from the first one before you are attached to the other. △

Example: Set a Secondary Repository

```
    /*
     *  Insert code to access the primary repository.
     *
     *  Get the list of available secondary repositories
     *  under this primary repository.
     */
l_reps=makelist();
l_meta=setniteml(l_meta,l_reps,'REPOSITORIES');
call send(i_api,'_GET_METADATA_',l_rc,l_meta);
if l_rc = 0 then do;

   num_reps=listlen(l_reps);
   if num_reps > 0 then do;
         /*
          * If there are any secondary repositories, select
          * one to set as the active one.
          */
      l_sec_rep=getiteml(l_reps,1);
      call send(i_api,'_SET_SECONDARY_REPOSITORY_',
                l_rc, l_sec_rep,sec_repos_id);
         /*
          * If l_rc = 0 then sec_repos_id contains
          * the 8 character repository id of this
          * repository.
          * This id is used as the first part of any
          * identifiers used to access metadata in this
          * secondary repository.
          */
     if l_rc = 0 then do;

/* continue processing */
```

See Also

_SET_PRIMARY_REPOSITORY_

_UPDATE_METADATA_

Updates specified metadata in a repository

Category: Write Methods

Syntax

CALL SEND(*i_api*, '_UPDATE_METADATA_', *l_rc*, *l_meta*);

Parameters

Parameter	Type	Description
i_api	Class	specifies the passed instance of METAAPI.CLASS. See "Using the Metadata API Class" on page 14.
l_rc	N	specifies the return codes for the method. A nonzero code indicates failure and means that *l_rc* is an error list identifier. For the error list format, see "Error Codes" on page 14.
l_metadata	L	specifies the passed metadata property list for the object that is to be updated. For the general format of this list, see "Metadata Property List" on page 14.

Details

l_meta
specifies the passed metadata property list for the object that is to be updated. The ID value in *l_meta* should be *reposid.typeid.instanceid*.

To add properties to an existing object, the ID value in *l_meta* should be *reposid.typeid.instanceid*.

Using _UPDATE_METADATA_

Keep the following in mind when you use the _UPDATE_METADATA_ method:

- Always check the return code of a write method call. A nonzero return code indicates that a failure has occurred while trying to write to the metadata. If a nonzero return code is returned, none of the changes that are indicated by this method call will be made.
- The_UPDATE_METADATA_ method supports indirect adds of dependent types. For example, a physical storage (WHSASSTR) will be created and added to a data table (WHDATTBL) when it is passed as the physical storage property in the *l_meta* list for the update of WHDATTBL. As another example, a new column (WHCOLDAT) will be created and added to a data table (WHDATTBL) when it is passed in the Columns property in the *l_meta* list for the update of WHDATTBL.
- The_UPDATE_METADATA_ method does not support cascading updates. You can update properties of an existing instance of a type only by using an _UPDATE_METADATA_ call directly on that instance.

- When you pass a list for a property of an object through the _UPDATE_METADATA_ method, and the existing property list is empty for the object — or if the list can be a list of objects, UPDATE will add (for independent types) or create and add (for dependent types) the object to the existing list.

  For example, given an existing job (WHJOB) with several output tables associated with it, an existing detail table (WHDETAIL) will also be associated with that job when it is passed in the output tables property in the *l_meta* list for the update of WHJOB. As another example, given an existing OLAP table (WHOLPTBL) with several extended attributes (WHEXTATR) associated with it, a new extended attribute will be created and associated with when it is passed in the extended attributes property in the *l_meta* list for the update of WHOLPTBL.

- When you pass a list for a property through the _UPDATE_METADATA_ method, and the property can only be a list with a single object, UPDATE will replace the existing object with another existing object that is being passed in *l_meta*.

  For example, given that an OLAP table has an existing library (WHLIBRY) associated with it, a call to the _UPDATE_METADATA_ method for the OLAP table (WHOPLTBL) will replace the library with another existing library when it is passed in the *l_meta* list for the update of WHOLPTBL.

Examples

Example 1: Add New Columns to the Selected Table

```
l_meta=makelist();

   /*
    * object_id is the ID of an existing Data Table object.
    */

l_meta=insertc(l_meta,object_id,-1,'ID');

   /*
    *  Define a column.  The COLUMNS property
    *  contains a sublist per column.
    */

l_cols=makelist();
l_col=makelist();

l_cols=insertl(l_cols,l_col,-1);

l_meta=insertl(l_meta,l_cols,-1,'COLUMNS');

col_id=repos_id||'.'||'WHCOLUMN';

l_col=insertc(l_col,col_id,-1,'ID');
l_col=insertc(l_col,'SaleDate',-1,'NAME');
l_col=insertc(l_col,'Date of Sale',-1,'DESC');
l_col=insertc(l_col,'N',-1,'TYPE');
l_col=insertn(l_col,8,-1,'LENGTH');
l_col=insertc(l_col,'DATE7.',-1,'FORMAT');
l_col=insertc(l_col,'DATE7.',-1,'INFORMAT');

   /*
    *  Update any additional properties.
```

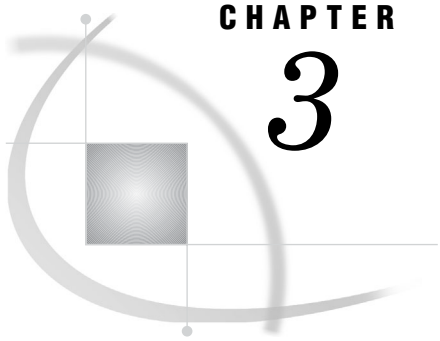
```
    *  :
    *  :
    */

   /*
    *  Update the table.
    */

call send(i_api,'_UPDATE_METADATA_',l_rc,l_meta);

if l_rc = 0 then do;

   put 'Table Updated successfully';

   end; /* if */
else do;

   msg=getnitemc(l_rc,'MSG',1,1,'ERROR:
       _UPDATE_METADATA_ FAILED');
   put msg;

   list_rc=dellist(l_rc);

   end; /* else */


l_meta=dellist(l_meta,'Y');
```

Example 2: Add Extended Attributes

```
   /*
    * object_id is the ID of an existing Data Table object.
    */

l_meta=makelist();

l_meta=insertc(l_meta,object_id,-1,'ID');

   /*
    * Attributes
    */

l_attrs=makelist();
l_attr=makelist();

l_attr=insertc(l_attr,'Loader',1,'NAME');
l_attr=insertc(l_attr,'Oracle',-1,'VALUE');

l_attr=insertc(l_attr,'Name of loader used',-1,'DESC');
l_attrs=insertl(l_attrs,l_attr,-1);


l_meta=insertl(l_meta,l_attrs,-1,'EXTENDED ATTRIBUTES');
call send(i_api, '_UPDATE_METADATA_',l_rc,l_meta);
```

See Also

_DELETE_METADATA_, _ADD_METADATA_



CHAPTER

3

SAS/Warehouse Administrator Metadata Types

Overview of SAS/Warehouse Administrator Metadata Types

This section describes the metadata types that are defined for SAS/Warehouse Administrator. These types are grouped under the WHOUSE component. They are stored in the SASHELP.DWAPI catalog. For a complete list of these types, see "Index to SAS/Warehouse Administrator Metadata Types" on page 70.

To use a metadata type, you pass its ID (such as WHDETAIL) to the metadata API methods listed in "Index to Metadata API Methods" on page 16.

What Is a Metadata Type?

A *metadata type* is a template that models the metadata for a particular kind of object in an application. For example, the metadata type WHDETAIL models the metadata that is maintained for a detail table in SAS/Warehouse Administrator. WHDETAIL's parameter list matches the items of metadata maintained for a detail table, such as ID, NAME, COLUMNS, and INPUT SOURCES.

A three-level metadata identifier (*REPOSID.TYPEID.INSTANCEID*) is passed to methods that are used to read or write metadata. The type ID in this identifier, such as WHDETAIL, specifies a metadata type that describes the content of the metadata to be read or written.

You can use all metadata types with the read methods. See "Writing Metadata" on page 59 for a discussion of metadata types and write methods.

Metadata Repository Types

You can store an application's metadata in a repository. A metadata repository type is a template that models the metadata for a particular kind of metadata repository. For example, the metadata repository type WHDWENV models the metadata for an environment repository in SAS/Warehouse Administrator. WHDWENV's parameter list matches the items of metadata that are maintained for an environment, such as ID, NAME, DESCRIPTION, and LIBRARY.

Repository types are used with the Repository Methods that are described in "Index to Metadata API Methods" on page 16. These methods are used to attach to a given repository so that its metadata can be read or written.

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 in an environment. Accordingly, there are two metadata repository types for SAS/Warehouse Administrator:

WHDWENV
: specifies the metadata repository type for a data warehouse environment. For details, see "WHDWENV" on page 104.

WHDW
: specifies the metadata repository type for a data warehouse. For details, see "WHDW" on page 101.

Metadata Type Inheritance

Metadata types inherit properties from their parent type, as shown in the foldout in Appendix 2. Independent metadata types are represented as a rectangle. Dependent types are represented by a rectangle with rounded corners. (For an explanation of these broad categories, see "Independent and Dependent Metadata Objects" on page 53.)

Using Metadata Types

Relationships Among Metadata Types

This section describes the relationships among metadata types in SAS/Warehouse Administrator. By understanding these relationships, you can

- access metadata types by using their associated properties
- identify which metadata types can be created independently and which ones must be created in association with other types.

Metadata type relationships are presented in several diagrams, each diagram showing only a part of the total structure. These diagrams identify various ways to access a given type of metadata. The following notes apply to all diagrams:

- Each node in the diagram represents a type or supertype.
- Each line (connection) indicates that two types have a relationship between them. The text closest to each node indicates the name of the property that will return the corresponding node's general information.
- The type names that are contained in the nodes represent the highest supertype that can have this relationship. When you process a relationship, use the _IS_SUBTYPE_OF_ method to determine the current node type.
- Independent metadata types are represented as a rectangle. Dependent types are represented by a rectangle with rounded corners.

Independent and Dependent Metadata Objects

A *metadata object* is an instance of a metadata type—the metadata for an element in an application, such as a table or column.

An *independent* metadata object can be created by itself. For example, a WHPERSON object can be created independently of any other object.

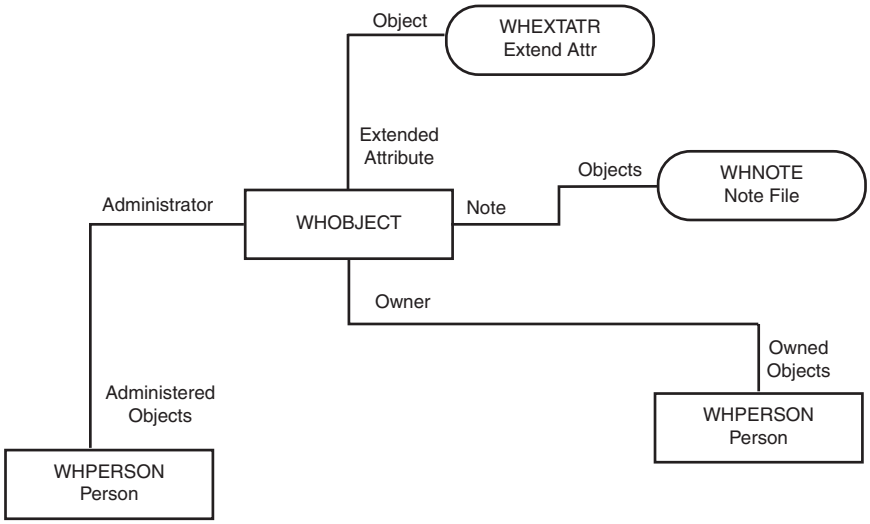
A *dependent* metadata object cannot be created by itself. For example, a WHCOLUMN object cannot be created without first being associated with a WHTABLE object.

In the metadata type models in this section, independent metadata types are represented as a rectangle, and dependent types are represented by a rectangle with rounded corners.

General Metadata Type Model

The following figure shows how to access general information about any metadata object in SAS/Warehouse Administrator.

Figure 3.1 General Metadata Type Model



Host Metadata Type Model

The following figure shows how to access a common set of metadata for any host in SAS/Warehouse Administrator.

Figure 3.2 Host Metadata Type Model

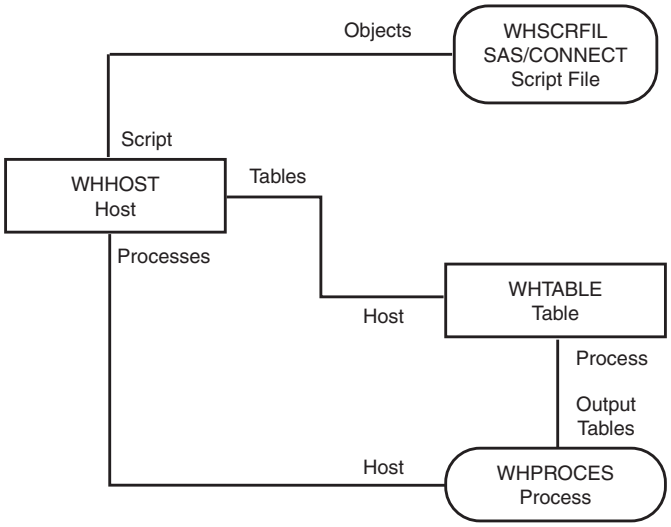


Table Property Metadata Type Model

The following figure shows how to access property metadata for any table in SAS/Warehouse Administrator.

Figure 3.3 Table Property Metadata Type Model

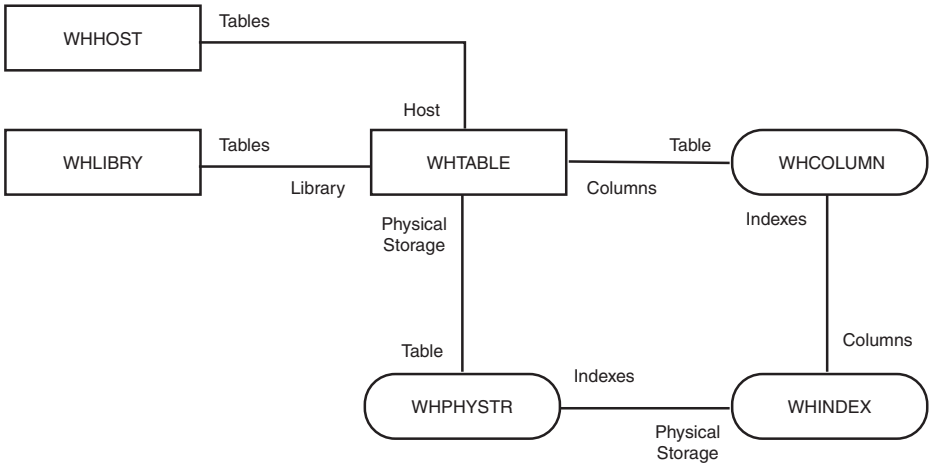
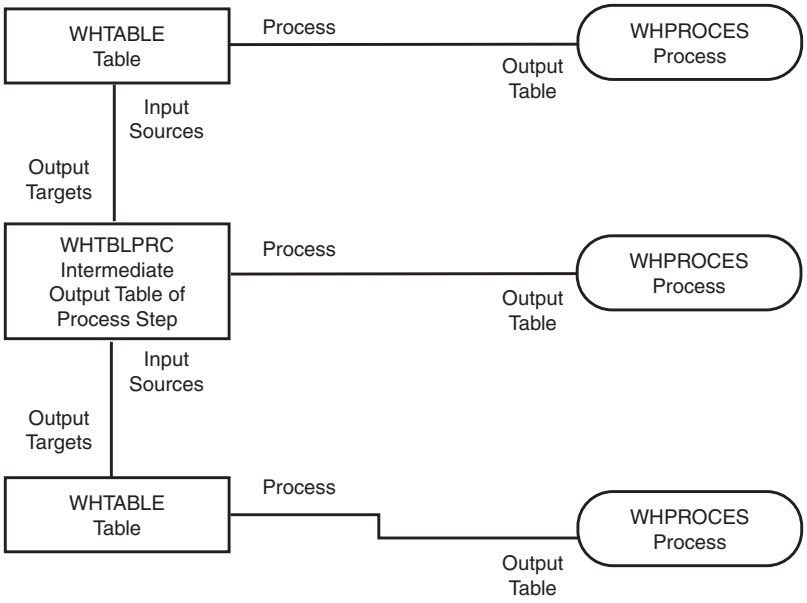


Table Process Metadata Type Model

The following figure shows how to access process metadata for any table in SAS/Warehouse Administrator.

Figure 3.4 Table Process Metadata Type Model



Tables inherit the metadata that is shown in Figure 3.1 on page 54, Figure 3.6 on page 57, and Figure 3.7 on page 57.

Note: There can be zero or more intermediate WHTBLPRC objects between two WHTABLE objects. Use the _IS_SUBTYPE_OF_ method to determine if the object that you are currently processing is WHTBLPRC. △

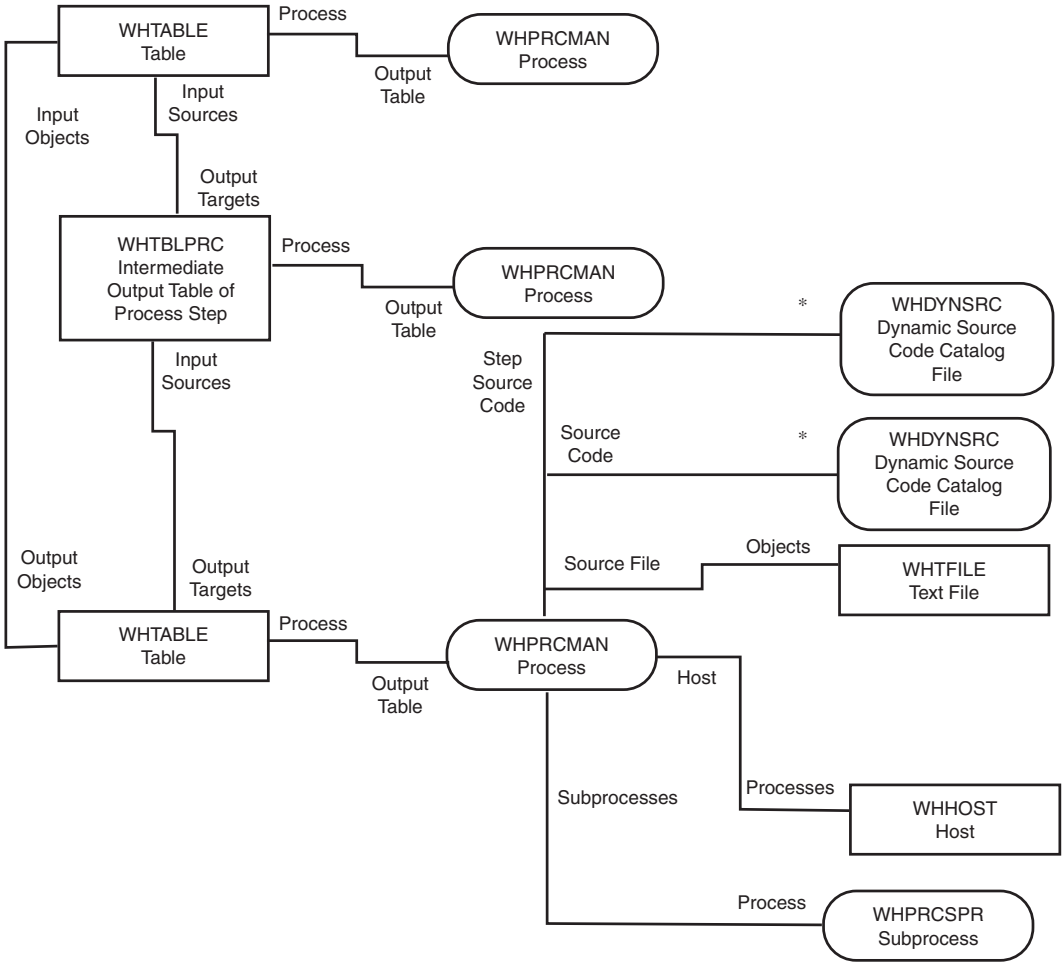
Note: If there are no intermediate WHTBLPRC objects, the outputs from the OUTPUT TARGETS and OUTPUT OBJECTS properties are identical. The same is true for the INPUT SOURCES and INPUT OBJECTS properties. △

Note: When you check the type of a table object, check for WHTBLPRC and not for WHTABLE. Because WHTBLPRC is a subtype of WHTABLE, this check would always come back true. △

Process Type Model

In SAS/Warehouse Administrator, load processes and similar jobs are defined through the Process Editor. Each process is defined by a metadata object. The following figure shows an example process flow.

Figure 3.5 Process Type Model



Note: See the metadata types that are marked with an asterisk (*) in the previous figure. For those types, because the SOURCE CODE property points to an entry that is dynamically generated when requested, this relationship cannot be traversed in the WHDYNSRC to WHPRCMAN direction. △

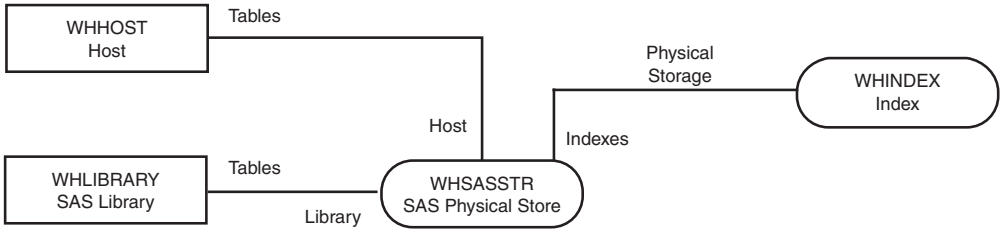
Note that there can be zero or more WHTBLPRC type objects between two WHTABLE subtype objects. The previous figure shows one intermediate WHTBLPRC object.

This diagram shows the overall process flow, as well as any relationships that might be specific to the WHPRCMAN type objects. Note that for simplicity, the relationships have been drawn for only one of the WHPRCMAN type objects in the diagram, but these relationships exist for all WHPRCMAN type objects.

Physical Storage Metadata Type Models

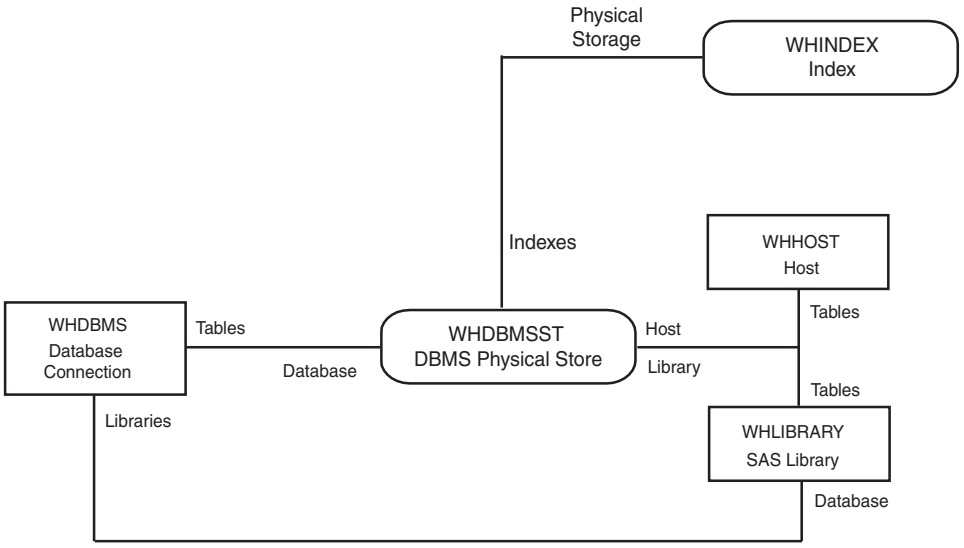
Physical storage information is different for SAS data stores (type WHSASSTR) and DBMS data stores (type WHDBMSST). The following figure shows how to access a common set of metadata for a SAS data store.

Figure 3.6 SAS Data Store Metadata Type Model



The following figure shows how to access a common set of metadata for a DBMS data store.

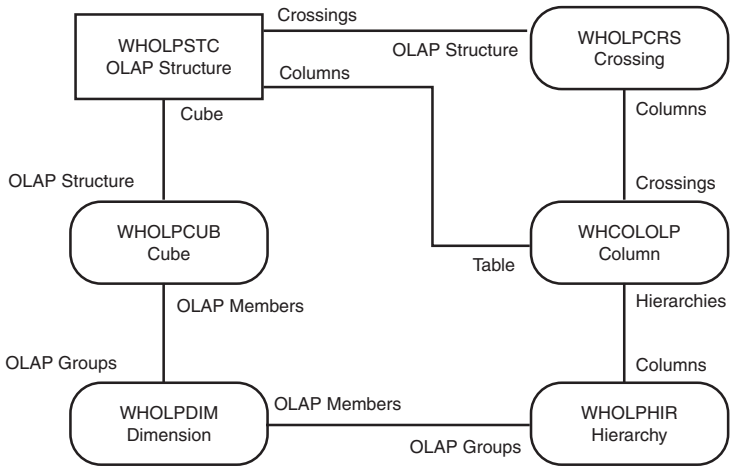
Figure 3.7 DBMS Data Store Metadata Type Model



OLAP Metadata Type Model

The following figure shows how to access metadata that defines the structure of an OLAP table, Group, or MDDB in SAS/Warehouse Administrator including OLAP cubes, crossings, dimensions, hierarchies, and columns.

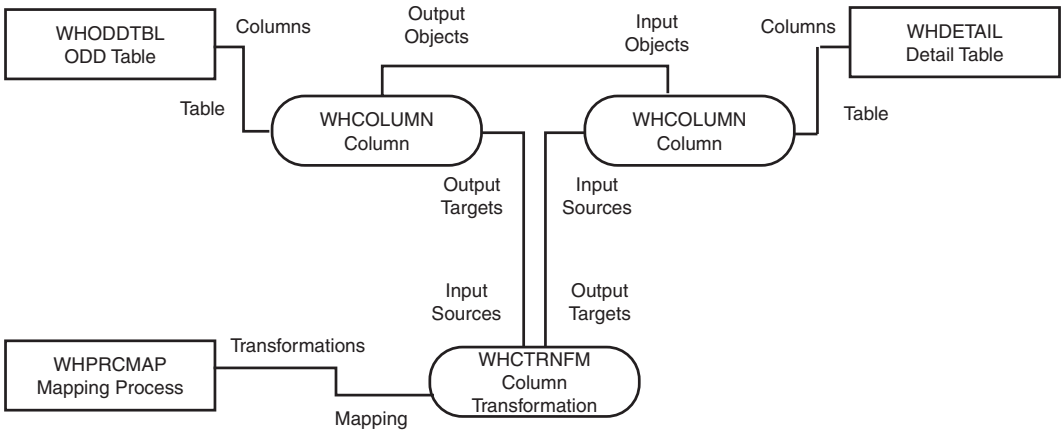
Figure 3.8 OLAP Metadata Type Model



Column Mapping Types: ODD to Detail Table Model

The following figure shows how to access the metadata that defines the column mappings between an operational data definition (ODD) and a detail table.

Figure 3.9 Mapping Model: ODD to Detail Table



The ODD type (WHODDTBL) and detail table type (WHDETAIL) inherit the metadata that is shown in Figure 3.4 on page 55.

Note: If the logic that is needed to transform the operational data column into the detail data column is not important for your application, then you can use the output objects/input objects relationship. For details, see "INPUT and OUTPUT Properties" on page 64. △

Note: There can be zero or more intermediate WHCTRNFM objects between two WHCOLUMN objects. Use the _IS_SUBTYPE_OF_ method to determine if the object that you are currently processing is a WHCOLUMN or a WHCTRNFM. △

Note: If there are no intermediate WHCTRNFM objects, the outputs from the OUTPUT TARGETS and OUTPUT OBJECTS properties are identical. The same holds true for the INPUT SOURCES and INPUT OBJECTS properties. △

Writing Metadata

You can read all of the metadata types that are defined for SAS/Warehouse Administrator, but you cannot write them all. You can pass only certain types to the metadata API write methods. Not all write methods are valid for those types that can be written.

You can write metadata for many objects that can be displayed in the Explorer frame. You can also write metadata for host definitions and other entities that are shared among warehouses within an environment.

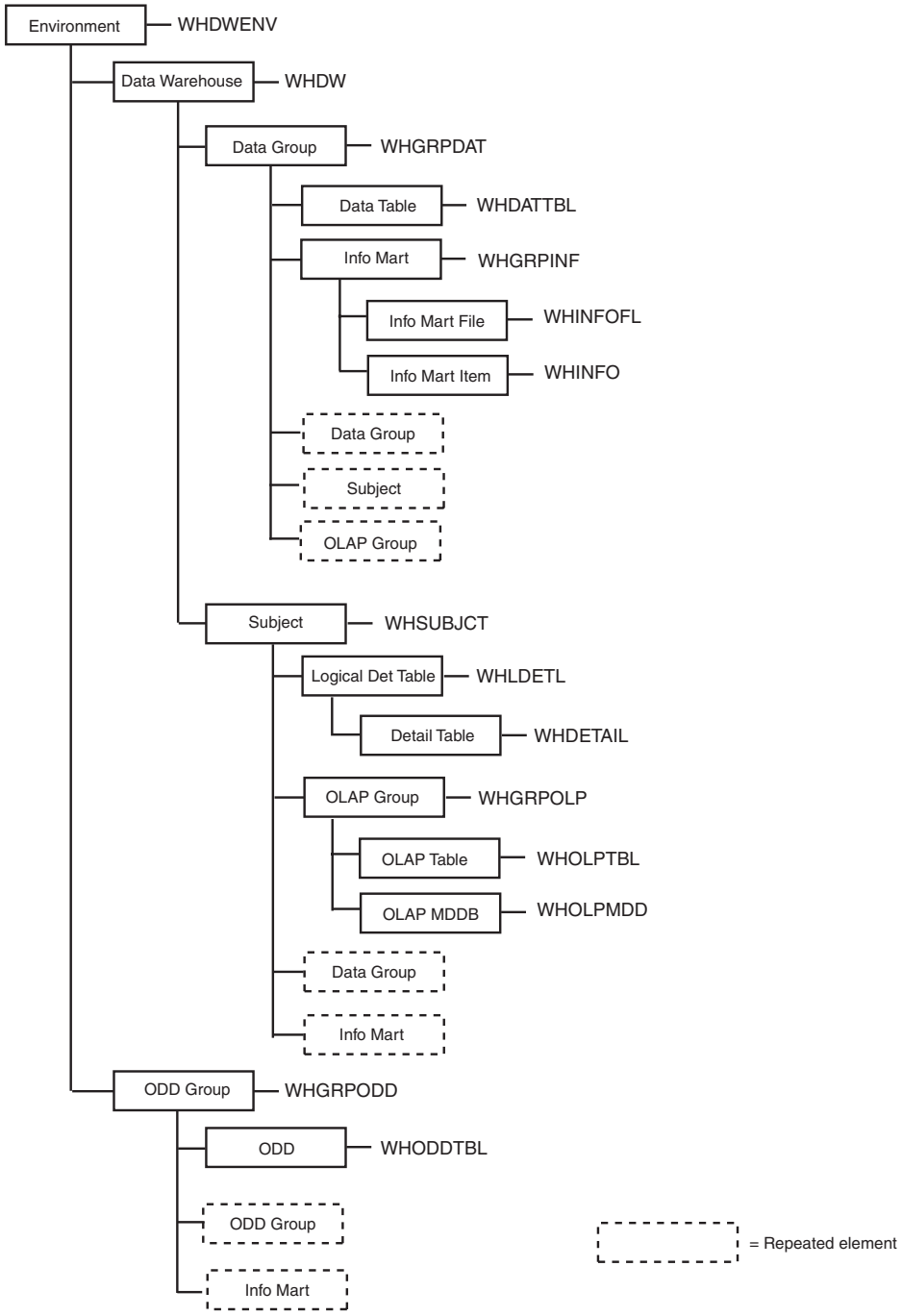
The documentation for each type in the metadata type dictionary identifies the write methods that are valid for each metadata type.

Writing Explorer Objects

Objects that are displayed in the SAS/Warehouse Administrator Explorer frame, such as warehouses, subjects, and tables, are members of groups. When you add the metadata for an Explorer object, you must identify the group to which it belongs. This is done by passing the metadata identifier of the target group along with the other parameters for the object.

The metadata types for Explorer objects have a GROUP property that lists the metadata identifiers of the groups in which to add a new object. The following figure lists the groups and the metadata types that are valid in each group.

Figure 3.10 Hierarchy of Groups and Members in SAS/Warehouse Administrator Explorer



Note: Although the GROUP property takes a list of GROUP identifiers, the object is currently added only to the first GROUP that is specified in the list. For example, when adding a WHDETAIL type object (a detail table), the metadata identifier that is specified in the GROUP property list item must be of the type WHLDETL. △

Overview of the Process Editor

This section gives a brief overview of the Process Editor so that you can better understand how the process metadata types relate to the user interface. For details about the Process Editor window, display the Process Editor, then select

Help ▶ Using This Window

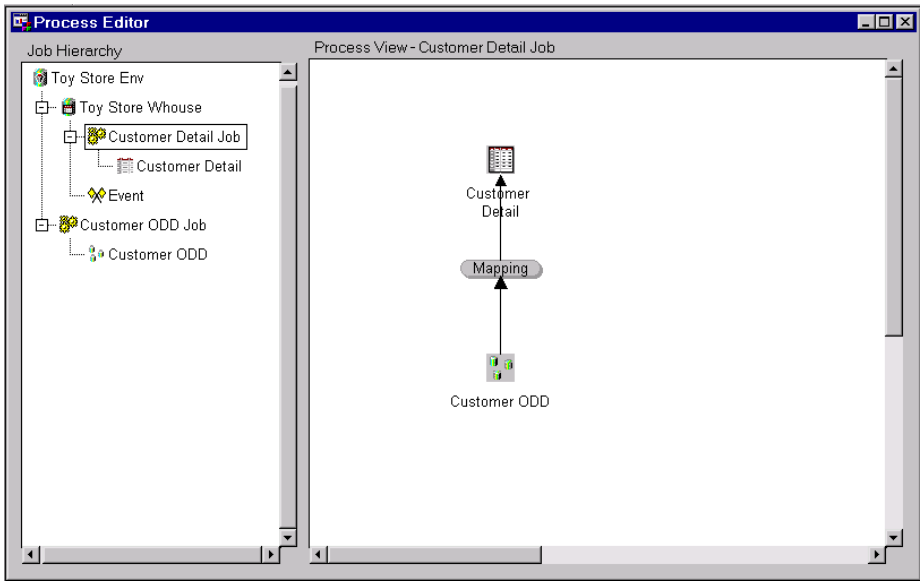
from the menu bar.

If you open the Process Editor from the Explorer by selecting

Tools ▶ Process Editor

from the menu bar, Job Hierarchy is the default view in the left panel, as shown in the following display.

Display 3.1 Process Editor: Job Hierarchy and Process View



In the left panel, the Job Hierarchy displays all of the jobs that are defined in the current Warehouse environment. In the preceding figure, only two jobs are defined: Customer Detail Job and Customer ODD Job. The Customer Detail Job (item in the left panel with the rectangle around it) is the active job.

In the right panel, the Process View shows the process flow that is associated with the active job (Customer Detail Job). A *process flow* is a user-defined diagram in the Process View of the Process Editor. It is composed of symbols, with connecting arrows and descriptive text, that illustrate the sequence of each process that is associated with the job that is selected in the Job Hierarchy of the Process Editor. The process flow illustrates how the data moves from input source(s) to output table(s) and what extractions and transformations occur in between.

Note: A job only creates the output table(s) that are listed under its icon in the left panel of the Process Editor. The other loadable tables in the process flow are inputs to the job. △

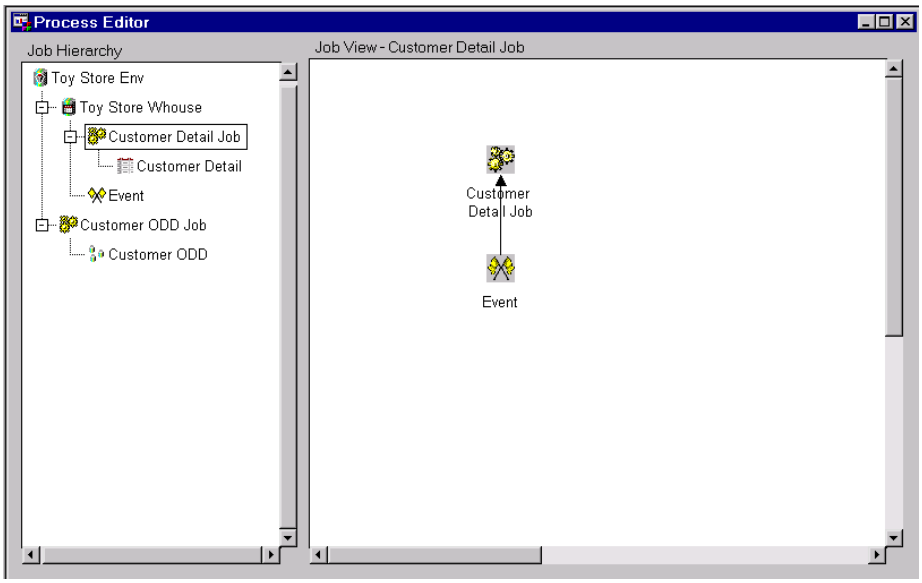
For example, in the preceding display, the Customer Detail Job only creates the Customer Detail table. It does not create Customer ODD. Customer ODD is created by

a separate job—Customer ODD Job. Customer ODD is an input to the Customer Detail Job.

In the previous display, note that one event has been defined for the Toy Store Whouse. An event is a metadata record that specifies a condition for controlling a job, such as checking for certain return codes or verifying the existence of a file. To use events, you must create them, include them in a *job flow*, and then write a metadata API program that reads the job flow and generates code for it.

Job flows are displayed in the Job View of the Process Editor. In order to switch from the Process View to the Job View in the right panel of the Process Editor, click the right mouse button in the background and select **`Job View`** from the pop-up menu. The right panel in the following display illustrates a job flow that has been defined for the Customer Detail Job.

Display 3.2 Process Editor: Job Hierarchy and Job View



Reading Process Flow Metadata

In SAS/Warehouse Administrator, a *process* is a metadata record that is used to generate or retrieve a routine that creates warehouse data stores, or one that extracts data, transforms data, or loads data into data stores. You can link these tables together to form a process flow. The Process Editor in SAS/Warehouse Administrator is used to create process flows such as the one shown in the following figure.

Figure 3.11 Process Flow in SAS/Warehouse Administrator Process Editor

In the previous figure, information moves from the bottom up—from the ODD named ODD 1, to a mapping step, to the Credit data table.

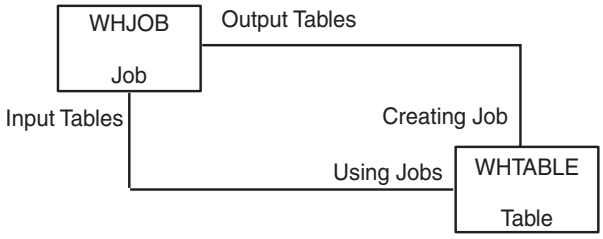
The icons shown in the figure—ODD 1 and Credit data table—represent loadable tables. A loadable table can be a source, such as ODD 1 in the figure; a target, such as Credit data table; or both a target and a source.

The Mapping box that is shown in the figure represents an intermediate output table—the output of a process step between sources and targets.

Note: Process flow diagrams do not depict process objects. These diagrams show how data moves from one loadable table (icon), through an intermediate output table (box), to a target loadable table (icon). △

In addition to the process metadata, the process flow metadata has information about how the tables are related to the job. The following figure shows the properties that relate jobs to tables.

Figure 3.12 Process Flow Metadata: Jobs



Loadable Tables and WHTABLE Subtypes

Each loadable table has metadata of subtype WHTABLE. For a list of WHTABLE subtypes, see the diagram on the foldout in Appendix 2.

WHTABLE subtypes give you information about where the output data of the step resides and any other metadata about the object that has been gathered using the Properties frames, such as data host, data library, table name, and columns.

For each WHTABLE subtype, you can retrieve the corresponding process metadata (WHPROCES) by using the PROCESS property. Any step for which no process information exists will return an empty list for the PROCESS property. The RESPONSIBILITY property will indicate whether a process has been defined for this table, and if so, who is responsible for generating the code.

Intermediate Output Tables and WHTBLPRC Subtypes

Each intermediate output table has metadata of subtype WHTBLPRC. For a list of WHTBLPRC subtypes, see "Metadata Type Inheritance" on page 52.

All WHTBLPRC subtypes have a property, CREATES DATA, that indicates whether the table has output data or is a placeholder only. If CREATES DATA =0, then the table is a placeholder only. (The **`This process has no output data`** selection has been made on the process properties Output Data tab.) An analogy would be a DATA step that performs processing but is coded with DATA _NULL_.

Using the _IS_SUBTYPE_OF_ method of the API, you can determine if the currently returned table from the INPUT SOURCE property is an intermediate table or an actual loadable table. You can use the method as follows:

```
call send(i_api, '_IS_SUBTYPE_OF_',rc,
input_source_type,'WHTBLPRC',is_process_table);
```

If IS_PROCESS_TABLE is returned as a 1, then the current table is an intermediate table in a process step. If it returns a zero, then it is a loadable table.

Process Objects and WHPROCES Subtypes

Each process (metadata object that creates a table) is of subtype WHPROCES. For a list of WHPROCES subtypes, see "Metadata Type Inheritance" on page 52.

These subtypes give you the process information that has been entered using the Edit Load Step frame or the Process Properties frame for a loadable object. This information includes the name of the person who is writing the code, the host where the code should execute, and column mappings.

For each WHPROCES subtype, you can retrieve the corresponding WHTABLE by using the OUTPUT TABLES property. For more information on the relationships of metadata that are associated with processes, see the table and process models in "Relationships Among Metadata Types" on page 53.

INPUT and OUTPUT Properties

There are two sets of properties that deal with process flow to a table or column—one for input and one for output.

INPUT SOURCES
: specifies an SCL list of general identifying information about the nearest intermediate output table or loadable table that is a source to the current table or column.

  Given the process flow diagram that is shown in Figure 3.11 on page 63, the INPUT SOURCES property of Credit data table would return the intermediate table named Mapping.

INPUT OBJECTS
: specifies an SCL list of general identifying information about the nearest loadable table that is a source to the current table or column.

  Given the process flow diagram that is shown in Figure 3.11 on page 63, the INPUT OBJECTS property of Credit data table would return the loadable ODD table named ODD 1.

OUTPUT TARGETS
: specifies an SCL list of general identifying information about the nearest intermediate output table or loadable table that is a target for the current table or column.

  Given the process flow diagram that is shown in Figure 3.11 on page 63, the OUTPUT TARGETS property of ODD 1 would return the intermediate table named Mapping.

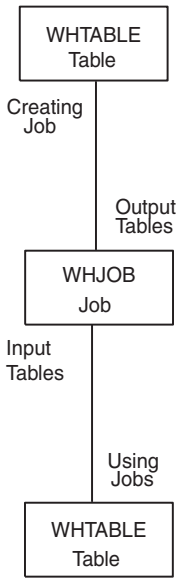
OUTPUT OBJECTS
specifies an SCL list of general identifying information about the nearest loadable table that is a target for the current table or column.

Given the process flow diagram that is shown in Figure 3.11 on page 63, the OUTPUT OBJECTS property of ODD 1 would return the loadable table named Credit data table.

Input Tables, Output Tables, and Job Metadata

Each job can have input and output tables that are associated with them. As shown in Figure 3.13 on page 65, the WHJOB type has two properties, Output Tables and Input Tables, that can retrieve this information. Both properties will return WHTABLE subtypes. A WHJOB type can return more than one WHTABLE as its input or output. The WHTABLE subtype has two properties that associate it to the job: Using Jobs and Creating Job. The WHTABLE subtype property, Using Jobs, will return all WHJOB types that use the WHTABLE subtype as an input table. The WHTABLE subtype property, Creating Job, will return only one WHJOB type because you can create a table only in one job.

Figure 3.13 WHJOB: Input Tables and Output Tables



Reading Job Metadata

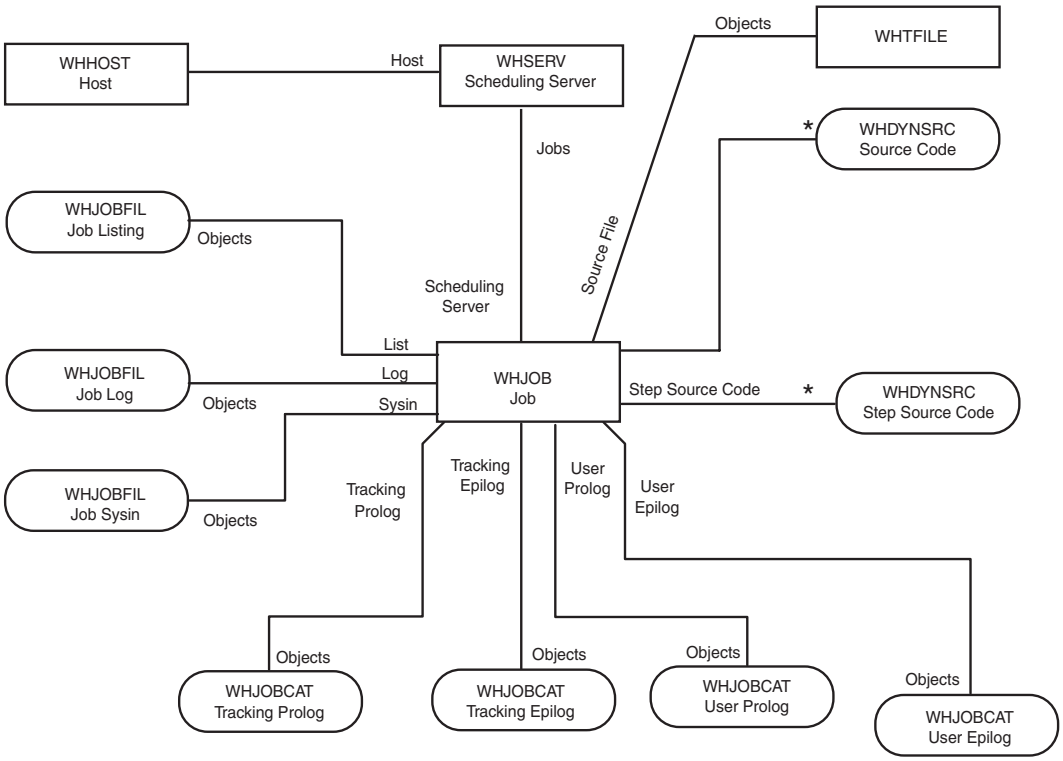
In SAS/Warehouse Administrator, a *job* is a metadata object that specifies the processes that create one or more data stores (output tables). You can join these processes together to form a job flow. The Process Editor in SAS/Warehouse Administrator is used to create job flows.

Each job has metadata of type WHJOB. WHJOB types give you information about the job that has been gathered using the Properties frames, such as scheduling server, location of generated source files, scheduling starting times, and tracking user prologs and epilogs. You can retrieve the corresponding metadata by using job properties. For example, to retrieve the associated tracking prolog for a job, you need to use the

TRACKING PROLOG property that returns a WHJOBCAT metadata type that contains all of the tracking prolog information.

Each scheduled job requires a scheduling server association. When you request the SCHEDULING SERVER property of a job, a WHSERV object will be returned. The HOST property of WHSERV returns a WHHOST object that is the defined host for this scheduling server as shown in the following figure.

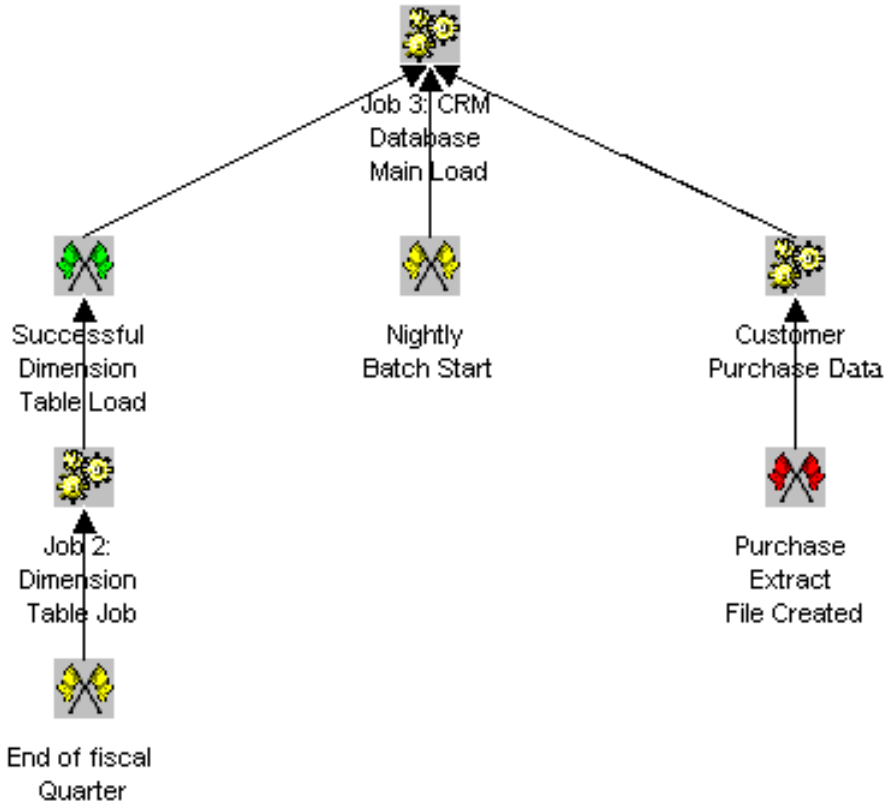
Figure 3.14 Job Type Model



Reading Job Flow Metadata

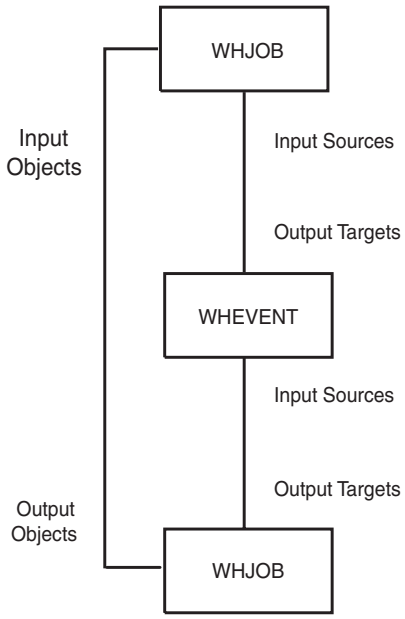
In SAS/Warehouse Administrator, *job flow* defines the relationship between jobs and events. This metadata is used to define dependencies between jobs within the warehouse. The Process Editor in SAS/Warehouse Administrator is used to create job flows such as the one shown in the following figure:

Figure 3.15 Job Flow in SAS/Warehouse Administrator



In the preceding figure you can interpret the chart as follows: Job 3: CRM Database Main Load is dependent in the settings of two events, Successful Dimension Table Load and Nightly Batch Start, as well as the execution of the job Customer Purchase Data. The following figure shows the metadata relationships that are defined between jobs and events. These relationships define the job flow.

Figure 3.16 Job and Event Relationships



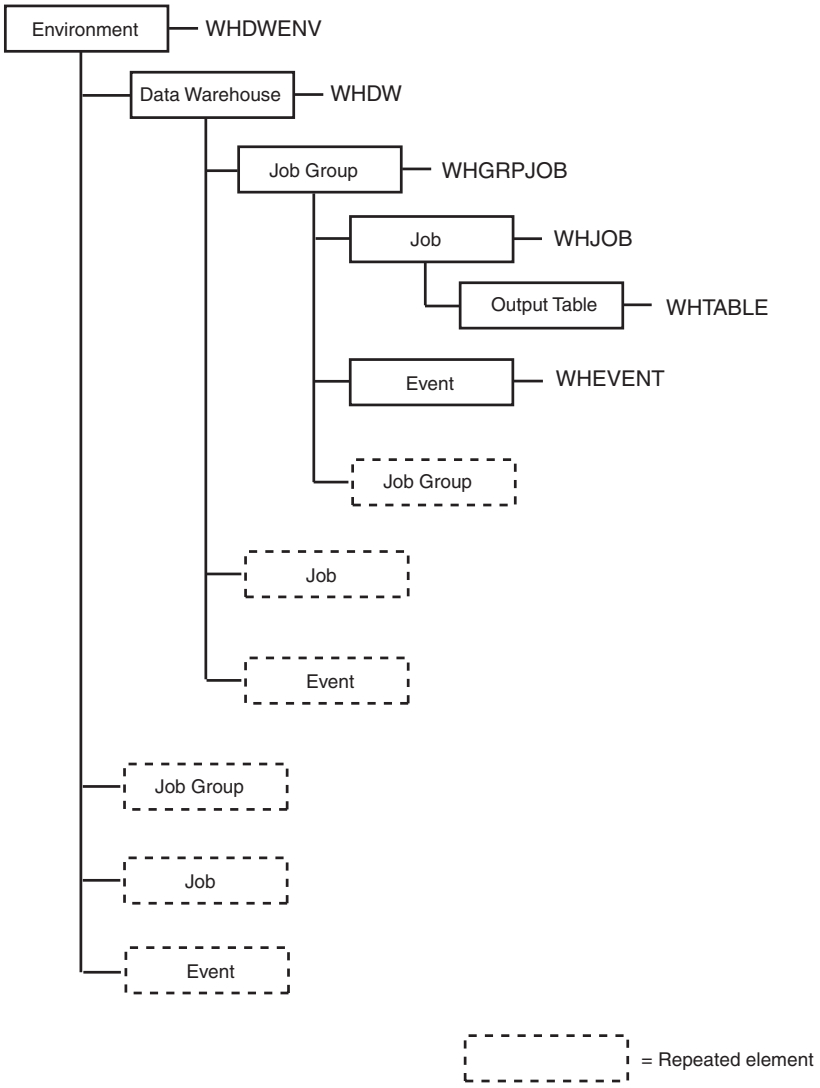
The previous figure shows the relationship between jobs and events in the Job View of the Process Editor. These relationships are used to define Job Dependencies within SAS/Warehouse Administrator.

Note: If there are no intermediate WHEVENT objects, the outputs from OUTPUT TARGETS and OUTPUT OBJECTS properties are identical. The same is true for INPUT SOURCES and INPUT OBJECTS. △

Reading Job Hierarchy Metadata

This section describes how to read the metadata for the objects in the Job Hierarchy panel of the Process Editor, as shown in Display 3.1 on page 61. These objects have a PROCESS GROUPS property that lists the metadata identifiers of the group that contains the object. The types also have a PROCESS MEMBERS property that lists the metadata identifiers of the members of the object. The following figure shows the types that these properties can return.

Figure 3.17 Metadata Types in the Job Hierarchy



Using Icon Information

The catalog that is returned for the icon property will always be SASHELP.I0808. Depending on your particular use of the value that is returned, this image size might not fit your needs. The returned image name can reside in other SASHELP catalogs that contain different sizes. When the icon property is used with an _ADD_METADATA_ or _UPDATE_METADATA_ property list, the image name that is passed must exist in both the SASHELP.I0808 and SASHELP.I0404 catalogs.

If the image entry that is passed is not passed as residing in one of these catalogs or the passed entry name cannot be found in both of these catalogs, an error is returned. If a blank value is passed for the ICON item in the property list for the _ADD_METADATA_ or _UPDATE_METADATA_ method, the default icon for the type will be used. To reset the icon back to the default icon for this type, you should pass a blank value as the value of the ICON item in the property list that is passed to _UPDATE_METADATA_.

Index to SAS/Warehouse Administrator Metadata Types

The metadata type dictionary describes SAS/Warehouse Administrator types in alphabetical order. In this section, metadata types are listed by category in order to give you a general idea of what types are available and how they are used.

Table 3.1 SAS/Warehouse Administrator Metadata Types

Category	SAS/Warehouse Administrator Metadata Type	Description
Column Types	"WHCOLDAT" on page 74	Metadata type for data table columns
	"WHCOLDTL" on page 75	Metadata type for detail table columns
	"WHCOLODD" on page 77	Metadata type for ODD columns
	"WHCOLOLP" on page 79	Metadata type for OLAP columns
	"WHCOLSCL" on page 81	Metadata type for statistic columns in summary tables and MDDBs
	"WHCOLSUM" on page 83	Base metadata type for columns in summary tables and MDDBs
	"WHCOLTIM" on page 85	Metadata type for _LOADTM columns
	"WHCOLUMN" on page 87	Base metadata type for table columns
Extended Attribute Type	"WHEXTATR" on page 114	Metadata type for extended attributes
Global Metadata Types	"WHDBMS" on page 95	Metadata type for DBMS connection definitions
	"WHHOST" on page 126	Metadata type for host definitions
	"WHPERSON" on page 202	Metadata type for person records
	"WHSERV" on page 235	Metadata type for the scheduling server
	"WHSRVAT" on page 237	Metadata type for the Windows NT AT scheduling server
	"WHSRVCRN" on page 240	Metadata type for Unix Cron scheduling server
	"WHSRVNUL" on page 242	Metadata type for the Null scheduling server
Index Type	"WHINDEX" on page 129	Metadata type for indexes that are associated with tables and columns
Object Types—Explorer	"WHDATTBL" on page 92	Metadata type for data tables
	"WHDETAIL" on page 99	Metadata type for detail tables
	"WHDW" on page 101	Metadata type for data warehouses
	"WHDWENV" on page 104	Metadata type for warehouse environments
	"WHGRPDAT" on page 116	Metadata type for data groups
	"WHGRPINF" on page 118	Metadata type for InfoMarts
	"WHGRPODD" on page 121	Metadata type for ODD groups
	"WHGRPOLP" on page 123	Metadata type for OLAP groups

	"WHGRPSUM" on page 125	Metadata type for summary groups
	"WHINFO" on page 131	Metadata type for InfoMart items
	"WHINFOFL" on page 135	Metadata type for InfoMart files
	"WHLDETL" on page 146	Metadata type for detail logical tables
	"WHODDTBL" on page 184	Metadata type for ODDs
	"WHOLPMDD" on page 196	Metadata type for OLAP MDDBs
	"WHOLPSTC" on page 198	Base metadata type for OLAP tables, groups, and MDDBs
	"WHOLPTBL" on page 200	Metadata type for OLAP tables, groups, and MDDBs
	"WHSUBJCT" on page 244	Metadata type for subjects in a warehouse
	"WHSUMDDB" on page 248	Metadata type for SAS Summary MDDBs
	"WHSUMTBL" on page 251	Metadata type for summary tables
	"WHTABLE" on page 254	Base metadata type for tables
Object Types—Intermediate Output Tables	"WHTBLMAP" on page 257	Metadata type for intermediate output tables that are produced by column mapping processes
	"WHTBLPRC" on page 259	Base metadata type for intermediate output tables that are produced by processes
	"WHTBLREC" on page 261	Metadata type for intermediate output tables that are produced by record selector processes
	"WHTBLUSR" on page 263	Metadata type for intermediate output tables that are produced by user exit processes
	"WHTBLXFR" on page 265	Metadata type for intermediate output tables that are produced by data transfer processes
Object Types—OLAP	"WHOLAP" on page 188	Base metadata type for OLAP dimension, hierarchy, and crossing
	"WHOLPCRS" on page 189	Metadata type for OLAP crossing
	"WHOLPCUB" on page 191	Metadata type for OLAP cube
	"WHOLPDIM" on page 193	Metadata type for OLAP dimension
	"WHOLPHIR" on page 194	Metadata type for OLAP hierarchy
Object Types—Process Editor	"WHEFILE" on page 109	Metadata type for external file inputs to ODDs
	"WHEVENT" on page 112	Metadata type for events
	"WHGRPJOB" on page 120	Metadata type for job groups
	"WHJOB" on page 138	Metadata type for jobs
	"WHODTTBL" on page 186	Metadata type for ODTs (Data Files)

	"WHPOBJCT" on page 205	Metadata type for the Process Editor
Object Types	"WHOBJECT" on page 182	Base metadata type for SAS/Warehouse Administrator objects
Physical Storage Types	"WHDBMSST" on page 97	Metadata type for DBMS physical stores
	"WHDYNSAS" on page 107	Metadata type for dynamically generated SAS physical stores
	"WHMDDSTR" on page 177	Metadata type for OLAP MDDB physical store
	"WHPHYSTR" on page 204	Base metadata type for physical storage objects
	"WHSASSTR" on page 231	Metadata type for SAS physical data stores
Process Types—Load	"WHLDOMDD" on page 149	Metadata type for OLAP MDDB load processes
	"WHLDOPRX" on page 150	Metadata type for OLAP Proxy load processes
	"WHLDOTBL" on page 152	Metadata type for OLAP table load processes
	"WHLDRDAT" on page 155	Metadata type for data table load processes
	"WHLDRDTL" on page 157	Metadata type for detail table load processes
	"WHLDREXT" on page 159	Metadata type for external file load processes
	"WHLDRIMF" on page 161	Metadata type for InfoMart file load processes
	"WHLDRINF" on page 163	Metadata type for InfoMart item load processes
	"WHLDRLDT" on page 165	Metadata type for detail logical table load processes
	"WHLDRMDB" on page 167	Metadata type for SAS MDDB load processes
	"WHLDRODD" on page 169	Metadata type for ODD load processes
	"WHLDRODT" on page 171	Metadata type for ODT (Data File) load processes
	"WHLDRSUM" on page 173	Metadata type for summary table load processes
	"WHPRCLDR" on page 207	Base metadata type for table load processes
Process Types	"WHCTRNFM" on page 89	Metadata type for column transformations
	"WHPRCMAN" on page 209	Base metadata type for main processes
	"WHPRCMAP" on page 211	Metadata type for data mapping processes
	"WHPRCPST" on page 213	Metadata type for post-load processes
	"WHPRCREC" on page 215	Metadata type for record selector processes
	"WHPRCSPR" on page 217	Base metadata type for subprocesses
	"WHPRCUSR" on page 219	Metadata type for user exit processes
	"WHPRCXFR" on page 221	Metadata type for data transfer processes
	"WHPROCES" on page 223	Base metadata type for processes

	"WHROWSEL" on page 228	Metadata type for a row selector
	"WHSUBSET" on page 246	Metadata type for subsetting processes that are associated with data mappings
Root Metadata Type—SAS/ Warehouse Administrator	"WHROOT" on page 226	Root type for all SAS/Warehouse Administrator metadata types
SAS Library Types	"WHDYNLIB" on page 106	Metadata type for dynamic SAS libraries
	"WHLIBRY" on page 175	Base metadata type for SAS libraries
	"WHREPLIB" on page 225	Metadata type for metadata repositories
Text File Types	"WHDYNSRC" on page 108	Metadata type for dynamically generated source code entries in SAS catalogs
	"WHJOBCAT" on page 143	Metadata type for scheduler catalog source file entries
	"WHJOBFIL" on page 145	Metadata type for scheduler external file entries
	"WHNOTE" on page 179	Metadata type for notes
	"WHSCRFIL" on page 233	Metadata type for SAS/CONNECT script files
	"WHSRCCAT" on page 236	Base metadata type for SAS catalog entry source code files
	"WHTFILE" on page 267	Base metadata type for text files
	"WHTXTCAT" on page 268	Base metadata type for SAS catalog entry text files
	"WHTXTFIL" on page 269	Base metadata type for external text files

Using the Metadata Type Dictionary

In the dictionary, types are listed in alphabetical order. The documentation for each type includes only what is unique for that type. For additional property and usage information, see the documentation for the parent type.

General Identifying Information

The documentation for many types refers to *general identifying information*. This phrase refers to the ID, NAME, and DESC properties. The ID and NAME properties are described under WHROOT. For more detail, see "Identifying Metadata" on page 7.

WHCOLDAT

Metadata type for Data Table columns

Category: Column Types

Parent

"WHCOLUMN" on page 87

Overview

WHCOLDAT models the metadata for data table columns in SAS/Warehouse Administrator. To display these columns with the SAS/Warehouse Administrator Explorer:

1. Select a data table with the right mouse button.
2. Select **`Properties`** from the pop-up menu.
3. Go to the Columns tab.

Properties

The following table lists all of the properties for WHCOLDAT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Format	C	Yes	Yes	No
Id	C	* Req	Yes	No

Indexes	L	No	No	No
Informat	C	Yes	Yes	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Length	N	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Table	L	* Auto supplied	No	No
Type	C	* Req	* Req	No

Using WHCOLDAT

Add	Update	Delete
No	Yes	Yes

WHCOLDAT is a dependent type, like all subtypes of WHCOLUMN. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in “Relationships Among Metadata Types” on page 53.

WHCOLDTL

Metadata type for detail table columns

Category: Column Types

Parent

“WHCOLUMN” on page 87

Overview

WHCOLDTL models the metadata for detail table columns in SAS/Warehouse Administrator. To display these columns with the SAS/Warehouse Administrator Explorer:

1. Select a detail table with the right mouse button.
2. Select **`Properties`** from the pop-up menu.
3. Go to the Columns tab.

Properties

The following table lists all of the properties for WHCOLDTL and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Format	C	Yes	Yes	No
Id	C	* Req	Yes	No
Indexes	L	No	No	No
Informat	C	Yes	Yes	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Length	N	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes

NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Table	L	* Auto supplied	No	No
Type	C	* Req	* Req	No

Using WHCOLDTL

Add	Update	Delete
No	Yes	Yes

WHCOLDTL is a dependent type, like all subtypes of WHCOLUMN. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in "Relationships Among Metadata Types" on page 53.

WHCOLODD

Metadata type for ODD columns

Category: Column Types

Parent

"WHCOLUMN" on page 87

Overview

WHCOLODD models the metadata for operational data definition (ODD) table columns in SAS/Warehouse Administrator. To display these columns with the SAS/Warehouse Administrator Explorer:

1. Select an ODD with the right mouse button.
2. Select **`Properties`** from the pop-up menu.
3. Go to the Columns tab.

Properties

The following table lists all of the properties for WHCOLODD and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Format	C	Yes	Yes	Yes
Id	C	* Req	Yes	No
Indexes	L	No	No	No
Informat	C	Yes	Yes	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Length	N	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Table	L	* Auto supplied	No	No
Type	C	* Req	* Req	No

Using WHCOLODD

Add	Update	Delete
No	Yes	Yes

WHCOLODD is a dependent type, like all subtypes of WHCOLUMN. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in "Relationships Among Metadata Types" on page 53.

WHCOLOLP

Metadata type for OLAP columns

Category: Column Types

Parent

"WHCOLUMN" on page 87

Overview

WHCOLOLP replaces the WHCOLSUM metadata type from Release 1.3. WHCOLOLP models the metadata for OLAP tables, Groups, and MDDBs in the SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHCOLOLP and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Crossings	L	No	No	Yes
Cvalue	C	No	Yes	No
Desc	C	No	Yes	No
Extended Attributes	L	No	Yes	Yes
Hierarchies	L	No	No	No
Format	C	No	Yes	No
Id	C	No	* Req	No
Indexes	L	No	No	No

Informat	C	No	Yes	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Length	N	No	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	* Default	No
Note	L	No	Yes	Yes
NValue	N	No	Yes	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Statistic	C	No	Yes	No
Sort Order	C	No	Yes	No
Summary Role	C	No	Yes	No
Table	L	No	No	No
Type	C	No	Yes	No

New properties for WHCOLOLP are as follows:

CROSSINGS
specifies an SCL list of general identifying information about the crossings that are associated with an OLAP group, Table, or MDDB.

HIERARCHIES
specifies an SCL list of general information about the hierarchies that are associated with an OLAP group, Table, or MDDB.

SORT ORDER
specifies a character string that contains the sort order of the column. Valid values are **`ASCENDING`**, **`DESCENDING`**, **`ASCFORMATTED`**, **`DESFORMATTED`**, and **`DSORDER`**.

STATISTIC
specifies a character string that contains the name of the statistic used to compute this statistic column.

SUMMARY ROLE
specifies a character string that contains the role of the column in the summary data. Valid values are **`CLASS`**, **`STATISTIC`**, **`ID`**, and **`_TYPE_`**.

Using WHCOLOLP

Add	Update	Delete
No	Yes	Yes

WHCOLOLP is a dependent type, like its parent, WHCOLUMN. To understand how all the subtypes of WHCOLOLP relate to other types, see the OLAP Metadata Type Model in “Relationships Among Metadata Types” on page 53.

WHCOLSCL

Metadata type for statistic columns in summary tables and MDDBs

Category: Column Types

Parent

"WHCOLSUM" on page 83

Overview

WHCOLSCL models the metadata for statistic columns in summary tables and MDDBs in SAS/Warehouse Administrator. To display these columns with the SAS/Warehouse Administrator Explorer:

1. Select a summary group with the right mouse button.
2. Select **`Properties`** from the pop-up menu.
3. Go to the Column Roles tab.

The statistic columns in a summary group are shared by all summary tables and MDDBs in the group.

Properties

The following table lists all of the properties for WHCOLSCL and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one. ** Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Alias	C	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes

Format	C	No	No	No
Id	C	No	No	No
Indexes	L	No	No	No
Informat	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Length	N	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Statistic Type	C	No	No	No
Summary Role	C	No	No	No
Table	L	No	No	No
Type	C	No	No	No

WHCOLSCL has the following new property:

STATISTIC TYPE
specifies the character string for the type of statistic. For example: **SUM**, **COUNT**, **AVERAGE**, **MAX**, and **MIN**.

See “WHCOLSUM” on page 83 for a description of the new properties for summary table column types.

Using WHCOLSCL

Add	Update	Delete
No	No	No

WHCOLSCL is a dependent type, like all subtypes of WHCOLUMN. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in “Relationships Among Metadata Types” on page 53.

WHCOLSUM

Base metadata type for columns in summary tables and MDDBs

Category: Column Types

Parent

"WHCOLUMN" on page 87

Overview

WHCOLSUM is a base metadata type for all columns in summary tables and MDDBs in SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHCOLSUM and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Alias	C	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Format	C	No	No	No
Id	C	No	No	No
Indexes	L	No	No	No
Informat	C	No	No	No
Input Objects	L	No	No	No

Input Sources	L	No	No	No
Length	N	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Summary Role	C	No	No	No
Table	L	No	No	No
Type	C	No	No	No

WHCOLSUM has the following two new properties:

ALIAS
specifies a character string for the column name as registered by SAS/Warehouse Administrator.

Note that the NAME for a summary column contains the physically stored name, and the ALIAS contains the name as seen through SAS/Warehouse Administrator. A column's NAME and ALIAS are the same, except for MDDB columns.

For MDDB columns, the NAME property returns the name as it would be returned through the SASSFIO libname engine when looking at a specific hierarchy. The ALIAS property returns the name as seen through SAS/Warehouse Administrator.

SUMMARY ROLE
specifies a character string for the role of the column in the summary data. For example: **`CLASS`**, **`STATISTIC`**, **`ID`**, **`FREQUENCY`**, and **`TIME`**.

Using WHCOLSUM

Add	Update	Delete
No	No	No

WHCOLSUM is a dependent type, like all subtypes of WHCOLUMN. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in "Relationships Among Metadata Types" on page 53.

WHCOLTIM

Metadata type for _LOADTM columns

Category: Column Types

Parent

"WHCOLUMN" on page 87

Overview

WHCOLTIM models the metadata for _LOADTM columns. A _LOADTM column is an optional column that you can specify for SAS/Warehouse Administrator tables. This column contains automatically generated time values that indicate when particular rows of data were loaded into a table. To specify these columns in SAS/Warehouse Administrator:

1. In the Explorer, select a table with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select **`Edit Load Step`**.
5. Go to the Load Options tab.
6. Select (or deselect) **`Add Load Time Column to Table`**.

Properties

The following table lists all of the properties for WHCOLTIM and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Format	C	Yes	Yes	No
Id	C	* Req	Yes	No
Indexes	L	No	No	No
Informat	C	Yes	Yes	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Length	N	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Table	L	* Auto supplied	No	No
Type	C	* Req	* Req	No

Using WHCOLTIM

Add	Update	Delete
No	Yes	Yes

WHCOLTIM is a dependent type, like all subtypes of WHCOLUMN. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in “Relationships Among Metadata Types” on page 53.

WHCOLUMN

Base metadata type for table columns

Category: Column Types

Parent

"WHROOT" on page 226

Overview

WHCOLUMN is the base metadata type for table columns in SAS/WarehouseAdministrator.

Properties

The following table lists all of the properties for WHCOLUMN and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Format	C	No	No	No
Id	C	No	* Req	No
Indexes	L	No	No	No
Informat	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Length	N	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No

Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Table	L	No	No	No
Type	C	No	No	No

New properties for WHCOLUMN are as follows:

FORMAT
specifies a SAS format that is assigned to this column.

INDEXES
specifies an SCL list of general identifying information about the indexes that contain this column.

INFORMAT
specifies a SAS informat that is assigned to this column.

INPUT OBJECTS
represents an SCL list of general identifying information about the objects input to this column. For more details about input objects, see “INPUT and OUTPUT Properties” on page 64.

INPUT SOURCES
represents an SCL list of general identifying information about the sources input to this column. This list must be of type WHCTRNFM, WHCOLUMN, or subtypes of these. The input sources to this column must be appropriately related through a common process. For more details about input sources, see “INPUT and OUTPUT Properties” on page 64.

LENGTH
specifies the value length of this column.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects output from this column. For more details about output objects, see “INPUT and OUTPUT Properties” on page 64.

OUTPUT TARGETS
represents an SCL list of general identifying information about the targets output from this column. The output targets to this column must be appropriately related through a common process. For more details about output targets, see “INPUT and OUTPUT Properties” on page 64.

TABLE
represents an SCL list of general identifying information about the table object to which this column belongs.

TYPE
specifies the type of data that is contained in this column. Valid types are **C** (character) or **N** (numeric).

Using WHCOLUMN

Add	Update	Delete
No	No	No

WHCOLUMN and its subtypes are dependent types. To understand how all subtypes of WHCOLUMN relate to other types, see the column mapping models in "Relationships Among Metadata Types" on page 53.

WHCTRNFM

Metadata type for column transformations

Category: Process Types

Parent

"WHROOT" on page 226

Overview

WHCTRNFM models the metadata for column transformation processes in the SAS/Warehouse Administrator Process Editor. A *column transformation* is a data mapping process in which data from the source column is either mapped one-to-one to a target column or is transformed before it is loaded into the target column. The WHCTRNFM type corresponds to the one–to-one mappings or derived mappings that are defined on the Columns tab of the Mapping Process Properties window. The following is one way to add a derived mapping through the SAS/Warehouse Administrator interface:

1. In the Explorer, select a table with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select

   Add ▶ Inputs
5. Select an input source from the Input Selector window.
6. When the input source and the Mapping box display in the Process Editor, select the Mapping box with the right mouse button, and then select **`Properties`**.
7. Enter the column information, until you come to the Column Mapping tab.
8. Select a column.
9. Click Derive Mapping.
10. Enter the derived mapping information (transformation details).

Properties

The following table lists all of the properties for WHCTRNFM and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	* Req when mapping is one-to-one	Yes	No
Mapping	L	* Auto supplied	No	No
Mapping Type	C	* Req	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No

Source Code	L	No	No	Yes
Source Text	L	Yes for derived mapping only	Yes for derived mapping only	No

New properties for WHCTRNFM are as follows:

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this transformation. Input objects are discussed in "INPUT and OUTPUT Properties" on page 64.

INPUT SOURCES
specifies an SCL list of general identifying information about the objects that are input to this transformation. This list points to the input columns or transforms. If you use a column as the input for a transform, the name of the column's physical table must exist.

Objects are verified to ensure that they are part of the same process. If they are not part of the same process, an error message is produced. Input sources are discussed in "INPUT and OUTPUT Properties" on page 64.

MAPPING
specifies an SCL list of general identifying information about the mapping process to which this transformation belongs.

MAPPING TYPE
indicates the type of column mapping. Possible values are

`ONE TO ONE`—mappings that do not include any transformations.

`DERIVED`—mappings that include transformations.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are output from this transformation. Output objects are discussed in "INPUT and OUTPUT Properties" on page 64.

OUTPUT TARGETS
specifies an SCL list of general identifying information about the targets that are output from this column. This list points to the output column, transform, or row selection object. Objects are verified to ensure that they are part of the same process. If they are not part of the same process, an error message is produced. Only a single WHCOLUMN subtype object can be specified, while multiple WHROWSEL objects can be included. Output targets are discussed in "INPUT and OUTPUT Properties" on page 64.

SOURCE CODE
specifies an SCL list of general identifying information about the source code for this transformation.

SOURCE TEXT
represents an SCL list of character items. Each item can contain a maximum of 200 characters of source code. You can add or update this property for a derived mapping, but it is ignored for a one-to-one mapping.

Using WHCTRNFM

Add	Update	Delete
No	Yes	Yes

WHCTRNFM is a dependent type. To understand how it relates to other types, see the column mapping models in "Relationships Among Metadata Types" on page 53.

WHDATTBL

Metadata type for data tables

Category: Object Types—Explorer

Parent

"WHTABLE" on page 254

Overview

WHDATTBL models the metadata for data tables in SAS/Warehouse Administrator. A *data table* is a multipurpose table. You can use it as a detail data store, a summary data store, a look-up table included as part of a join, or a table that holds information that does not fit anywhere else.

A data table can be a SAS table or view or a DBMS table or view. To add a data table with the SAS/Warehouse Administrator Explorer:

1. Select a data group with the right mouse button.
2. Select

   Add Item ► data table

3. Select the data table with the right mouse button.
4. Select **`Properties`**.
5. Enter the data table information.

Properties

The following table lists all of the properties for WHDATTBL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Resolved View Code	L	No	No	Yes
Table Name	C	Yes	Yes	No

Using Jobs	L	No	No	No
View Code	L	Yes	Yes	Yes

New properties for WHDATTBL are as follows:

RESOLVED VIEW CODE
specifies an SCL list of general identifying information about the source code that is used to view (open) this data table. This property will return a copy of the source code with the *&loc* reference replaced with the appropriate location information. See the *Note* below.

VIEW CODE
specifies an SCL list of general identifying information about the source code that is used to view (open) this data table. This property will return a copy of the source code with the *&loc* reference unresolved. See the *Note* below.

Note: The VIEW CODE and RESOLVED VIEW CODE properties are very closely related. △

When you write the source code to view (open) a data table, you can insert *&loc* into the text as a placeholder for the data table's location information—information such as *libref.catalog.entry.type*, for example. The VIEW CODE property will return a copy of the source code with the *&loc* reference unresolved. The RESOLVED VIEW CODE property will return a copy of the source code with the *&loc* reference replaced with the appropriate location information.

The RESOLVED VIEW CODE property is provided as a convenience and removes the burden from the application of parsing the returned code and replacing the *&loc* reference. If the source code does not contain the *&loc* placeholder, the returned source code is the same for both properties.

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when adding an input source to the table.

Using WHDATTBL

Add	Update	Delete
Yes	Yes	Yes

WHDATTBL is an independent type, like its parent, WHTABLE. To understand how all subtypes of WHTABLE relate to other types, see the table model in "Relationships Among Metadata Types" on page 53.

When you update or add the VIEW CODE property, see "Using WHINFO" on page 134.

You can also use the WHDATTBL type to read Data Mart objects that were created prior to SAS/Warehouse Administrator, Release 1.3.

WHDBMS

Metadata type for DBMS connection definitions

Category: Global Metadata Types

Parent

"WHROOT" on page 226

Overview

WHDBMS models the metadata for a database management system connection definition in SAS/Warehouse Administrator. All warehouses in an environment can share DBMS definitions. In SAS/Warehouse Administrator, to add a DBMS connection definition to the current environment in the Explorer:

1. Select

   File ▶ Setup

   from the pull-down menu.
2. Select **`DBMS Connections`**.
3. Click Add.
4. Enter the connection information.

Properties

The following table lists all of the properties for WHDBMS and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Connection Options	L	Yes	Yes	No
Cvalue	C	Yes	No	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes

Icon	C	Yes	Yes	No
Id	C	No	* Req	No
Libraries	L	No	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Nickname	C	Yes	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Password	C	No	No	No
Tables	L	No	No	No
Userid	C	No	No	No

New properties for WHDBMS are as follows:

CONNECTION OPTIONS
represents an SCL list of options that are needed to access the DBMS in this connection. The SQL sublist contains the options that are needed to access the DBMS through the SQL Pass-Through facility. These are options normally specified in the CONNECT TO statement. The DBLOAD sublist contains statements that are needed to access the DBMS when you use PROC DBLOAD.

LIBRARIES
specifies an SCL list of general identifying information about the libraries that are associated with this DBMS connection.

NICKNAME
specifies the nickname of the DBMS in this connection. Valid nicknames are limited to DB2/AIX, DB2/MVS, Informix, Oracle, SQL Server, and SYBASE.

If a passed nickname is not a known nickname, it is accepted if it is a valid SAS name.

PASSWORD
represents the maximum 200-character string for a password that is registered for this database connection. This property contains the registered password only if the API application is a *secure application.* The only secure applications currently supported are those registered as add-in generators. See the *SAS/Warehouse Administrator User's Guide* for documentation on add-in generators. If the API application is not secure, this property returns a blank value if no password has been registered, and it returns XXXXXXXX if the password has been registered.

TABLES
specifies an SCL list of general identifying information about the tables that are associated with this DBMS connection.

USERID
represents the maximum 200-character string for the user ID that is registered for this database connection. This property contains the registered user ID only if the API application is a *secure application.* The only secure applications currently supported are those that are registered as add-in generators. See the *SAS/Warehouse Administrator User's Guide* for documentation on add-in

generators. If the API application is not secure, this property returns a blank value if no user ID has been registered, and it returns XXXXXXXX if the user ID has been registered.

Using WHDBMS

Add	Update	Delete
Yes	Yes	Yes

WHDBMS is an independent type. To understand how it relates to other types, see the physical storage models in "Relationships Among Metadata Types" on page 53.

Note: The USERID and PASSWORD attributes are only valid with the _GET_METADATA_ method when the API application is a secure application. Currently, the only secure applications are those that are registered as add-in generators. △

WHDBMSST

Metadata type for DBMS physical stores

Category: Physical Storage Types

Parent

"WHPHYSTR" on page 204

Overview

WHDBMSST models the metadata for database management system physical stores in SAS/Warehouse Administrator. To specify DBMS format for a table in a warehouse, from the SAS/Warehouse Administrator Explorer:

1. Select a table with the right mouse button.
2. Select **`Properties`** from the pop-up menu.
3. Go to the Physical Storage tab.
4. Select **`DBMS`** as the storage format.

Properties

The following table lists all of the properties for WHDBMSST and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Database	L	* Req (see Property Dependencies)	No	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Indexes	L	Yes	Yes	Yes
Library	L	* Req (see Property Dependencies)	Yes	No
Load Technique	C	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	Yes	Yes	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Table	L	* Auto supplied	No	No
Table Name	C	Yes	Yes	No
Table Options	L	* Default	* Default	No

Property Dependencies You must define either a database or a library for a physical store instance. If you provide neither, you get an error message. If you provide a library but no database, and a DBMS connection is defined for that library, the value for the DATABASE property will be obtained from the DBMS connection definition. If you

supply both a library and a database, the two properties must match (the library must be appropriate for the database). Otherwise, you get an error message.

New properties for WHDBMSST are as follows:

DATABASE
specifies an SCL list of general identifying information about the database connection that is used for this table.

HOST
specifies an SCL list of general identifying information about the host on which this data is accessed.

LIBRARY
specifies an SCL list of general identifying information about the library that is used to load this database table. If the table is not loaded by using a SAS DBMS Libname engine, then no information is returned for this property.

TABLE OPTIONS
specifies an SCL list of options that are used in creating or loading this table.

The LOAD sublist is appropriate for DBMS tables that are created with code generation level 1.1. It contains any DBLOAD statements that are used to create or load the table.

The CREATE and APPEND sublists are appropriate for DBMS tables that are created with code generation level 2.0. The CREATE sublist contains any SQL options that are used to create the table. The APPEND sublist contains any data set options that are used to load the table. One particularly useful option for the APPEND sublist is the data set option BULKLOAD=, which supports bulk loading of DBMS tables.

Using WHDBMSST

Add	Update	Delete
No	Yes	No

WHDBMSST is a dependent type. To understand how it relates to other types, see the physical storage models in "Relationships Among Metadata Types" on page 53.

WHDETAIL

Metadata type for detail tables

Category: Object Types—Explorer

Parent

"WHTABLE" on page 254

Overview

WHDETAIL models the metadata for detail tables in SAS/Warehouse Administrator. A *detail table* is a detail data store. It can be a SAS table or view or a DBMS table or view. To add a detail table with the SAS/Warehouse Administrator Explorer:

1 Select a detail logical table with the right mouse button.
2 Select **`Add New Table`**.
3 Select the detail table with the right mouse button.
4 Select **`Properties`**.
5 Enter the detail table information.

Properties

The following table lists all of the properties for WHDETAIL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No

Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using

Add	Update	Delete
Yes	Yes	Yes

WHDETAIL is an independent type, like all subtypes of WHTABLE. To understand how WHTABLE and WHDETAIL relate to other types, see the table and column models in “Relationships Among Metadata Types” on page 53.

WHDW

Metadata type for data warehouses

Category: Object Types—Explorer

Parent

“WHOBJECT” on page 182

Overview

WHDW models the metadata for data warehouses in SAS/Warehouse Administrator. A *warehouse* is a grouping element for subjects and data groups. It is the object that is

used to implement a data warehouse or a data mart. In the SAS/Warehouse Administrator Explorer, to add a data warehouse to an environment:

1 Select the environment with the right mouse button.

2 Select

Add Item ► Data Warehouse

3 Select the data warehouse with the right mouse button.

4 Enter the warehouse information.

Properties

The following table lists all of the properties for WHDW and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	Yes	No
Cvalue	C	No	Yes	No
Desc	C	No	Yes	No
Extended Attributes	L	No	Yes	Yes
Group	L	No	Yes	No
Icon	C	No	Yes	No
Id	C	No	* Req	No
Job Info Library	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	* Default	No
Note	L	No	Yes	Yes
NValue	N	No	Yes	No
Owner	L	No	Yes	No

Process Groups	L	No	No	No
Process Members	L	No	No	No

New properties for WHDW are as follows:

JOB INFO LIBRARY
specifies an SCL list of general identifying information about the job information library that is associated with this repository. The job information library is the location where job status information is stored for scheduled jobs.

LIBRARY
specifies an SCL list of general identifying information about the SAS library that contains the metadata for this data warehouse. See the metadata type WHLIBRY for the format of this list.

PROCESS GROUPS
specifies an SCL list of general identifying information about the process groups to which this object belongs.

PROCESS MEMBERS
specifies an SCL list of general identifying information about the process members that belong to this object. The list must be of type WHGRPJOB, WHJOB, WHEVENT.

Using WHDW

Add	Update	Delete
No	Yes	No

WHDW is used with the _SET_SECONDARY_REPOSITORY_ method to access the metadata for a particular data warehouse. To set a secondary repository, you must pass one of these two properties in the *l_meta* list for the _SET_SECONDARY_REPOSITORY_ method:

ID
represents the metadata identifier of the secondary repository.

LIBRARY
allows the stored metadata information to be overridden with the optional information that is specified here.

For details, see "_SET_SECONDARY_REPOSITORY_" on page 43.

WHDW is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

WHDWENV

Metadata type for warehouse environments

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHDWENV models the metadata for warehouse environments in SAS/Warehouse Administrator. An *environment* is a grouping element for warehouses and ODD groups. It is a directory that stores metadata, such as host definitions, that is shared among one or more warehouses and ODD groups.

In the SAS/Warehouse Administrator desktop folder, environments are displayed as icons. In the SAS/Warehouse Administrator Explorer, the environment that you selected from the desktop folder is the top-most object. To add an environment to the SAS/Warehouse Administrator desktop interface:

1. Click in a clear area with the right mouse button.
2. Select

   Add Item ▶ Data Warehouse Environment

3. Enter the environment information.

Properties

The following table lists all of the properties for WHDWENV and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Active Repositories	L	No	No	No
Administrator	L	No	No	No
Cvalue	C	No	Yes	No

Desc	C	No	Yes	No
Extended Attributes	L	No	Yes	Yes
Group	L	No	Yes	No
Icon	C	No	Yes	No
Id	C	No	* Req	No
Job Info Library	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	Yes	Yes
NValue	N	No	Yes	No
Owner	L	No	No	No
Process Groups	L	No	No	No
Process Members	L	No	Yes	No
Repositories	L	No	No	No

New properties for WHDWENV are as follows:

ACTIVE REPOSITORIES
specifies an SCL list of general identifying information about the currently active secondary metadata repositories in this environment.

JOB INFO LIBRARY
specifies an SCL list of general identifying information about the job information library that is associated with this repository. The job information library is the location where job status information is stored for scheduled jobs.

LIBRARY
specifies an SCL list of general identifying information about the SAS library that contains the metadata for this environment.

PROCESS GROUPS
specifies an SCL list of general identifying information about the process groups to which this object belongs.

PROCESS MEMBERS
specifies an SCL list of general identifying information about the process members that belong to this object. The list must be of type WHDW, WHGRPJOB, WHJOB, or WHEVENT. A WHDW object will be rejected if it is not in the current data warehouse. Any attempt to remove the current data warehouse from this property will be ignored.

REPOSITORIES
specifies an SCL list of general identifying information about the metadata repositories in this environment.

Using WHDWENV

Add	Update	Delete
No	Yes	No

WHDWENV is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

WHDWENV is used with the _SET_PRIMARY_REPOSITORY_ method to access the metadata for a warehouse environment. For details, see "_SET_PRIMARY_REPOSITORY_" on page 40.

WHDYNLIB

Metadata type for dynamic SAS libraries

Category: SAS Library Types

Parent

"WHLIBRY" on page 175

Overview

WHDYNLIB models the metadata for dynamic SAS libraries in SAS/Warehouse Administrator. This metadata type corresponds to the default, temporary working directories that are identified on the Output Data tab of the properties window for data mappings, user exits, record selectors, and data transfers.

For example, here is how to display the Output Data tab for a user exit process in the SAS/Warehouse Administrator interface:

1. In the Explorer, select a table with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select

   Add ▶ User Exit

5. Enter the user exit information, until you come to the Output Data tab.

   By default, the fields on the Output Data tab will display a temporary working directory for user exit output.

Properties

WHDYNLIB has the same properties as "WHLIBRY" on page 175. Unlike WHLIBRY properties, however, WHDYNLIB properties cannot be written through the metadata API. They can only be read.

Using WHDYNLIB

Add	Update	Delete
No	No	No

WHDYNLIB is an independent type, like its parent, WHLIBRY. To understand how all subtypes of WHLIBRY relate to other types, see the physical storage models in "Relationships Among Metadata Types" on page 53.

WHDYNSAS

Metadata type for dynamically generated SAS physical stores

Category: Physical Storage Types

Parent

"WHSASSTR" on page 231

Overview

WHDYNSAS models the metadata for dynamically generated SAS physical stores in SAS/Warehouse Administrator. The WHDYNSAS type is a placeholder for tables that do not currently support the definition of both physical information and ACCESS information—tables that do not have a Physical Storage tab in their property windows.

Properties

WHDYNSAS has the same properties as its parent, "WHSASSTR" on page 231. However, unlike WHSASSTR properties, WHDYNSAS properties cannot be written through the metadata API. They can only be read.

Using WHDYNSAS

Add	Update	Delete
No	No	No

WHDYNSAS is a dependent type, like its parent, WHSASSTR. To understand how subtypes of WHSASSTR relate to other types, see the physical storage model for WHSASSTR in "Relationships Among Metadata Types" on page 53.

WHDYNSRC

Metadata type for dynamically generated source code entries in SAS catalogs

Category: Text File Types

Parent

"WHSRCCAT" on page 236

Overview

WHDYNSRC models the metadata for dynamically generated source code catalog entries in SAS/Warehouse Administrator. These entries are generated when you select the

View Code ▶ All

or

Step

option for a table in the Process Editor.

Properties

The following table lists all of the properties for WHDYNSRC and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Entry	C	No	No	No
Extended Attributes	L	No	No	Yes
Full Entry	C	No	No	No
Id	C	No	No	No

Library	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Objects	L	No	No	No

Using WHDYNSRC

Add	Update	Delete
No	No	No

WHDYNSRC is a dependent type, like all of the subtypes of WHTFILE. To understand how all subtypes of WHTFILE relate to other types, see the process model in "Relationships Among Metadata Types" on page 53.

WHEFILE

Metadata type for external file inputs to ODDs

Category: Object Types—Process Editor

Parent

"WHROOT" on page 226

Overview

WHEFILE models the metadata for external file objects in the Process Editor. An *external file* is a file of type other than SAS that is an input to an operational data definition (ODD). Here is one way to add an external file in SAS/Warehouse Administrator:

1. In the Explorer, select an ODD with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the ODD with the right mouse button.
4. Select

   Add ▶ External File

   from the pop-up menu.
5. Select the external file with the right mouse button.

6 Select **Properties**.

7 Enter the external file information.

Properties

The following table lists all of the properties for WHEFILE and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Method	C	Yes	Yes	No
Creating Job	L	Yes	Yes	Yes
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Fileref	C	Yes	Yes	No
Host	L	Yes	Yes	Yes
Icon	C	Yes	Yes	No
Id	C	* Req	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Options	L	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No

Path	L	Yes	Yes	No
Process	L	Yes	Yes	Yes

New properties for WHEFILE are as follows:

ACCESS METHOD
indicates the SAS filename access method specification.

CREATING JOB
specifies a list of general identifying information about the job that creates this file. This list must be a WHJOB or a subtype of WHJOB. A valid CREATING JOB property is required before you can add any INPUT SOURCES. If the CREATING JOB is removed, then any work tables in the chain of INPUT SOURCES will be deleted as well.

FILEREF
represents the fileref that is used to access this file using a SAS filename statement. The maximum length is 8 characters.

HOST
specifies an SCL List of general identifying information about the host on which this file is accessed.

ICON
indicates the catalog entry name of the associated icon. For more information about icons, see "Using Icon Information" on page 69.

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this external file. For more details about input objects, see "INPUT and OUTPUT Properties" on page 64.

INPUT SOURCES
specifies an SCL list of general, identifying information about the sources that are input to this file. For more details about input sources, see "INPUT and OUTPUT Properties" on page 64.

OPTIONS
represents an SCL list of filename statement options. The list contains multiple entries to support options that might be too long to fit in one list item.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are output from this external file. For more details about output objects, see "INPUT and OUTPUT Properties" on page 64.

OUTPUT TARGETS
specifies an SCL list of general identifying information about the targets that are output from this external file. For more details about output targets, see "INPUT and OUTPUT Properties" on page 64.

PATH
indicates an SCL list of host-specific path designations.

PROCESS
specifies an SCL list of general identifying information about the process that created this file.

Using WHEFILE

Add	Update	Delete
Yes	Yes	No

WHEFILE is an independent type.

WHEVENT

Metadata type for events

Category: Object Types—Process Editor

Parent

"WHPOBJCT" on page 205

Overview

WHEVENT models the metadata for an event. An *event* is a metadata record that specifies a condition for controlling a Job, such as checking for certain return codes or verifying the existence of a file. To use events, you must create them, include them in a job flow, and then write a metadata API program that reads the job flow and generates code for it.

Properties

The following table lists all of the properties for WHEVENT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No

Extended Attributes	L	Yes	Yes	Yes
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Process Groups	L	* Req	Yes	No
Process Members	L	No	No	No

New properties for WHEVENT are as follows:

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this event.

INPUT SOURCES
specifies an SCL list of general identifying information about the sources that are input to this event. This list must be of type WHJOB or WHEVENT. Adding an object beneath itself is prevented.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are output from this event.

OUTPUT TARGETS
specifies an SCL list of general identifying information about the targets that are output from this event. This list must be of type WHJOB or WHEVENT. Adding an object beneath itself is prevented.

Using WHEVENT

Add	Update	Delete
Yes	Yes	Yes

WHEVENT is an independent type.

WHEXTATR

Metadata type for extended attributes

Category: Extended Attribute Type

Parent

"WHROOT" on page 226

Overview

WHEXTATR models the metadata for the EXTENDED ATTRIBUTE property in SAS/Warehouse Administrator. Extended attributes store site-defined metadata that is not part of the standard metadata for that object.

For each object that supports the EXTENDED ATTRIBUTE property, you can enter one or more EXTENDED ATTRIBUTE records. Each EXTENDED ATTRIBUTE record has a field for NAME, DESCRIPTION, and VALUE. For example, here is an EXTENDED ATTRIBUTE record for a table that is named Sales Detail Data:

NAME:	Sales Detail Data Web Page
DESCRIPTION:	URL to Web doc for Sales Detail table
VALUE:	http://www.ourserver.com/warehouse1/tables/sales_dd.html

Note: Each EXTENDED ATTRIBUTE record for a given element must have a unique NAME. △

Most SAS/Warehouse Administrator Explorer objects, some columns within objects, and all process objects in the Process Editor (Data Mappings, User Exits, Extractions, and so on) provide access to an EXTENDED ATTRIBUTE property.

In the SAS/Warehouse Administrator interface, to add extended attributes to an Explorer object or a process, display the property window for that object or process, select

File ▶ Extensions

from the pull-down menu, and enter the extended attribute.

In the SAS/Warehouse Administrator interface, to add extended attributes to a table column, display the property window for the table, go the Columns tab, select a column, then select

Edit ▶ Column Extensions

from the pull-down menu, and enter the extended attribute.

Properties

The following table lists all of the properties for WHEXTATR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the

_UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one. ** Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
CValue	C	Yes	No	No
Desc	C	Yes	Yes	No
Extended Attributes	L	No	No	No
Id	C	* Auto supplied	* Req	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	Yes	No	No
Note	L	Yes	No	No
NValue	N	Yes	No	No
Object	L	* Auto supplied	No	No
Type	C	* Default (defaults to "C")	No	No
Value	C	Yes	Yes	No

New Properties for WHEXTATR are as follows:

OBJECT
specifies an SCL list of general identifying information about the object that owns this extended attribute.

TYPE
represents the one-character string that indicates whether the extended attribute is numeric or character.

N — numeric type extended attribute (not supported in this release).

C — character type extended attribute.

VALUE
represents the 200-character string that contains the extended attribute text (such as a URL or file path).

Using WHEXTATR

Add	Update	Delete
No	No	Yes

WHEXTATR is a dependent type.

You can add, update, or delete the EXTENDED ATTRIBUTES property from any type under WHROOT that supports the appropriate method (_ADD_METADATA, and so on).

The EXTENDED ATTRIBUTE property behaves like the COLUMN property list does on the WHTABLE type. You can pass the EXTENDED ATTRIBUTE property with the _ADD_METADATA method that adds the owning object. After the owning object exists, you can add new attributes by using the _UPDATE_METADATA method on the owning object.

To update an existing attribute, send the _UPDATE_METADATA method to the attribute itself. To remove an attribute from an owning object, send the _DELETE_METADATA method to the attribute itself.

Reading EXTENDED ATTRIBUTE If the _GET_METADATA_ method is called on an API object that has an extended attribute, the VALUE property of the extended attribute will be returned even when the *expand* parameter is set to 0 for the _GET_METADATA_ call.

In general, it is good practice to use the SCL UPCASE or LOWCASE functions to read text values, as with the _GET_METADATA_ method. This is especially useful in reading the NAME, DESCRIPTION, and VALUE fields in the EXTENDED ATTRIBUTE property. The text in these fields is stored as the user entered them, and it can be in mixed case.

WHGRPDAT

Metadata type for Data Groups

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHGRPDAT models the metadata for data groups in SAS/Warehouse Administrator. A *data group* is a grouping element for data tables, InfoMarts, and other data groups. To add a data group with the SAS/Warehouse Administrator Explorer:

1 Select a warehouse, a subject, or a parent data group with the right mouse button.

2 Select

Add Item ▶ Data Group

3 Select the data group with the right mouse button.

4 Select **`Properties`**.

5 Enter the data group information.

Properties

The following table lists all of the properties for WHGRPDAT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	No	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owner	L	Yes	Yes	No

Using WHGRPDAT

Add	Update	Delete
Yes	Yes	Yes

WHGRPDAT is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in “Relationships Among Metadata Types” on page 53.

You can also use the WHGRPDAT type to read Data Mart Group objects that were created prior to SAS/Warehouse Administrator, Release 1.3.

WHGRPINF

Metadata type for InfoMarts

Category: Object Types—Explorer

Parent

“WHOBJECT” on page 182

Overview

WHGRPINF models the metadata for InfoMarts in SAS/Warehouse Administrator. An InfoMart is used to organize InfoMart items and InfoMart files. To add an InfoMart with the SAS/Warehouse Administrator Explorer:

1. Select a subject, data group, or an ODD group with the right mouse button.
2. Select

   Add Item ▶ Information Mart

3. Select the information mart with the right mouse button.
4. Select **`Properties`**.
5. Enter the information mart information.

Properties

The following table lists all of the properties for WHGRPINF and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	No	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owner	L	Yes	Yes	No

Using WHGRPINF

Add	Update	Delete
Yes	Yes	Yes

WHGRPINF is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in “Relationships Among Metadata Types” on page 53.

WHGRPJOB

Metadata type for job groups

Category: Object Types—Process Editor

Parent

"WHPOBJCT" on page 205

Overview

WHGRPJOB models the metadata for a job group. A *job* is a metadata record that specifies the processes that create one or more data stores (output tables).

Properties

The following table lists all of the properties for WHGRPJOB and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owner	L	Yes	Yes	No

Process Groups	L	* Req	Yes	No
Process Members	L	No	No	No

Using WHGRPJOB

Add	Update	Delete
Yes	Yes	Yes

WHGRPJOB is an independent type.

WHGRPODD

Metadata type for ODD groups

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHGRPODD models the metadata for operational data definition groups in SAS/Warehouse Administrator. An ODD group is used to organize ODDs and InfoMarts. To add an ODD group with the SAS/Warehouse Administrator Explorer:

1. Select an environment with the right mouse button.
2. Select

   Add Item ▶ Operational Data Definition (ODD) Group

3. Select the ODD group with the right mouse button.
4. Select **`Properties`**.
5. Enter the ODD group information.

Properties

The following table lists all of the properties for WHGRPODD and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owner	L	Yes	Yes	No

Using WHGRPODD

Add	Update	Delete
Yes	Yes	Yes

WHGRPODD is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in “Relationships Among Metadata Types” on page 53.

WHGRPOLP

Metadata type for OLAP groups

Category: Object Types—Explorer

Parent

"WHOLPSTC" on page 198

Overview

WHGRPOLP models the metadata for OLAP groups in the SAS/Warehouse Administrator Explorer. An *OLAP group* is a grouping element for doing HOLAP, ROLAP, MOLAP, or MIXED type processing using OLAP tables or MDDBs. To add an OLAP group with the SAS/Warehouse Administrator Explorer:

1. Select a subject or data group with the right mouse button.
2. Select

   Add Item ▶ OLAP Group

3. Select the OLAP group with the right mouse button.
4. Select **`Properties`**.
5. Enter the OLAP group information.

Properties

The following table lists all of the properties for WHGRPOLP and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes

Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Crossings	L	Yes	Yes	Yes
Cube	L	Yes	Yes	Yes
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
OLAP Type	C	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHGRPOLP

Add	Update	Delete
Yes	Yes	Yes

WHGRPSUM

Metadata type for summary groups

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHGRPSUM models the metadata for summary groups in SAS/Warehouse Administrator. A *summary group* is a grouping element for summary tables or MDDBs. Each table or MDDB in the group uses the summary group's assignments for input data source, column roles, and fiscal time. A summary group also defines the default class variables and analysis variables that are used when you build the dimensions for each summary level within the group. To add a summary group with the SAS/Warehouse Administrator Explorer:

1. Select a subject with the right mouse button.
2. Select

   Add Item ▶ Summary Group

3. Select the summary group with the right mouse button.
4. Select **`Properties`**.
5. Enter the summary group information.

Properties

The following table lists all of the properties for WHGRPSUM and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get

detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Group	L	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Owner	L	No	No	No

Using WHGRPSUM

Add	Update	Delete
No	No	No

WHGRPSUM is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in “Relationships Among Metadata Types” on page 53.

WHHOST

Metadata type for host definitions

Category: Global Metadata Types

Parent

“WHROOT” on page 226

Overview

WHHOST models the metadata for host definitions in SAS/Warehouse Administrator. A *host definition* is a metadata record that specifies a computer where data stores reside, where processes and jobs execute, or where process output is sent. Host definitions are included in the metadata records for data stores, processes, and scheduling server definitions in an environment. In SAS/Warehouse Administrator, to add a host definition to the current environment in the Explorer:

1. Select

   File ► Setup

   from the pull-down menu.
2. Select **`Hosts`**.
3. Click Add.
4. Enter the host information.

Properties

The following table lists all of the properties for WHHOST and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Comamid	C	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Icon	C	Yes	Yes	No
Id	C	No	* Req	No
Locale	C	* Req	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No

Operating System	C	Yes	Yes	No
Processes	L	No	No	No
Remote Address	C	Yes	Yes	No
SAS Version	C	Yes	Yes	No
Script	L	Yes	Yes	Yes
Source Code	L	No	No	Yes
Tables	L	No	No	No
Use Script	C	Yes	Yes	No

New properties for WHHOST are as follows:

COMAMID
indicates the SAS/CONNECT Access Method (comamid) option value that is needed to access this host.

ICON
specifies the catalog entry name of the associated icon. For more information about icons, see “Using Icon Information” on page 69.

LOCALE
indicates the location of this host. Values can be either **LOCAL** or **REMOTE**.

OPERATING SYSTEM
represents the operating system for this host. Valid values are defined by what is available on the Host Options tab of the Host Properties window. Some possible values are **CMS**, **MVS**, **OS/2**, **UNIX**, **VMS**, **Windows**.

PROCESSES
specifies an SCL list of general identifying information about the processes that execute on this host.

REMOTE ADDRESS
represents the remote address of this host.

SAS VERSION
indicates the version of SAS that is running on this host. Valid values are defined by what is available on the Host Options tab of the Host Properties window.

SCRIPT
specifies an SCL list of general identifying information about the SAS/CONNECT script that is associated with this host.

SOURCE CODE
specifies an SCL list of general identifying information about the source code that is needed to access this host.

TABLES
specifies an SCL list of general identifying information about the tables that reside on this host.

USE SCRIPT
specifies whether a SAS/CONNECT SIGNON script is used to connect to a remote host. Valid entries are **NO** (no script is used) or **YES** (a script is used).

Using WHHOST

Add	Update	Delete
Yes	Yes	Yes

WHHOST is an independent type. To understand how WHHOST relates to other types, see the host, process, and physical storage models in "Relationships Among Metadata Types" on page 53.

WHINDEX

Metadata type for indexes that are associated with tables and columns

Category: Index Type

Parent

"WHROOT" on page 226

Overview

WHINDEX models the metadata for SAS indexes that are associated with tables and columns in SAS/Warehouse Administrator. The tables can be in SAS or DBMS format. To specify a SAS index for a table in the SAS/Warehouse Administrator Explorer:

1. Select a table with the right mouse button.
2. Select **`Properties`** from the pop-up menu.
3. Go to the Physical Storage tab.
4. Click Define.
5. Go to the Indexing tab.
6. Enter the index information.

Properties

The following table lists all of the properties for WHINDEX and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that

the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Clustered	N	* Default	No	No
Columns	L	Yes	No	No
Cvalue	C	Yes	No	No
Desc	C	Yes	No	No
Extended Attributes	L	Yes	No	Yes
Host	L	Yes	Yes	No
Id	C	* Req	No	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Req	No	No
Note	L	Yes	No	Yes
NValue	N	Yes	No	No
Options	L	Yes	No	No
Physical Storage	L	* Auto supplied	No	No
Unique	N	* Default	No	No

New properties for WHINDEX are as follows:

CLUSTERED
specifies a numeric value (**0** for No, **1** for Yes) that indicates whether this index is clustered.

COLUMNS
specifies an SCL list of general identifying information about the columns that are involved in this index.

OPTIONS
indicates an SCL list of character strings that contains any options that are entered by the user.

PHYSICAL STORAGE
specifies an SCL list of general identifying information about the physical storage definition to which this index is associated.

UNIQUE
specifies a numeric value (**0** for No, **1** for Yes) that indicates whether this index is a unique index.

Using WHINDEX

Add	Update	Delete
No	No	Yes

To update a WHINDEX:

1. Read the existing index, using _GET_METADATA_ with the *all* and *expand* parameters set to **1**.
2. Change the properties as appropriate in the returned list.
3. Delete the existing index by using the _DELETE_METADATA_ method and the ID of the existing index.
4. Issue an _UPDATE_METADATA_ call to the associated physical storage object, such as the WHINDEX PHYSICAL STORAGE property contents. In the passed *l_meta* list, include the INDEXES property, which has a sublist of the copied WHINDEX metadata list.

WHINDEX is a dependent type. It is dependent on a physical storage definition, such as a subtype of WHPHYSTR.

WHINFO

Metadata type for InfoMart items

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHINFO models the metadata for information mart items (InfoMart items) in SAS/Warehouse Administrator. An *InfoMart item* is an object that contains or displays information that is generated from detail data or summary data in the warehouse. These items are usually SAS charts, reports, graphs, or queries. To add an InfoMart item with the SAS/Warehouse Administrator Explorer:

1. Select an information mart with the right mouse button.
2. Select

   Add Item ▶ Information Mart Item

3 Select the information mart item with the right mouse button.
4 Select **`Properties`**.
5 Enter the information mart item information.

Properties

The following table lists all of the properties for WHINFO and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	Yes
Desc	C	Yes	Yes	No
Entry	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Full Entry	C	No	No	No
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No
Members	L	No	No	No

Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	Yes	Yes	No
Process	L	Yes	Yes	Yes
Resolved View Code	L	No	No	Yes
Using Processes	L	No	No	No
View Code	L	Yes	Yes	Yes

New properties for WHINFO are as follows:

CREATING JOB
specifies a list of general identifying information about the job that creates this InfoMart item. Must be a WHJOB or a subtype of WHJOB. A valid CREATING JOB property is required before you can add any INPUT SOURCES. If the CREATING JOB property is removed, then any work tables in the chain of INPUT SOURCES will be deleted as well.

ENTRY
represents the three-level name of the catalog entry that contains the InfoMart item. An example would be *source.loadfile.source.*

FULL ENTRY
represents the four-level name of the catalog entry that contains the InfoMart item. An example would be *libref.source.loadfile.source*.

HOST
specifies an SCL list of general identifying information about the host on which this InfoMart item is accessed.

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this InfoMart item. For more information about input objects, see “INPUT and OUTPUT Properties” on page 64.

INPUT SOURCES
specifies an SCL list of general identifying information about the sources that are input to this InfoMart item. For more information about input sources, see “INPUT and OUTPUT Properties” on page 64.

LIBRARY
specifies an SCL list of general identifying information about the SAS library that contains this InfoMart item. For details about SAS library metadata, see “WHLIBRY” on page 175.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are output from this InfoMart item. For more information about output objects, see "INPUT and OUTPUT Properties" on page 64.

OUTPUT TARGETS
specifies an SCL list of general identifying information about the targets that are output from this InfoMart item. For more information about output targets, see "INPUT and OUTPUT Properties" on page 64.

PROCESS
specifies an SCL list of general identifying information about the process that created this InfoMart item.

RESOLVED VIEW CODE
specifies an SCL list of general identifying information about the source code that is used to view (open) this InfoMart item. This property will return a copy of the source code with the *&loc* reference replaced with the appropriate location information. See the *Note* below.

VIEW CODE
specifies an SCL list of general identifying information about the source code that is used to view (open) this InfoMart item. This property will return a copy of the source code with the *&loc* reference unresolved. See the *Note* below.

Note: The VIEW CODE and RESOLVED VIEW CODE properties are very closely related. △

When writing the source code to view (open) an InfoMart, you can insert *&loc* into the text as a placeholder for the InfoMart's location information—information such as *libref.catalog.entry.type*, for example. The VIEW CODE property will return a copy of the source code with the *&loc* reference unresolved. The RESOLVED VIEW CODE property will return a copy of the source code with the *&loc* reference replaced with the appropriate location information.

The RESOLVED VIEW CODE property is provided as a convenience and removes the burden from the application of parsing the returned code and replacing the *&loc* reference. If the source code does not contain the *&loc* placeholder, the returned source code is the same for both properties.

Property Dependencies You must define a CREATING JOB in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHINFO

Add	Update	Delete
Yes	Yes	Yes

WHINFO is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

When you update or add the VIEW CODE property, the VIEW CODE source must be in a catalog SOURCE entry. The source will be copied word for word with one blank appended between each word.

There are three ways to specify the source's location. In order of precedence, they are: FULL ENTRY, LIBRARY/ENTRY, and ID.

FULL ENTRY signifies that the passed entry name is currently accessible and should be read as the source code.

LIBRARY/ENTRY signifies that the specified ENTRY name in the specific library should be read as the source code. The LIBRARY property contains a reference to a defined WHLIBRY object. If necessary, the referenced library will be allocated before reading the entry.

ID signifies that the source code exists in an already defined source code catalog object (WHSRCCAT), whose ID is passed. In this scenario, the library that is associated with the passed source code object will be allocated, if necessary. To get the ID for the existing VIEW CODE, you must issue a _GET_METADATA_ call for the WHINFO type's VIEW CODE property.

WHINFOFL

Metadata type for InfoMart files

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHINFOFL models the metadata for an information mart file. An *InfoMart file* is a metadata record that specifies the location of a file and the technique for opening that file. It is used to specify a file other than a SAS file that you want to register in a SAS/Warehouse Administrator environment. The file can be a spreadsheet, an HTML report, or any file that you can using an external application.

To add an InfoMart file with the SAS/Warehouse Administrator Explorer:

1. Select an information mart with the right mouse button.
2. Select

   Add Item ▶ Information Mart File

3. Select the information mart file with the right mouse button.
4. Select **`Properties`**.
5. Enter the information mart file information.

Properties

The following table lists all of the properties for WHINFOFL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	Yes
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
File Type	C	Yes	Yes	No
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Location	C	Yes	Yes	No
Members	L	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Process	L	Yes	Yes	Yes
Resolved View Code	L	No	No	Yes
View Code	L	Yes	Yes	Yes

New properties for WHINFOFL are as follows:

CREATING JOB
specifies a list of general identifying information about the job that creates this InfoMart file. This list must be of type WHJOB or a subtype of WHJOB. A valid CREATING JOB property is required before you can add any INPUT SOURCES. If the CREATING JOB property is removed, then any work tables in the chain of INPUT SOURCES will be deleted as well.

FILE TYPE
indicates a character string that describes the type of file that is being defined. The file can be a spreadsheet, an HTML report, or any file that you can by an application other than SAS. Maximum 40 characters.

LOCATION
indicates a character string that identifies the location of an InfoMart file. Maximum 200 characters.

RESOLVED VIEW CODE
specifies an SCL list of general identifying information about the source code that is used to view (open) this InfoMart item. This property will return a copy of the source code with the *&loc* reference replaced with the appropriate location information. See the *Note* below.

VIEW CODE
specifies an SCL list of general identifying information about the source code that is used to view (open) this InfoMart item. This property will return a copy of the source code with the *&loc* reference unresolved. See the *Note* below.

Note: The VIEW CODE and RESOLVED VIEW CODE properties are closely related. △

When you write the source code to view (open) an InfoMart, you can insert *&loc* into the text as a placeholder for the InfoMart's location information—information such as *libref.catalog.entry.type*, for example. The VIEW CODE property will return a copy of the source code with the *&loc* reference unresolved. The RESOLVED VIEW CODE property will return a copy of the source code with the *&loc* reference replaced with the appropriate location information.

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHINFOFL

Add	Update	Delete
Yes	Yes	Yes

WHINFOFL is an independent type, like all subtypes of WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

When update or add the VIEW CODE property, see "Using WHINFO" on page 134.

WHJOB

Metadata type for jobs

Category: Object Types—Process Editor

Parent

"WHPOBJCT" on page 205

Overview

WHJOB models the metadata for a job. A *job* is a metadata record that specifies the processes that create one or more data stores (output tables).

Properties

The following table lists all of the properties for WHJOB and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Actual End Date	C	No	No	No
Actual Start Date	C	No	No	No
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
External Job ID	C	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Input Tables	L	No	No	No

Job ID	C	No	No	No
Job Type	C	Yes	Yes	No
List	L	No	No	Yes
Log	L	No	No	Yes
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Tables	L	Yes	Yes	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Process Groups	L	* Req	Yes	No
Process Members	L	No	No	No
Recurring Month Days	C	Yes	Yes	No
Recurring Months	C	Yes	Yes	No
Recurring Week Days	C	Yes	Yes	No
Responsibility	C	Yes	Yes	No
Return Code	N	No	No	No
Run Command	C	Yes	Yes	No
Scheduled Start Date	C	See Property Dependencies	See Property Dependencies	No
Scheduling Server	L	Yes	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Status	C	No	No	No
Step Source Code	L	No	No	Yes
Sysin	L	No	No	Yes
Tracking	N	Yes	Yes	No
Tracking Epilog	L	Yes	Yes	Yes
Tracking Prolog	L	Yes	Yes	Yes
User Epilog	L	Yes	Yes	Yes
Userpe	N	Yes	Yes	No
User Prolog	L	Yes	Yes	Yes

New properties for WHJOB are as follows:

ACTUAL END DATE
indicates the SAS datetime when the job actually ended (a character value that is formatted with the SAS DATETIME. format) or N/A if not available. This read-only property is set by job tracking code and job scheduling code.

ACTUAL START DATE
indicates the SAS datetime when the job actually started (a character value that is formatted with the SAS DATETIME. format) or N/A if not available. This read-only property is set by job tracking code and job scheduling code.

EXTERNAL JOB ID
indicates an optional 8-character field that can contain a value that uniquely identifies the job. This read-only property is set by job tracking code and job scheduling code.

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this job.

INPUT SOURCES
specifies an SCL list of general identifying information about the sources that are input to this job. This list must be of type WHJOB or WHEVENT. Adding an object beneath itself is prevented.

INPUT TABLES
specifies a list of general identifying information about the tables that are used in this job to create the output tables. These tables are not created by this job; they are the inputs to the tables that are created by this job.

JOB ID
represents a unique identifier that is assigned by the Job Scheduler utility.

JOB TYPE
indicates how often the job runs. Valid values are **ONCE**, **DAILY**, **WEEKLY**, and **MONTHLY**.

LIST
specifies the location of the job list file. This property is a WHJOBFIL object. This read-only property is set by job tracking code and job scheduling code.

LOG
indicates the location of the job log file. This property is a WHJOBFIL object. This read-only property is set by job tracking code and job scheduling code.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are output from this job. This list points to successor jobs.

OUTPUT TABLES
specifies a list of general identifying information about the tables that are created by this job. This list must be of type WHTABLE, WHINFO, WHINFOFL, WHSUMDDB, WHEFILE, or subtypes of these.

OUTPUT TARGETS
specifies an SCL list of general identifying information about the targets that are output from this job. This list must be of type WHJOB or type WHEVENT.

RECURRING MONTH DAYS
applies only to monthly jobs. An 85-character string indicates what day(s) in the month a job runs. Valid values are a list of integers from 1 through 31, delimited

with a comma or a semicolon. At least one integer is required. No duplicates are permitted. The string is converted to a comma-delimited list with no embedded blanks.

RECURRING MONTHS
applies only to monthly jobs. A 30-character string indicates what months a job runs, where 1=January and 12=December. Valid values are a list of integers from 1 through 12, delimited with a comma or a semicolon. At least one integer is required. No duplicates are permitted. The string is converted to a comma-delimited list with no embedded blanks.

RECURRING WEEK DAYS
applies only to weekly jobs. A 15-character string indicates what weekday(s) a job runs, where 0=Sunday and 6=Saturday. Valid values are a list of integers from 0 through 6, delimited with a comma or a semicolon. At least one integer is required. No duplicates are permitted. The string is converted to a comma-delimited list with no embedded blanks.

RESPONSIBILITY
specifies the character string that indicates who is currently responsible for the creation of the code that is associated with this process.

SAS indicates that SAS/Warehouse Administrator is creating this code dynamically based on the current metadata. **USER** indicates that the user has written the code for this process and is responsible for it.

RETURN CODE
specifies a numeric variable that indicates the return code from the job or N/A if not available. This read-only property is set by job tracking code and job scheduling code.

RUN COMMAND
indicates a 200-character string that contains the command that is issued to run the job.

SCHEDULED START DATE
represents the SAS datetime when the job is scheduled to start. (A character value that is formatted with a SAS DATETIME. format.)

SCHEDULING SERVER
indicates the scheduling server that this job runs on. This property is a subtype of WHSERV, such as WHSRVAT.

SOURCE CODE
specifies an SCL list of general identifying information about the source code for this process. This source code is the same as is seen when selecting

View Code ▶ All

in the SAS/Warehouse Administrator Process Editor.

The source code information that is returned here will be that of a temporary working location of a copy of the source code and might be different for each request for this information.

SOURCE FILE
specifies an SCL list of general identifying information about any user-registered code for a process. This list must be of type WHSRCCAT or a subtype of WHSRCCAT. However, WHJOBCAT or any subtype of WHJOBCAT will be rejected. For process steps that consist of user-written code, this property returns the registered source code location. For process steps that consist of code that is generated by SAS/Warehouse Administrator, this property will return an empty list.

STATUS
represents a 12-character string that indicates the status of the job. Valid values are a blank, **SCHEDULED**, **RUNNING**, **COMPLETE**, or **N/A** if not available. This read-only property is set by job tracking code and job scheduling code.

STEP SOURCE CODE
specifies an SCL list of general identifying information about the source code of the individual step in the process. This source code is the same as is seen when selecting

View Code ▶ Process

in the SAS/Warehouse Administrator Process Editor.

The source code information that is returned here will be that of a temporary working location of a copy of the source code and therefore might be different for each request for this information.

SYSIN
indicates the location of the job sysin file. This property is a WHJOBFIL object. This read-only property is set by job tracking code and job scheduling code.

TRACKING
enables or disables code generation for tracking prologs and epilogs. Values are
-1 — default to server definition
0 — disable
1 — enable
The default value for this property is **-1** (default to server definition).

TRACKING EPILOG
indicates the location of the tracking epilog, which is given to the Job Scheduler by the user. This property returns a WHJOBCAT object. The tracking epilog is appended to the input source code in order to update the job information file with the job completion information.

TRACKING PROLOG
indicates the location of the tracking prolog, which is given to the Job Scheduler by the user. This property returns a WHJOBCAT object.

USER EPILOG
indicates the location of the user epilog, which is given to the Job Scheduler by the user. This property returns a WHJOBCAT object.

USERPE
enables or disables code generation for user prologs and epilogs. Values are:
0 — disable
1 — enable
The default value for this property is *0* (disable).

USER PROLOG
indicates the location of the user prolog, which is given to the Job Scheduler by the user. This property returns a WHJOBCAT object.

Property Dependencies

If the JOB TYPE property is blank, then SCHEDULED START DATE is ignored. If the JOB TYPE is non-blank, then a valid SCHEDULED START DATE is required.

Using WHJOB

Add	Update	Delete
Yes	Yes	Yes

WHJOB is an independent type. To understand how jobs relate to other types, see the diagram on the foldout in Appendix 2.

WHJOBCAT

Metadata type for scheduler catalog source file entries

Category: Text File Types

Parent

"WHSRCCAT" on page 236

Overview

The WHJOBCAT type models the metadata for SAS catalog entries that are defined for jobs, such as entries for user-supplied source code, tracking options, and user-defined prologs and epilogs.

Properties

The following table lists all the properties for WHJOBCAT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Entry	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Full Entry	C	No	No	No
Id	C	* Req	* Req	No
Job Role	C	* Auto supplied	No	No
Library	L	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Objects	L	No	No	No
Responsibility	C	Yes	Yes	No

New properties for WHJOBCAT are as follows:

JOB ROLE
indicates the role that this object serves for the job. The possible values are **TRACKING PROLOG**, **TRACKING EPILOG**, **USER PROLOG**, and **USER EPILOG**.

RESPONSIBILITY
specifies the character string that indicates who is currently responsible for the creation of the code that is associated with this process. Possible values are **SAS** or **USER**.

SAS indicates that SAS/Warehouse Administrator is creating this code dynamically, based on the current metadata. **USER** indicates that the user has written the code for this process and is responsible for it.

Using WHJOBCAT

Add	Update	Delete
No	Yes	Yes

WHJOBCAT is a dependent type, like all subtypes of WHSRCCAT. To understand how all subtypes of WHSRCCAT relate to other types, see the process model in “Relationships Among Metadata Types” on page 53.

Use of _DELETE_METADATA for this type deletes SAS/Warehouse Administrator metadata, not the corresponding catalog entries.

WHJOBFIL

Metadata type for scheduler external file entries

Category: Text File Types

Parent

"WHTXTFIL" on page 269

Overview

The WHJOBFIL type models the metadata for all external file entries that are described to the Job Scheduler utility. Properties for this type are set when you schedule a job through SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHJOBFIL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Id	C	No	No	No
Job Role	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No

Objects	L	No	No	No
Remote	N	No	No	No

WHJOBFIL has the following new properties:

JOB ROLE
specifies the role this object serves for the job. The possible values are **LOG**, **LIST**, **SYSIN**, or **SOURCE**.

REMOTE
indicates a Boolean that states whether the object is remote to the metadata: **1**=Remote, **0**=Local.

Using WHJOBFIL

Add	Update	Delete
No	No	No

WHJOBFIL is a dependent type, like all subtypes of WHTXTFIL. To understand how all subtypes of WHTXTFIL relate to other types, see the process model in "Relationships Among Metadata Types" on page 53.

WHLDETL

Metadata type for detail logical tables

Category: Object Types—Explorer

Parent

"WHTABLE" on page 254

Overview

WHLDETL models the metadata for detail logical tables in SAS/Warehouse Administrator. A *detail logical table* is a multipurpose table that you can use as a detail table, a grouping element for detail tables, or a view on multiple that is related detail tables. If you use it as a grouping element, a detail logical table defines columns that are shared by all of its detail tables.

To add a detail logical table with the SAS/Warehouse Administrator Explorer:

1. Find a subject that does not already have a detail logical table (each subject can only have one).
2. Select the subject with the right mouse button.
3. Select

   Add Item ▶ Detail Logical Table

4 Select the detail logical table with the right mouse button.
5 Select **Properties**.
6 Enter the detail logical table information.

Properties

The following table lists all of the properties for WHLDETL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No

Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Property Dependencies You must define a CREATING JOB property in order to add INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHLDETL

Add	Update	Delete
Yes	Yes	Yes

WHLDETL is an independent type, like all subtypes of WHTABLE. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

The table that is created as part of the _ADD_METADATA_ method is added as a member to only the first group listed in the GROUP property list. Once you add a table, the GROUP property cannot be changed using the metadata API.

Adding and Linking Detail Tables SAS/Warehouse Administrator supports the linking of detail tables (tables of type WHDETAIL) between multiple subjects in a warehouse. Currently, however, the metadata API does not support the concept of linking detail tables to multiple subjects.

Deleting and Unlinking Detail Tables You can use the _DELETE_METADATA_ method to unlink a table from a list of subjects or delete the table entirely. The determination of the type of delete to perform is based on the presence and value of the GROUP property in the *l_meta* list that is passed to the _DELETE_METADATA_ method.

If the GROUP property is not passed in the *l_meta* list or an empty list is passed as the value of the GROUP property, then the table will be deleted entirely. If the GROUP property is passed as a nonempty list in the *l_meta* list, the table will be unlinked from all groups that are referenced in the GROUP property list. If an invalid GROUP identifier is passed in this list, an error is returned to the application and the table is not unlinked from any of the referenced groups.

WHLDOMDD

Metadata type for OLAP MDDB load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDOMDD models the metadata for OLAP MDDB load processes in the SAS/Warehouse Administrator Process Editor.

Properties

The following table lists all of the properties for WHLDOMDD and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No

Load Options	L	Yes (See Property Dependencies)	Yes (See Property Dependencies)	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDOMDD

Add	Update	Delete
No	Yes	No

WHLDOMDD is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in “Relationships Among Metadata Types” on page 53.

For details about reading process information, see “Reading Process Flow Metadata” on page 62.

WHLDOPRX

Metadata type for OLAP Proxy load processes

Category: Process Types—Load

Parent

"WHLDOMDD" on page 149

Overview

WHLDOPRX models the metadata for OLAP Proxy load processes in the SAS/Warehouse Administrator Process Editor.

Properties

The following table lists all of the properties for WHLDOPRX and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one. ** Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes (See Property Dependencies)	Yes (See Property Dependencies)	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No

Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **`Yes`**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDOPRX

WHLDOPRX is valid for the following metadata API write methods:

Add	Update	Delete
No	Yes	No

WHLDOPRX is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in “Relationships Among Metadata Types” on page 53.

For details about reading process information, see “Reading Process Flow Metadata” on page 62.

WHLDOTBL

Metadata type for OLAP table load processes

Category: Process Types—Load

Parent

“WHPRCLDR” on page 207

Overview

WHLDOTBL models the metadata for OLAP table load processes in the SAS/Warehouse Administrator Process Editor.

Properties

The following table lists all of the properties for WHLDOTBL and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes (See Property Dependencies)	Yes (See Property Dependencies)	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No

Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

The properties for WHLDOTBL are the same as for WHPRCLDR, with one exception—WHLDOTBL overrides the LOAD OPTIONS property.

LOAD OPTIONS
indicates an SCL list of options for the specified Load process. For WHLDOTBL, the LOAD OPTIONS property includes the SINGLE PASS item.

- GENERATION LEVEL (**`1.1`** or **`2.0`**) selects the release level of the code that SAS/Warehouse Administrator will generate for the specified Load process. Each level has specific options that it can support, as well as other characteristics.
- LOAD TIME COLUMN (**`YES`** or **`NO`**) indicates whether a Load Time Column will be added to the table that is being loaded by the specified process.

  Note: See Property Dependencies. △
- DROP INDEXES (**`YES`** or **`NO`**) specifies that you should drop (remove) any existing indexes on the table to be loaded before loading the data into the table. Based on the metadata definitions, the appropriate indexes will be recreated after loading the data. This option is useful when updating the indexes during loading is too slow.
- SINGLE PASS (**`YES`** or **`NO`**) specifies that when this item is set to **`YES`**, data for all crossings will be produced by a single PROC SUMMARY step for maximum performance when you use code that is generated by SAS/Warehouse Administrator. When this item is set to **`NO`**, data for each crossing will be produced by a separate PROC SUMMARY step in order to minimize memory utilization, which can be important in systems with memory size restrictions.
- TRUNCATE TABLE (**`YES`** or **`NO`**) specifies that when refreshing the data in a table, the table should be truncated (all data rows are removed but the table is not) instead of completely dropping the table and recreating it from scratch. This option is useful when the table has many options, privileges, and other characteristics defined in the database.
- UNION MULTIPLE INPUTS (**`YES`** or **`NO`**) specifies that any multiple inputs to the current Load process will be unioned together before loading the table. A union is identical to a SET statement in a SAS DATA step that contains multiple input table designations.

In the SAS/Warehouse Administrator interface, LOAD OPTIONS are specified on the Load Options tab of the Load process attributes window for a given data store. Here are some example return values for an OLAP table:

```
LOAD OPTIONS=(     GENERATION LEVEL='2.0'
                   LOAD TIME COLUMN='NO'
                   UNION MULTIPLE INPUTS='YES'
                   DROP INDEXES='NO'
                   TRUNCATE TABLE='NO'
                   SINGLE PASS='YES'
               )
```

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and/or STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDOTBL

Add	Update	Delete
No	Yes	No

WHLDOTBL is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRDAT

Metadata type for data table load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRDAT models the metadata for data table load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add a data table load process in SAS/Warehouse Administrator:

1. In the Explorer, select a data table with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select **Edit Load Step**.
5. Enter the process information.

Properties

The following table lists all of the properties for WHLDRDAT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto suppplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes (See Property Dependencies)	Yes (See Property Dependencies)	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes

Source File	L	Yes	Yes	Yes
Step Source Code	L	Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and/or STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRDAT

Add	Update	Delete
No	Yes	No

WHLDRDAT is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRDTL

Metadata type for detail table load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRDTL models the metadata for detail table load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add a detail table load process in SAS/Warehouse Administrator:

1. In the Explorer, select a detail table with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select **Edit Load Step**.
5. Enter the process information.

Properties

The following table lists all of the properties for WHLDRDTL and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes (See Property Dependencies)	Yes (See Property Dependencies)	No
Metadata Created	C	* Auto suplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes

Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and/or STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRDTL

Add	Update	Delete
No	Yes	No

WHLDRDTL is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDREXT

Metadata type for external file load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDREXT models the metadata for external file load processes in the SAS/Warehouse Administrator Process Editor. An *external file* is an input to an operational data definition (ODD) that extracts information from one or more sources that are not in SAS format. Here is one way to add an external file load process in SAS/Warehouse Administrator:

1. In the Explorer, select an ODD with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the ODD with the right mouse button.
4. Select

Add ▶ External File

from the pop-up menu.

5 Select the external file with the right mouse button.

6 Select **`Edit Load Step`**.

7 Enter the process information.

Properties

The following table lists all of the properties for WHLDREXT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No

Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDREXT

Add	Update	Delete
No	Yes	No

WHLDREXT is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRIMF

Metadata type for InfoMart file load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRIMF models the metadata for information mart file (InfoMart file) load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add InfoMart file load processes in SAS/Warehouse Administrator:

1 In the Explorer, select an information mart file with the right mouse button.

2 Select **Process** from the pop-up menu.

3 In the Process Editor, select the information mart file with the right mouse button.

4 Select **Edit Load Step**.

5 Enter the process information.

Properties

The following table lists all of the properties for WHLDRIMF and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No

Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRIMF

Add	Update	Delete
No	Yes	No

WHLDRIMF is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRINF

Metadata type for InfoMart item load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRINF models the metadata for information mart item (InfoMart item) load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add InfoMart item load processes in SAS/Warehouse Administrator:

1. In the Explorer, select an information mart item with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the information mart item with the right mouse button.

4 Select **`Edit Load Step`**.
5 Enter the process information.

Properties

The following table lists all of the properties for WHLDRINF and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No

Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRINF

Add	Update	Delete
No	Yes	No

WHLDRINF is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRLDT

Metadata type for detail logical table load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRLDT models the metadata for detail logical table load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add a detail logical table load process in SAS/Warehouse Administrator:

1. In the Explorer, select a detail logical table with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select **`Edit Load Step`**.
5. Enter the process information.

Properties

The following table lists all of the properties for WHLDRLDT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes

Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRLDT

WHLDRLDT is valid for the following metadata API write methods:

Add	Update	Delete
No	Yes	No

WHLDRLDT is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRMDB

Metadata type for SAS MDDB load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRMDB models the metadata for SAS MDDB (multidimensional database) load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add an MDDB load process in SAS/Warehouse Administrator:

1. In the Explorer, select an MDDB with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the MDDB with the right mouse button.
4. Select **`Edit Load Step`**.
5. Enter the process information.

Properties

The following table lists all of the properties for WHLDRMDB and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see “Using _UPDATE_METADATA_” on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see “Using _UPDATE_METADATA_” on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	No	No	No
Id	C	No	No	No
Load Options	L	No	No	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	No	No
Source Code	L	No	No	Yes
Source File	L	No	No	Yes
Step Source Code	L	Yes	Yes	Yes

Stepo Source Code	L	*Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRMDB

Add	Update	Delete
No	No	No

WHLDRMDB is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRODD

Metadata type for ODD load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRODD models the metadata for operational data definition (ODD) table load processes in the SAS/Warehouse Administrator Process Editor. An ODD is a metadata record that provides access to data stores. Here is one way to add an ODD load process in SAS/Warehouse Administrator:

1. In the Explorer, select an ODD with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the ODD with the right mouse button.
4. Select **`Edit Load Step`**.
5. Enter the process information.

Properties

The following table lists all of the properties for WHLDRODD and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see “Using _UPDATE_METADATA_” on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see “Using _UPDATE_METADATA_” on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes

Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRODD

WHLDRODD is valid for the following metadata API write methods:

Add	Update	Delete
No	Yes	No

WHLDRODD is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRODT

Metadata type for ODT (Data File) load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRODT models the metadata for operational data table (ODT) load processes in the SAS/Warehouse Administrator Process Editor. An ODT is a SAS table that is an input to an operational data definition in the Process Editor. In the Process Editor, the ODT is called a *data file*. Here is one way to add an ODT load process in SAS/Warehouse Administrator:

1. In the Explorer, select an ODD with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the ODD with the right mouse button.
4. Select

   Add ▶ Data File

5. Select the data file with the right mouse button.

6 Select **`Edit Load Step`**.

7 Enter the process information.

Properties

The following table lists all of the properties for WHLDRODT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No

Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRODT

WHLDRODT is valid for the following metadata API write methods:

Add	Update	Delete
No	Yes	No

WHLDRODT is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLDRSUM

Metadata type for summary table load processes

Category: Process Types—Load

Parent

"WHPRCLDR" on page 207

Overview

WHLDRSUM models the metadata for summary table load processes in the SAS/Warehouse Administrator Process Editor. Here is one way to add a summary table load process in SAS/Warehouse Administrator:

1. In the Explorer, select a summary table with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the summary table with the right mouse button.
4. Select **`Edit Load Step`**.
5. Enter the process information.

Properties

The following table lists all of the properties for WHLDRSUM and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	* Auto supplied	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Load Options	L	Yes	No	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Auto supplied	No	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	No	No	No
Responsibility	C	* Default	Yes	No
Source Code	L	No	No	Yes

Source File	L	Yes	Yes	Yes
Step Source Code	L	* Auto supplied	No	Yes
Subprocesses	L	Yes	Yes	Yes

Property Dependencies When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and STEP SOURCE CODE properties. You can add load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHLDRSUM

Add	Update	Delete
No	No	No

WHLDRSUM is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHLIBRY

Metadata type for SAS libraries

Category: SAS Library Types

Parent

"WHROOT" on page 226

Overview

WHLIBRY is the metadata type for SAS libraries in SAS/Warehouse Administrator. A *SAS library definition* is a metadata record for a SAS library that contains data, views, source code, or other information that is used in the current Warehouse environment.

Properties

The following table lists all of the properties for WHLIBRY and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Database	L	Yes	Yes	No
DBMS Libname	N	Yes	Yes	No
Desc	C	Yes	Yes	No
Engine	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Libref	C	* Req	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Options	L	Yes	Yes	No
Path	L	Yes	Yes	No
Preassigned	N	Yes	Yes	No
Tables	L	No	No	No

New properties for WHLIBRY are as follows:

DATABASE
specifies an SCL list of general identifying information about the Database Connection profile that is used by this library to access the DBMS.

DBMS LIBNAME
indicates a flag that is set to **0** (No) or **1** (Yes) to specify whether the library is a DBMS connection library.

ENGINE
indicates the SAS libname engine specification for this repository.

ICON
specifies the catalog entry name of the associated icon. For more information about icons, see "Using Icon Information" on page 69.

LIBREF
specifies the libref to assign to the metadata repository.

OPTIONS
specifies an SCL list of libname statement options. For a DBMS connection library, the list includes SQL options, USERID, PASSWORD, and other options that are required for the connection.

This property contains the registered user ID or password only if the API application is a secure application. The only secure applications that are currently supported are those registered as add-in generators. See the *SAS/Warehouse Administrator User's Guide* for documentation on add-in generators. If the API application is not secure, this property returns a blank value if no password has been registered, and it returns XXXXXXXX if the password has been registered.

PATH
specifies an SCL list of host-specific path designations. If the list contains more than one entry, then it is assumed that the libname is a concatenated libname and that each list entry is a directory in the concatenation.

PREASSIGNED
specifies the numeric indicator that states whether this libname is preassigned. It has a possible value of `0` (needs to be assigned) or `1` (is already assigned).

TABLES
specifies an SCL list of general identifying information about the tables that are registered as residing in this library.

Using WHLIBRY

Add	Update	Delete
Yes	Yes	Yes

WHLIBRY is an independent type. To understand how it relates to other types, see the physical storage models in "Relationships Among Metadata Types" on page 53.

WHMDDSTR

Metadata type for OLAP MDDB physical store

Category: Physical Storage Types

Parent

"WHSASSTR" on page 231

Overview

WHMDDSTR models the metadata for OLAP MDDB physical data stores in SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHMDDSTR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	No	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Indexes	L	No	No	Yes
Library	L	Yes	Yes	No
Load Technique	C	Yes	Yes	No
Metadata Created	C	* Auto	No	No
Metadata Updated	C	* Auto	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Table	L	* Auto supplied	No	No
Table Name	C	Yes	Yes	No
Table Options	L	* Default	* Default	No

Using WHMDDSTR

Add	Update	Delete
No	Yes	No

WHMDDSTR is a dependent type, like its parent, WHSASSTR.

WHNOTE

Metadata type for notes

Category: Text File Types

Parent

"WHTXTCAT" on page 268

Overview

WHNOTE models the metadata for notes in SAS/Warehouse Administrator. *Notes* are user-entered descriptions of objects, columns, or processes. In SAS/Warehouse Administrator, to add a note to an item, display the properties window for that item, go to the General tab or the Columns tab and click the Notes button. Notes can include any information that is useful to your organization, such as a description of the purpose of an item.

Properties

The following table lists all of the properties for WHNOTE and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	No	Yes	No
Entry	C	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Extended Attributes	L	Yes	Yes	Yes
Full Entry	C	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Id	C	Yes	Yes	No
Library	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	Yes	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No

Property Dependencies When you add a note, you must specify entry information. This can be done in two ways:

- Specify the LIBRARY and ENTRY properties.
- Specify the FULL ENTRY property.

Using WHNOTE

Add	Update	Delete
No	Yes	Yes

WHNOTE is a dependent type. To understand how it relates to other types, see the general information model in “Relationships Among Metadata Types” on page 53.

Use of _DELETE_METADATA for this type deletes SAS/Warehouse Administrator metadata, not the corresponding note.

Reading Notes

When you pass a WHNOTE object to the _GET_METADATA_ method, the method returns a copy of the note, not the actual note in the metadata. The copy is returned in a SAS catalog entry, normally in the WORK library. Because this is a copy, any modifications made to the contents of the catalog entry will not affect the actual note in the metadata.

Here is example WHNOTE code:

```
 *  Get the Note property of the object
 *  whose id is object_id.
 */
l_notemeta=makelist();
l_notemeta=setnitemc(l_notemeta,object_id,'ID');

l_note=makelist();
l_notemeta=insertl(l_notemeta,l_note,-1,'NOTE');

call send(i_api,'_GET_METADATA_',rc,l_notemeta);
if rc = 0 then do;

   /*
    *  Get the details of the Note if it has one.
    */

   if listlen(l_note) > 0 then do;
          call send(i_api,'_GET_METADATA_',rc,l_note,1);

      if rc = 0 then do;
         /*
          *  Get the Details of the Library that
          *  contains the copy of the Note.
          */

         l_notelib=getniteml(l_note,'LIBRARY');

         call send(i_api,'_GET_METADATA_',
                     rc,l_notelib,1);
         if rc = 0 then do;
            libref=getnitemc(l_notelib,'LIBREF');
            entry=getnitemc(l_note,'ENTRY');

              /*
              *  Code here to assign library if
              */  needed.

            end; /* if */
          end; /* if */

       end; /* if */

    end; /* if */
```

Updating Notes

To modify the contents of a note, pass the corresponding metadata ID and WHNOTE type ID to the _UPDATE_METADATA_ method. A combination of the LIBRARY, ENTRY, and FULL ENTRY properties are required and must contain the location of the new contents of the note. The contents of the catalog entry that is passed will completely replace the existing contents of the note in the metadata.

```
/*
 *  Pull the copy of the Note into the
 *  preview buffer and allow the
```

```
    *  user to edit it.
    */
rc=preview('clear');

rc=preview('copy',libref||'.'||entry);

rc=preview('edit');

if rc = 0 then do;

   /*
    *  If the user modified the copy, save the
    *  modifications back to the catalog entry and
    *  update the metadata with the new contents.
    */

rc=preview('save',libref||'.'||entry);

call send(i_api,'_UPDATE_METADATA_',rc,l_note);

end; /* if */

rc=preview('clear');
```

Creating Notes

To add a note to an object's metadata, pass the metadata ID of the object to the _ADD_METADATA_ or _UPDATE_METADATA_ method. In the properties list for these methods, the NOTE property must contain the properties list that is expected by the WHNOTE type. If a note already exists for the object that is passed the _UPDATE_METADATA_ method, the contents of the existing note will be replaced with the contents of the new note. Note that _ADD_METADATA_ is not valid for the WHNOTE type.

WHOBJECT

Base metadata type for SAS/Warehouse Administrator objects

Category: Object Types

Parent

"WHROOT" on page 226

Overview

WHOBJECT is the base metadata type for SAS/Warehouse Administrator objects. You can view most of these objects in both the SAS/Warehouse Administrator Explorer and the Process Editor. You can view Types WHEFILE, WHODTTBL, and the children of WHTBLPRC only in the Process Editor.

Properties

The following table lists all of the properties for WHOBJECT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Group	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Owner	L	No	No	No

New properties for WHOBJECT are as follows:

ADMINISTRATOR
specifies an SCL list of general identifying information about the person who is the administrator of the object. The list must be of type WHPERSON or a subtype of WHPERSON.

GROUP
specifies an SCL list of general identifying information about the groups to which this object belongs.

Note: If you pass the GROUP property to the _ADD_METADATA_ method, the object you add will only be added to the first group in the list. △

ICON
specifies the four-level catalog entry name (such as *libref.catalog.entry.IMAGE*) of the icon that is associated with this object. For more information about icons, see "Using Icon Information" on page 69.

MEMBERS
specifies an SCL list of general identifying information about the members of this object. This member list is closely related to the hierarchy that is depicted in the metadata views in SAS/Warehouse Administrator Explorer.

OWNER
specifies an SCL list of general identifying information about the person who owns the object. The list must be of type WHPERSON or a subtype of WHPERSON.

Using WHOBJECT

Add	Update	Delete
No	No	No

WHOBJECT is not used to read or write metadata in a repository. It is a template for all SAS/Warehouse Administrator Explorer objects. WHOBJECT is an independent type.

WHODDTBL

Metadata type for ODDs

Category: Object Types—Explorer

Parent

"WHTABLE" on page 254

Overview

WHODDTBL models the metadata for operational data definitions (ODD) in SAS/Warehouse Administrator. An *ODD* is a SAS data set, SAS view, SAS/ACCESS view descriptor, or SQL view descriptor that identifies an operational data source. In the SAS/Warehouse Administrator Explorer, to add an ODD to an environment:

1. Select the environment with the right mouse button.
2. Select

   Add Item ▶ ODD

3. Select the ODD with the right mouse button.
4. Enter the ODD information.

Properties

The following table lists all of the properties for WHODDTBL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	Yes	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No

Output Targets	L	* Req	No	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHODDTBL

Add	Update	Delete
Yes	Yes	Yes

WHODDTBL is an independent type, like its parent, WHTABLE. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

WHODTTBL

Metadata type for ODTs (Data Files)

Category: Object Types—Process Editor

Parent

"WHTABLE" on page 254

Overview

WHODTTBL models the metadata for operational data tables (ODT) in SAS/Warehouse Administrator. An *ODT* is a SAS table that is an input to an operational data definition (ODD) in the Process Editor. In the Process Editor, the ODT is called a data file. Here is one way to add an ODT in SAS/Warehouse Administrator:

1. In the Explorer, select an ODD with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the ODD with the right mouse button.
4. Select

   Add ▶ Data File

5 Select the data file with the right mouse button.

6 Select **Properties**.

7 Enter the ODT information.

Properties

The following table lists all of the properties for WHODTTBL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	No	No	No
Administrator	L	Yes	Yes	No
Columns	L	No	No	Yes
Creating Job	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	No	No	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No

Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Using WHODTTBL

Add	Update	Delete
Yes	Yes	Yes

WHODTTBL is an independent type. To understand how all subtypes of WHTABLE relate to other types, see the models in “Relationships Among Metadata Types” on page 53.

WHOLAP

Base metadata type for OLAP dimension, hierarchy, and crossing

Category: Object Types—OLAP

Parent

“WHROOT” on page 226

Overview

WHOLAP is the base metadata type for OLAP dimensions, hierarchies, and crossings.

Properties

The following table lists all of the properties for WHOLAP and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
OLAP Groups	L	No	No	No
OLAP Members	L	No	No	Yes

New properties for WHOLAP are as follows:

OLAP GROUPS
specifies an SCL list of general identifying information about the OLAP groups to which this object belongs.

OLAP MEMBERS
specifies an SCL list of general identifying information about the members of an OLAP object.

Using WHOLAP

Add	Update	Delete
No	No	No

WHOLAP and its children are independent types.

WHOLPCRS

Metadata type for OLAP crossing

Category: Object Types—OLAP

Parent

"WHOLAP" on page 188

Overview

WHOLPCRS models the metadata for OLAP crossings in OLAP tables, groups, and MDDBs. A *crossing* is a unique list of zero or more class columns that defines a summarization level (subtable) to be stored in one or more OLAP summary data stores. That is, a crossing represents a grouping on which summary statistics are calculated. You must have at least one crossing for an OLAP table or an OLAP MDDB, and both summary data stores can have multiple crossings. All class columns must be in at least one crossing.

Properties

The following table lists all of the properties for WHOLPCRS and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Columns	L	Yes	No	Yes
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Id	C	* Req	No	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Req	Yes	No
Note	L	Yes	No	Yes
NValue	N	Yes	Yes	No

OLAP Groups	L	No	No	No
OLAP Members	L	No	No	Yes
OLAP Structure	L	* Auto supplied	No	No

New properties for WHOLPCRS are as follows:

COLUMNS
specifies an SCL list of general identifying information about the columns that are associated with an OLAP group, table, or MDDB.

OLAP STRUCTURE
specifies an SCL list of general identifying information about an OLAP group, table, or MDDB.

Using WHOLPCRS

Add	Update	Delete
No	Yes	Yes

WHOLPCRS is a dependent type, like its parent, WHOLAP.

WHOLPCUB

Metadata type for OLAP cube

Category: Object Types—OLAP

Parent

"WHOLAP" on page 188

Overview

WHOLPCUB models the metadata for OLAP Cubes. A *cube* is a multidimensional data source that might be "virtual" and represents the OLAP data from which you can generate a report.

Properties

The following table lists all of the properties for WHOLPCUB and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Id	C	* Req	No	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Req	Yes	No
Note	L	Yes	No	Yes
NValue	N	Yes	Yes	No
OLAP Groups	L	No	No	No
OLAP Members	L	Yes	Yes	Yes
OLAP Structure	L	* Auto supplied	No	No

WHOLPCUB has the following new property:

OLAP STRUCTURE
specifies an SCL list of general identifying information about an OLAP group, table, or MDDB.

Using WHOLPCUB

Add	Update	Delete
No	Yes	Yes

WHOLPCUB is a dependent type, like its parent, WHOLAP.

WHOLPDIM

Metadata type for OLAP dimension

Category: Object Types—OLAP

Parent

"WHOLAP" on page 188

Overview

WHOLPDIM models the metadata for OLAP dimensions in OLAP tables, groups, and MDDBs. A *dimension* organizes related columns, which are in hierarchies. For example, you could organize sales data into three dimensions: Geography, Time, and Product. The Time dimension could include these hierarchies, which provide different paths in order to drill down to increasing levels of detail: Time-by-Week and Time-by-Month.

Properties

The following table lists all of the properties for WHOLPDIM and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Id	C	* Req	No	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No

Name	C	* Req	Yes	No
Note	L	Yes	No	Yes
NValue	N	Yes	Yes	No
OLAP Groups	L	* Auto supplied	No	No
OLAP Members	L	Yes	Yes	Yes

Using WHOLPDIM

Add	Update	Delete
No	Yes	Yes

WHOLPDIM is a dependent type, like its parent, WHOLAP.

WHOLPHIR

Metadata type for OLAP hierarchy

Category: Object Types—OLAP

Parent

"WHOLAP" on page 188

Overview

WHOLPHIR models the metadata for OLAP hierarchies in OLAP tables, groups, and MDDBs. A *hierarchy* is a unique, ordered list of class columns that specifies related data and is a member of a dimension. Each hierarchy provides a navigational path in order to drill down to increasing levels of detail.

Properties

The following table lists all of the properties for WHOLPHIR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one. ** Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Update	Update Method	Read Method Expand Parm.
Columns	L	Yes	No	Yes
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Id	C	* Req	* Req	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	* Req	Yes	No
Note	L	Yes	No	Yes
NValue	N	Yes	Yes	No
OLAP Groups	L	* Auto supplied	No	No
OLAP Members	L	No	No	Yes

WHOLPHIR has the following new property:

COLUMNS
specifies an SCL list of general identifying information about the columns that are associated with an OLAP group, table, or MDDB.

Using WHOLPHIR

Add	Update	Delete
No	Yes	Yes

WHOLPHIR is a dependent type, like its parent, WHOLAP.

WHOLPMDD

Metadata type for OLAP MDDBs

Category: Object Types—Explorer

Parent

"WHOLPSTC" on page 198

Overview

WHOLPMDD replaces the WHSUMDDB metadata type. WHOLPMDD models the metadata for a SAS MDDB (multidimensional database) in SAS/Warehouse Administrator.

To add an OLAP MDDB with the SAS/Warehouse Administrator Explorer:

1. Select an OLAP group with the right mouse button.
2. Select

   Add Item ▶ OLAP MDDB

3. Select the table with the right mouse button.
4. Select **`Properties`**.
5. Enter the OLAP MDDB information.

Properties

The following table lists all of the properties for WHOLPMDD and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes

Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Crossings	L	Yes	Yes	Yes
Cube	L	Yes	Yes	Yes
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
OLAP Type	C	No	No	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHOLPMDD

Add	Update	Delete
Yes	Yes	Yes

WHOLPMDD is an independent type, like its parent, WHOLPSTC.

WHOLPSTC

Base metadata type for OLAP tables, groups, and MDDBs

Category: Object Types—Explorer

Parent

"WHTABLE" on page 254

Overview

WHOLPSTC is the base metadata type for OLAP tables, groups, and MDDBs.

Properties

The following table lists all of the properties for WHOLPSTC and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	No	No	No
Administrator	L	No	No	No
Columns	L	No	No	Yes
Creating Job	L	No	No	No
Crossings	L	No	No	Yes

Cube	L	No	No	Yes
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Group	L	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
OLAP Type	C	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	No	No	No
Physical Storage	L	No	No	Yes
Process	L	No	No	Yes
Table Name	C	No	No	No
Using Jobs	L	No	No	No

New properties for WHOLPSTC are as follows:

CROSSINGS
specifies an SCL list of general identifying information about the crossings that are associated with an OLAP group, table, or MDDB.

CUBE
specifies an SCL list of general identifying information about the cube that is associated with an OLAP table, group, or MDDB.

OLAP Type
specifies the character string that contains the type of OLAP configuration that is being created. The valid value for OLAP table is **DATA**. The valid value for OLAP MDDB is **MDDB**. The valid values for OLAP group are **HOLAP**, **ROLAP**, **MOLAP**, and **MIXED**.

Using WHOLPSTC

Add	Update	Delete
No	No	No

WHOLPSTC and its children are independent types.

WHOLPTBL

Metadata type for OLAP tables

Category: Object Types—Explorer

Parent

"WHOLPSTC" on page 198

Overview

WHOLPTBL replaces the WHSUMTBL metadata type. WHOLPTBL models the metadata for OLAP tables in SAS/Warehouse Administrator. An OLAP table can be a SAS table or view, or DBMS table or view.

To add an OLAP table with the SAS/Warehouse Administrator Explorer:

1. Select an OLAP group with the right mouse button.
2. Select

   Add Item ▶ OLAP table

3. Select the table with the right mouse button.
4. Select **Properties**.
5. Enter the OLAP table information.

Properties

The following table lists all of the properties for WHOLPTBL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Note: A CREATING JOB property is required if the INPUT SOURCES property is also specified.
△

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	Yes	Yes	No
Administrator	L	Yes	Yes	No
Columns	L	Yes	Yes	Yes
Creating Job	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Crossings	L	Yes	Yes	Yes
Cube	L	Yes	Yes	Yes
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	Yes	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes (see Property Dependencies)	Yes (see Property Dependencies)	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No

OLAP Type	C	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	Yes	Yes	No
Owner	L	Yes	Yes	No
Physical Storage	L	Yes	Yes	Yes
Process	L	Yes	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Property Dependencies You must define a CREATING JOB in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHOLPTBL

Add	Update	Delete
Yes	Yes	Yes

WHOLPTBL is an independent type, like its parent, WHOLPSTC.

WHPERSON

Metadata type for person records

Category: Global Metadata Types

Parent

"WHROOT" on page 226

Overview

WHPERSON models the metadata for person records in SAS/Warehouse Administrator. These records are used to identify owners, administrators, and other people who are responsible for warehouse elements. In SAS/Warehouse Administrator, to add a person record to the current environment in the Explorer:

1 Select

File ► Setup

from the pull-down menu.

2 Select **Contacts**.

3 Click Add.

4 Enter the person's information.

Properties

The following table lists all of the properties for WHPERSON and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Address	L	Yes	Yes	No
Administered Objects	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Email Address	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owned Objects	L	No	No	No
Phone	L	Yes	Yes	No
Title	L	Yes	Yes	No

New properties for WHPERSON are as follows:

ADDRESS
: specifies an SCL list of character items that contain the different lines of the person's address. Each item has a maximum length of 200 characters. Currently, a maximum of four lines is supported.

ADMINISTERED OBJECTS
specifies an SCL list of the general identifying information about the objects for which this person is designated as the administrator.

EMAIL ADDRESS
indicates the person's e-mail address. The maximum length of this field is 200 characters.

ICON
represents the catalog entry name of the associated icon. For more information about icons, see "Using Icon Information" on page 69.

NAME
indicates the maximum 200-character string for the person's name.

OWNED OBJECTS
specifies an SCL list of the general identifying information of the objects for which this person is designated as the owner.

PHONE
specifies an SCL list of character items that contain the different lines of the person's phone number(s). Each item has a maximum length of 200 characters. Currently, a maximum of two lines is supported.

TITLE
specifies an SCL list of character items that contain the person's title. Each item has a maximum length of 200 characters. Currently, a maximum of two lines is supported.

Using WHPERSON

Add	Update	Delete
Yes	Yes	Yes

WHPERSON is an independent type. To understand how it relates to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

WHPHYSTR

Base metadata type for physical storage objects

Category: Physical Storage Types

Parent

"WHROOT" on page 226

Overview

WHPHYSTR is the base metadata type for physical storage objects in SAS/Warehouse Administrator.

Properties

New properties for WHPHYSTR are as follows:

INDEXES
specifies an SCL list of general identifying information about the indexes that are defined for this store.

LOAD TECHNIQUE
specifies the character string that indicates how this table is loaded. The current returned values can be **REFRESH**, **APPEND**, or **MERGE**. Note that you can extend this list over time.

TABLE
specifies an SCL list of general identifying information about the table for which this physical storage definition is used.

TABLE NAME
indicates the name of the table in the data store.

Using WHPHYSTR

Add	Update	Delete
No	No	No

WHPHYSTR is a dependent type. WHPHYSTR is not used to read or write metadata from a repository. See the usage information for its subtypes: WHSASSTR or WHDBMSST. See also the physical storage models in "Relationships Among Metadata Types" on page 53.

WHPOBJCT

Base metadata type for the Process Editor

Category: Object Types—Process Editor

Parent

"WHROOT" on page 226

Overview

WHPOBJCT is the base metadata type for job process objects.

Properties

The following table lists all of the properties for WHPOBJCT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Icon	C	No	No	No
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Owner	L	No	No	No
Process Groups	L	No	No	No
Process Members	L	No	No	No

New properties for WHPOBJCT are as follows:

ADMINISTRATOR
: specifies an SCL list of general identifying information about the person who is the administrator of the object. The list must be of type WHPERSON or a subtype of WHPERSON. When you add a metadata type that includes this property, if a value is not provided, an appropriate value will be copied from the process group if a value is available.

ICON
: indicates the four-level catalog entry name (such as *libref.catalog.entry.IMAGE*) of the icon that is associated with this object. For more information about icons, see “Using Icon Information” on page 69.

OWNER
: specifies an SCL list of general identifying information about the person who owns the object. The list must be of type WHPERSON or a subtype of WHPERSON. When you add a metadata type that includes this property, if a value is not provided, an appropriate value will be copied from the process group if a value is available.

PROCESS GROUPS
specifies an SCL list of general identifying information about the process groups to which this object belongs. This list must be of type WHDW, WHDWENV, WHGRPJOB, or a subtype of those. A process object cannot be a member of more than one PROCESS GROUP. At least one group is required.

PROCESS MEMBERS
specifies an SCL list of general identifying information about the process members that belong to this object.

Using WHPOBJCT

Add	Update	Delete
No	No	No

WHPOBJCT is an independent type. WHPOBJCT is not used to read or write metadata in a repository.

WHPRCLDR

Base metadata type for table load processes

Category: Process Types—Load

Parent

"WHPRCMAN" on page 209

Overview

WHPRCLDR is the base metadata type for table load processes in the SAS/Warehouse Administrator Process Editor.

Properties

The following table lists all of the properties for WHPRCLDR and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Host	L	No	No	No
Id	C	No	No	No
Load Options	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Tables	L	No	No	No
Owner	L	No	No	No
Responsibility	C	No	No	No
Source Code	L	No	No	Yes
Source File	L	No	No	Yes
Step Source Code	L	No	No	Yes
Subprocesses	L	No	No	Yes

WHPRCLDR has the following new property:

LOAD OPTIONS
specifies an SCL list of options for the specified load process. The options are as follows:

- GENERATION LEVEL (**`1.1`** or **`2.0`**) selects the release level of the code that SAS/Warehouse Administrator will generate for the specified load process. Each level has specific options that it can support, as well as other characteristics.
- LOAD TIME COLUMN (**`YES`** or **`NO`**) indicates whether a Load Time column will be added to the table that is being loaded by the specified process.

  Note: See Property Dependencies. △

- DROP INDEXES (**`YES`** or **`NO`**) specifies that you should drop (remove) any existing indexes on the table to be loaded before you load the data into the table. Based on the metadata definitions, the appropriate indexes will be recreated after loading the data. This option is useful when updating the indexes during loading is too slow.
- TRUNCATE TABLE (**`YES`** or **`NO`**) specifies that when you refresh the data in a table, the table should be truncated (all data rows are removed but the table is not) instead of completely dropping the table and recreating it from scratch. This option is useful when the table has many options, privileges, and other characteristics that are defined in the database.

- UNION MULTIPLE INPUTS (**YES** or **NO**) specifies that any multiple inputs to the current load process will be unioned together before you load the table. A *union* is identical to a SET statement in a SAS data step that contains multiple input table designations.

In the SAS/Warehouse Administrator interface, LOAD OPTIONS are specified on the Load Options tab of the Load process attributes window for a given data store. Here are some example return values for a data store whose Load process attributes window includes a Load Options tab:

```
LOAD OPTIONS=(     GENERATION LEVEL='2.0'
                   LOAD TIME COLUMN='NO'
                   UNION MULTIPLE INPUTS='YES'
                   DROP INDEXES='NO'
                   TRUNCATE TABLE='NO'
              )
```

Property Dependencies Subtypes of WHPRCLDR enable you to add or update the LOAD OPTIONS property. When you add or update the LOAD OPTIONS property, if the value of the LOAD TIME COLUMN item is **YES**, then a valid load time column must exist for the table that is associated with this load process to avoid errors when processing the SOURCE CODE and STEP SOURCE CODE properties. You can add a load time column to a table as described in the documentation for the WHCOLTIM type.

Using WHPRCLDR

Add	Update	Delete
No	No	No

WHPRCLDR is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHPRCMAN

Base metadata type for main processes

Category: Process Types

Parent

"WHPROCES" on page 223

Overview

WHPRCMAN is one of the base types for main processes in the SAS/Warehouse Administrator Process Editor. It is the parent of WHPRCLDR, the base type for all load process metadata.

Properties

The following table lists all of the properties for WHPRCMAN and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Host	L	No	No	No
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Tables	L	No	No	No
Owner	L	No	No	No
Responsibility	C	No	No	No
Source Code	L	No	No	Yes
Source File	L	No	No	Yes
Step Source Code	L	No	No	Yes
Subprocesses	L	No	No	Yes

New properties for WHPRCMAN are as follows:

OUTPUT TABLES
specifies an SCL list of general identifying information about the output tables of this process. All WHPRCMAN processes will have at least one output table that is associated with them (a WHTABLE subtype).

SOURCE CODE
specifies an SCL list of general identifying information about the source code for this process. This source code is the same as is seen when you select

View Code ► All

in the SAS/Warehouse Administrator Process Editor.

The source code information that is returned here will be that of a temporary, working location of a copy of the source code and might be different for each request for this information.

SOURCE FILE
specifies an SCL list of general identifying information about any user-registered code for a process. This list must be of type WHSRCCAT or a subtype of WHSRCCAT. WHJOBCAT or any subtype of WHJOBCAT will be rejected, however. For process steps that consist of user-written code, this property returns the registered source code location. For process steps that consist of code that is generated by SAS/Warehouse Administrator, this property will return an empty list.

SUBPROCESSES
specifies an SCL list of general identifying information about any subprocesses that might be registered for this process. This list must be of type WHPRCSPR or a subtype of WHPRCSPR.

Using WHPRCMAN

Add	Update	Delete
No	No	No

WHPRCMAN is a dependent type. WHPRCMAN has the same usage information as "WHPROCES" on page 223.

WHPRCMAP

Metadata type for data mapping processes

Category: Process Types

Parent

"WHPRCMAN" on page 209

Overview

WHPRCMAP models the metadata for data mapping processes in the SAS/Warehouse Administrator Process Editor. A *data mapping* is a metadata record used to generate or retrieve a routine that maps columns from one or more data sources into one or more data tables, detail tables, OLAP tables, or OLAP MDDBs. Common mappings include one-to-one (one data source to a target table), joins (one or more data sources merged by one or more common columns), and unions (two or more data sources appended to a target table). Here is one way to add a data mapping process in SAS/Warehouse Administrator:

1. In the Explorer, select a detail table or a data table with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select

   Add ▶ Inputs

5. Enter the mapping information.

Properties

The following table lists all of the properties for WHPRCMAP and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No

Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	Yes	Yes	No
Responsibility	C	Yes	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	No	No	Yes
Subprocesses	L	Yes	Yes	Yes
Transformations	L	Yes	Yes	Yes

WHPRCMAP has the following new property:

TRANSFORMATIONS
specifies an SCL list of general identifying information about the transformations that are defined for this mapping.

Using WHPRCMAP

Add	Update	Delete
No	Yes	No

WHPRCMAP is a dependent type. WHPRCMAP has the same usage information as "WHPROCES" on page 223. See the process model diagram in "Relationships Among Metadata Types" on page 53.

WHPRCPST

Metadata type for post-load processes

Category: Process Types

Parent

"WHPRCSPR" on page 217

Overview

WHPRCPST models the metadata for post-load processes in SAS/Warehouse Administrator. A *post-load process* is code that is specified in the Process Editor to

execute after a table is loaded. Here is one way to add a post-load process in SAS/Warehouse Administrator:

1. In the Explorer, select a table with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select **Edit Load Step**.
5. Go to the Post Processing tab.
6. Enter the process information.

Properties

The following table lists all of the properties for WHPRCPST and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Is Active	N	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owner	L	No	No	No

Process	L	* Auto supplied	No	No
Responsibility	C	No	No	No
Step Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes

New Types for WHPRCPST are as follows:

PROCESS
specifies an SCL list of general identifying information about the load process to which this post-load process belongs. This list must be a subtype of WHPRCLDR.

SOURCE FILE
specifies an SCL list of general identifying information about any user-registered code for a process. This list must be of type WHSRCCAT or a subtype of WHSRCCAT. WHJOBCAT or any subtype of WHJOBCAT will be rejected, however. For process steps that consist of user-written code, this property returns the registered source code location. For process steps that consist of code that is generated by SAS/Warehouse Administrator, this property will return an empty list.

Using WHPRCPST

Add	Update	Delete
No	Yes	Yes

WHPRCPST is a dependent type. WHPRCPST has the same usage information as "WHPROCES" on page 223.

WHPRCREC

Metadata type for record selector processes

Category: Process Types

Parent

"WHPRCMAN" on page 209

Overview

WHPRCREC models the metadata for record selector processes in SAS/Warehouse Administrator. A *record selector process* is a metadata record that is used to generate or retrieve a routine that subsets data prior to loading it to a specified table. For example, you can use a record selector process to subset the operational data specified in an ODD.

Here is one way to add a record selector process in SAS/Warehouse Administrator:

1 In the Explorer, select a table with the right mouse button.
2 Select **Process** from the pop-up menu.
3 In the Process Editor, select an ODD for that table with the right mouse button.
4 Select

   Insert ▶ Record Selector

5 Enter the record selector information.

Properties

The following table lists all of the properties for WHPRCREC and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No

Owner	L	Yes	Yes	No
Responsibility	C	No	No	Yes
Source Code	L	No	No	Yes
Source File	L	No	No	Yes
Step Source Code	L	No	No	Yes
Subprocesses	L	Yes	Yes	Yes

Using WHPRCREC

Add	Update	Delete
No	Yes	No

WHPRCREC is a dependent type. WHPRCREC has the same usage information as "WHPROCES" on page 223.

WHPRCSPR

Base metadata type for subprocesses

Category: Process Types

Parent

"WHPROCES" on page 223

Overview

WHPRCSPR is the base metadata type for subprocesses in SAS/Warehouse Administrator, such as subset processes and post-processing processes.

Properties

The following table lists all of the properties for WHPRCSPR and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Host	L	No	No	No
Id	C	No	No	No
Is Active	N	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Owner	L	No	No	No
Process	L	No	No	No
Responsibility	C	No	No	No
Step Source Code	L	No	No	Yes

New properties for WHPRCSPR are as follows:

IS ACTIVE
specifies the numeric value that indicates whether this subprocess is active (**1**) or inactive (**0**).

PROCESS
specifies the main process object that is associated with this subprocess. This object must be a subtype of WHPRCMAN.

Using WHPRCSPR

Add	Update	Delete
No	No	No

WHPRCSPR is a dependent type. WHPRCSPR has the same usage information as "WHPROCES" on page 223.

WHPRCUSR

Metadata type for user exit processes

Category: Process Types

Parent

"WHPRCMAN" on page 209

Overview

WHPRCUSR models the metadata for user exit processes in SAS/Warehouse Administrator. A *user exit process* is a metadata record that is used to retrieve a user-written routine. You must store the routine in a SAS catalog with an entry type of SOURCE or SCL. Here is one way to add a user exit process in SAS/Warehouse Administrator:

1. In the Explorer, select a table with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select

   Add ▶ User Exit

5. Enter the user exit information.

Properties

The following table lists all of the properties for WHPRCUSR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	Yes	Yes	No
Responsibility	C	No	No	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	No	No	Yes
Subprocesses	L	No	No	Yes

Using WHPRCUSR

Add	Update	Delete
No	Yes	No

WHPRCUSR is a dependent type. WHPRCUSR has the same usage information as “WHPROCES” on page 223.

WHPRCXFR

Metadata type for data transfer processes

Category: Process Types

Parent

"WHPRCMAN" on page 209

Overview

WHPRCXFR models the metadata for data transfer processes in SAS/Warehouse Administrator. A *data transfer process* is a metadata record that is used to generate or retrieve a routine that moves data from one host to another. Data transfers are required when an input source and the target data reside on different hosts. Here is one way to add a data transfer process in SAS/Warehouse Administrator:

1. In the Explorer, select a table with the right mouse button.
2. Select **`Process`** from the pop-up menu.
3. In the Process Editor, select a table or ODD with the right mouse button.
4. Select

   Insert ▶ Data Transfer

5. Enter the transfer information.

Properties

The following table lists all of the properties for WHPRCXFR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Output Tables	L	* Auto supplied	No	No
Owner	L	Yes	Yes	No
Responsibility	C	Yes	Yes	No
Source Code	L	No	No	Yes
Source File	L	Yes	Yes	Yes
Step Source Code	L	No	No	Yes
Subprocesses	L	No	No	Yes

Using WHPRCXFR

Add	Update	Delete
No	Yes	No

WHPRCXFR is a dependent type. WHPRCXFR has the same usage information as “WHPROCES” on page 223.

WHPROCES

Base metadata type for processes

Category: Process Types

Parent

"WHROOT" on page 226

Overview

WHPROCES is the base type for all process metadata in SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHPROCES and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Host	L	No	No	No
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Owner	L	No	No	No

Responsibility	C	No	No	No
Step Source Code	L	No	No	Yes

New properties for WHPROCES are as follows:

ADMINISTRATOR
specifies an SCL list of general identifying information about the person who is the administrator of the process object. The list must be of type WHPERSON or a subtype of WHPERSON.

HOST
specifies an SCL list of general identifying information about the host on which this process is to execute. The list must be of type WHHOST or a subtype of WHHOST.

OWNER
specifies an SCL list of general identifying information about the person who owns the process object. The list must be of type WHPERSON or a subtype of WHPERSON.

RESPONSIBILITY
specifies the character string that indicates who is currently responsible for the creation of the code that is associated with this process. Possible values are **SAS** or **USER**.

SAS indicates that SAS/Warehouse Administrator is creating this code dynamically based on the current metadata. **USER** indicates that the user has written the code for this process and is responsible for it.

STEP SOURCE CODE
specifies an SCL list of general identifying information about the source code of the individual step in the process. This source code is the same as is seen when you select

View Code ▶ Step

in the SAS/Warehouse Administrator Process Editor.

The source code information that is returned here will be that of a temporary, working location of a copy of the source code and therefore might be different for each request for this information.

Using WHPROCES

Add	Update	Delete
No	No	No

WHPROCES is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in “Relationships Among Metadata Types” on page 53.

For details about reading process information, see “Reading Process Flow Metadata” on page 62.

WHREPLIB

Metadata type for metadata repositories

Category: SAS Library Types

Parent

"WHLIBRY" on page 175

Overview

WHREPLIB models metadata repositories in SAS/Warehouse Administrator. SAS/Warehouse Administrator has a *partitioned* metadata repository scheme. Each warehouse environment has a repository that is named _MASTER. Each data warehouse within an environment has a repository that is named _DWMD. In SAS/Warehouse Administrator, these repositories are created when the environment or warehouse is created.

Properties

The following table lists all of the properties for WHREPLIB and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	Yes	No
Database	L	Yes	No	No
DBMS Libname	N	* Default	No	No
Desc	C	No	Yes	No
Engine	C	No	Yes	No
Extended Attributes	L	No	Yes	Yes
Icon	C	No	Yes	No
Id	C	No	* Req	No
Libref	C	No	Yes	No
Metadata Created	C	No	No	No

Metadata Updated	C	No	No	No
Name	C	No	* Default	No
Note	L	No	Yes	Yes
NValue	N	No	Yes	No
Options	L	No	Yes	No
Path	L	No	Yes	No
Preassigned	N	No	Yes	No
Tables	L	No	No	No

Using WHREPLIB

Add	Update	Delete
No	Yes	No

WHREPLIB is an independent type, like its parent, WHLIBRY. To understand how all subtypes of WHLIBRY relate to other types, see the physical storage models in "Relationships Among Metadata Types" on page 53.

For a general discussion of metadata repositories, see "Metadata Repositories" on page 10.

WHROOT

Root type for all SAS/Warehouse Administrator metadata types

Category: Root Metadata Type—SAS/Warehouse Administrator

Parent

SASHELP.FSP.OBJECT

Overview

WHROOT is the root for all metadata types in SAS/Warehouse Administrator.

Properties

New properties for WHROOT are as follows:

CVALUE
: indicates the 40-character string that a site can use to extend the metadata that is maintained by SAS/Warehouse Administrator. Use it for site-specific character metadata.

DESC
indicates the optional text that describes the purpose of an object or other information that is useful to a site.

EXTENDED ATTRIBUTES
specifies an SCL list that a site can use to extend the metadata that is maintained by SAS/Warehouse Administrator. Items in the list are

OBJECT specifies the general identifying information about the owning object for this extended attribute.

TYPE indicates the data type of the attribute, **C** for character data is the only valid type for this release.

VALUE indicates the 200-character string that contains the extended attribute text, such as a URL or a file path to a document that describes the owning object.

For usage details, see "Using WHEXTATR" on page 116. The EXTENDED ATTRIBUTES property is implemented with the object "WHEXTATR" on page 114.

ID
indicates the metadata identifier for a specific metadata object in a repository. The identifier is 26 characters in length and is in the format: *REPOSID.TYPEID.INSTANCEID*.

REPOSID specifies the ID of the repository in which the metadata resides. It is eight characters in length.

TYPEID specifies the type of metadata object, such as WHDETAIL. It is eight characters in length.

INSTANCEID distinguishes one metadata object from all others of that type in a given repository. It is eight characters in length.

METADATA CREATED
specifies the SAS datetime value for when the metadata for this object was initially created. (A character value that is formatted with a SAS DATETIME. FORMAT.)

METADATA UPDATED
specifies the SAS datetime value for when the metadata for this object was updated. (A character value that is formatted with a SAS DATETIME. FORMAT.)

NAME
indicates the name of the metadata object. The name that is returned is in 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 so on.

The length of the name depends on the individual type. All names can be at most 40 characters in length. Some types, such as WHPERSON, allow the name to be longer than 40 characters. The maximum length of the name is 40 characters unless otherwise noted in a particular type.

NOTE
indicates the user-entered descriptions of objects, columns, or processes. NOTE metadata is modeled by the WHNOTE type. For details, see "WHNOTE" on page 179.

NVALUE
indicates the numeric value that a site can use to extend the metadata that is maintained by SAS/Warehouse Administrator. Use it for site-specific numeric metadata.

Note: The documentation for many metadata types refers to general identifying information. This phrase refers to the ID, NAME, and DESC properties. For more details, see "Identifying Metadata" on page 7. △

Using WHROOT

Add	Update	Delete
No	No	No

WHROOT is an independent type. WHROOT is not used to read or write metadata in a repository. It is a template for all metadata types in SAS/Warehouse Administrator.

WHROWSEL

Metadata type for a row selector

Category: Process Types

Parent

"WHPRCSPR" on page 217

Overview

The WHROWSEL type models the metadata for all row selectors. Here is one way to specify row selector metadata through the SAS/Warehouse Administrator interface:

1. Display a process flow with a mapping in the Process Editor.
2. In the Process Editor, click the right mouse button on a mapping and select **`Properties`**.
3. Go to the Output Data tab.
4. Click the Generation Options button.
5. Click the Row Selection tab.
6. For the rows that are selected, select **`Row Selection Conditions`** and then click the Define button.
7. In the Expression Builder, select **`Component of Input Tables`** and then select an input table and column.
8. Click OK on each window until you return to the Process Editor.

Properties

The following table lists all of the properties for WHROWSEL and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one. ** Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	No	No	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Is Active	N	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Tables	L	* Req	No	No
Outputs Targets	L	No	No	No
Owner	L	No	No	No
Process	L	* Auto supplied	No	No
Responsibility	C	No	No	No
Selection Type	C	Yes (See Property Dependencies)	Yes	No
Source Text	L	Yes (See Property Dependencies)	Yes	No
Step Source Code	L	No	No	Yes

New properties for WHROWSEL are as follows:

INPUT OBJECTS
specifies an SCL list of general identifying information about the columns that are input to this subprocess.

INPUT SOURCES
specifies an SCL list of general identifying information about the nearest intermediate output table or loadable table that is a source to the current table or column. This list must be of type WHCTRNFM or a subtype of WHCOLUMN, and it must have the appropriate relation to the main process. For WHROWSEL, this property can return the same list as INPUT OBJECTS. However, if the user builds the row selector using columns from the output table (in Expression Builder), INPUT SOURCES will return a transformation (WHCTRNFM) and INPUT OBJECTS will return the columns that are used in that transformation.

OUTPUT OBJECTS
specifies a property that is currently unused.

OUTPUT TABLES
specifies an SCL list of general identifying information about the output tables for this subprocess. This list must be of type WHTBLMAP or a subtype of WHTBLMAP. All WHROWSEL subprocesses will have at least one output table that is associated with them.

OUTPUT TARGETS
specifies a property that is currently unused.

PROCESS
specifies an SCL list of general identifying information about the mapping process that called this row selection process. This list must be of type WHPRCMAP or a subtype of WHPRCMAP.

SOURCE TEXT
specifies an SCL list of character items that specify a WHERE clause or other subsetting code. Each item can contain a maximum of 200 characters of source code.

Note: See Property Dependencies. △

SELECTION TYPE
specifies a character string that indicates the row selection type. Valid row selection types are **ALL ROWS**, **ROW SELECTION CONDITIONS**, or **USER DEFINED STATEMENTS**.

Note: See Property Dependencies. △

Property Dependencies When you use the indirect add approach,

- SELECTION TYPE defaults to **ALL ROWS** if not otherwise specified by the user.
- SOURCE TEXT is optional for indirect adds if the SELECTION TYPE is **ALL ROWS**; otherwise, SOURCE TEXT is required.

Using WHROWSEL

Add	Update	Delete
No	Yes	Yes

WHROWSEL is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHSASSTR

Metadata type for SAS physical data stores

Category: Physical Storage Types

Parent

"WHPHYSTR" on page 204

Overview

WHSASSTR models the metadata for SAS physical data stores in SAS/Warehouse Administrator. These stores are specified for tables that are stored in SAS format, using the Physical Storage tab in the table property window.

Properties

The following table lists all of the properties for WHSASSTR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes

Host	L	Yes	Yes	No
Id	C	* Req	* Req	No
Indexes	L	Yes	Yes	Yes
Library	L	Yes	Yes	No
Load Technique	C	Yes	Yes	No
Metadata Created	C	* Auto supplied	No	No
Metadata Updated	C	* Auto supplied	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Table	L	* Auto supplied	No	No
Table Name	C	Yes	Yes	No
Table Options	L	* Default	* Default	No

New properties for WHSASSTR are as follows:

HOST
specifies an SCL list of general identifying information about the host on which this data is accessed.

LIBRARY
specifies an SCL list of general identifying information about the SAS Library that contains this data store.

TABLE OPTIONS
specifies an SCL list of options that are used in creating or loading this table. The CREATE sublist contains the SQL options that are used to create the table. The LOAD sublist contains the DBLOAD statements that are used to load the table.

See the Usage notes for details about the TABLE OPTIONS property and data set passwords.

Using WHSASSTR

Add	Update	Delete
No	Yes	No

WHSASSTR is a dependent type. To understand how it relates to other types, see the physical storage models in “Relationships Among Metadata Types” on page 53.

TABLE OPTIONS Property and SAS Data Set Passwords

The actual data set passwords will only be returned using the _GET_METADATA_ method if the application is running as a *secure application*. The only method that is currently supported to run as a secure application is to run as an add-in generator. See

SAS / Warehouse Administrator User's Guide for documentation on running as an add-in generator. You can determine the presence of passwords however, regardless of whether the application is secure.

If the application is not secure and a password exists, the appropriate data set password option will be returned with a value of **xxxxxxxx** (8 uppercase Xs). If the application is secure, the actual password will be returned. The three supported password options are READ=, WRITE=, and ALTER=. You can search for these strings in the returned string (using the INDEX function) to determine if a password exists this type of access. It is the application's responsibility of retrieving these passwords—by prompting the user, for example.

An example of a data set with a WRITE password and the COMPRESS option follows:

```
TABLE OPTIONS=( CREATE=( 'WRITE=XXXXXX COMPRESS'
                )        )
```

When you use the _UPDATE_METADATA_ method, you can add passwords to an existing data set that has no password, but you cannot update an existing password. To update an existing data set password, see your administrator. Note that changes to the metadata alone can cause your metadata and data to become out of sync. You should use this functionality with extreme caution.

WHSCRFIL

Metadata type for SAS/CONNECT script files

Category: Text File Types

Parent

"WHTXTFIL" on page 269

Overview

WHSCRFIL models the metadata for SAS/CONNECT script files in SAS/Warehouse Administrator. These scripts are used to access a remote host. The location of such a script is specified as part of the host definition for a remote host. In SAS/Warehouse Administrator, to specify the location of a SAS/CONNECT script as part of a new remote host definition:

1. From the SAS/Warehouse Administrator desktop, select an environment with the right mouse button.
2. Select **Edit**.
3. From the Explorer pull-down menu, select

   File ▶ Setup

4. Select **Hosts**.
5. Click Add.
6. Enter the host information, including the SAS/CONNECT script field.

Properties

The following table lists all of the properties for WHSCRFIL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	Yes	No
Desc	C	No	Yes	No
Extended Attributes	L	No	Yes	Yes
Id	C	* Req	* Req	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	* Default	No
Note	L	No	Yes	Yes
NValue	N	No	Yes	No
Objects	L	No	No	No

Using WHSCRFIL

Add	Update	Delete
No	Yes	Yes

WHSCRFIL is a dependent type, like all subtypes of WHTFILE. To understand how all subtypes of WHTFILE relate to other types, see the process model in “Relationships Among Metadata Types” on page 53.

Use of _DELETE_METADATA for this type deletes SAS/Warehouse Administrator metadata, not the corresponding script file.

WHSERV

Base metadata type for the scheduling server

Category: Global Metadata Types

Parent

"WHROOT" on page 226

Overview

WHSERV is the base metadata type for scheduling servers in the Job Scheduler utility.

Properties

The following table lists all of the properties for WHSERV and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Icon	C	No	No	No
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No

WHSERV has the following new property:

ICON
specifies the four-level catalog entry name (such as *libref.catalog.entry.IMAGE*) of the icon associated with this object. For more information about icons, see "Using Icon Information" on page 69.

Using WHSERV

Add	Update	Delete
No	Yes	No

WHSERV is an independent type. WHSERV is not used to write metadata into a repository. It is a template for all of the scheduling servers in the Job Scheduler.

WHSRCCAT

Metadata type for SAS catalog entry source code files

Category: Text File Types

Parent

"WHTXTCAT" on page 268

Overview

WHSRCCAT is the metadata type for SAS catalog entry source code files for SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHSRCCAT and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Entry	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Full Entry	C	No	No	No
Id	C	* Req	* Req	No
Library	L	Yes	Yes	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Objects	L	No	No	No

Using WHSRCCAT

Add	Update	Delete
No	Yes	Yes

WHSRCCAT is a dependent type, like all of the subtypes of WHTFILE. To understand how all subtypes of WHTFILE relate to other types, see the process model in “Relationships Among Metadata Types” on page 53.

WHSRVAT

Metadata type for the Windows NT AT scheduling server

Category: Global Metadata Types

Parent

“WHSERV” on page 235

Overview

The WHSRVAT type models the metadata for a Windows NT AT Interface scheduling server.

Properties

The following table lists all of the properties for WHSRVAT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Command	C	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Generated Source Code	C	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Jobs	L	No	No	No
Local Work Directory	L	No	No	No
Log Filename	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Options	C	No	No	No
Print Filename	C	No	No	No
Remote Work Directory	L	No	No	No

Sysin Filename	C	No	No	No
Tracking	N	No	No	No

New properties for WHSRVAT are as follows:

COMMAND
indicates a 200-character string that the scheduling server uses to start SAS.

GENERATED SOURCE CODE
indicates a 200-character string that identifies the location where the generated source code resides.

HOST
indicates the host where the scheduling server is running. This property is a WHHOST object.

JOBS
specifies an SCL list of all of the jobs that are active on this scheduling server.

LOCAL WORK DIRECTORY
identifies the location of a directory on the local platform that is used for working storage by a local scheduling server.

LOG FILENAME
indicates a 200-character string that identifies the location where the job log resides. This follows the —LOG option in the command string.

OPTIONS
indicates a 200-character string that contains additional SAS options that are appended to the command string.

PRINT FILENAME
indicates a 200-character string that identifies the location where the job listing resides. This follows the —PRINT option in the command string.

REMOTE WORK DIRECTORY
identifies the location of a directory on a remote platform that is used for working storage by a remote scheduling server.

SYSIN FILENAME
indicates a 200-character string that identifies the location where the job sysin resides. This follows the —SYSIN option in the command string.

TRACKING
indicates a numeric indicator stating whether the scheduling server has a job track enabled. Valid values are
0—disabled
1—a job track enabled.

Using WHSRVAT

Add	Update	Delete
No	No	No

WHSRVAT is an independent type. To understand how scheduling servers relate to other types, see the diagram in Appendix 2.

WHSRVCRN

Metadata type for UNIX Cron scheduling server

Category: Global Metadata Types

Parent

"WHSERV" on page 235

Overview

The WHSRVCRN type models the metadata for a UNIX System V CRON scheduling server.

Properties

The following table lists all of the properties for WHSRVCRN and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Command	C	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Generated Source Code	C	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No

Jobs	L	No	No	No
Local Work Directory	L	No	No	No
Log Filename	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Options	C	No	No	No
Print Filename	C	No	No	No
Remote Work Directory	L	No	No	No
Sysin Filename	C	No	No	No
Tracking	N	No	No	No

New properties for WHSRVCRN are as follows:

COMMAND
indicates a 200-character string that the scheduling server uses to start SAS.

GENERATED SOURCE CODE
indicates a 200-character string that identifies the location where the generated source code resides.

HOST
indicates the host where the scheduling server is running. This property is a WHHOST object.

JOBS
specifies an SCL list of all of the jobs that are active on this scheduling server.

LOCAL WORK DIRECTORY
identifies the location of a directory on the local platform that is used for working storage by a local scheduling server.

LOG FILENAME
indicates a 200-character string that identifies the location where the job log resides. This follows the —LOG option in the command string.

OPTIONS
indicates a 200-character string that contains additional SAS options that are appended to the command string.

PRINT FILENAME
indicates a 200-character string that identifies the location where the job listing resides. This follows the —PRINT option in the command string.

REMOTE WORK DIRECTORY
identifies the location of a directory on a remote platform that is used for working storage by a remote scheduling server.

SYSIN FILENAME
indicates a 200-character string that identifies the location where the job sysin resides. This follows the —SYSIN option in the command string.

TRACKING
indicates a numeric indicator that states whether the scheduling server has a job track enabled. Valid values are
0—disabled
1—a job track enabled.

Using WHSRVCRN

Add	Update	Delete
No	No	No

WHSRVCRN is an independent type. To understand how scheduling servers relate to other types, see the metadata models on the foldout at the back of this document.

WHSRVNUL

Metadata type for the null scheduling server

Category: Global Metadata Types

Parent

"WHSERV" on page 235

Overview

The WHSRVNUL type models the metadata for a null scheduling server.

Properties

The following table lists all of the properties for WHSRVNUL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Command	C	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Generated Source Code	C	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Jobs	L	No	No	No
Local Work Directory	L	No	No	No
Log Filename	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Options	C	No	No	No
Print Filename	C	No	No	No
Remote Work Directory	L	No	No	No
Sysin Filename	C	No	No	No
Tracking	N	No	No	No

New properties for WHSRVNUL are as follows:

COMMAND
indicates a 200-character string that the scheduling server uses to start SAS.

GENERATED SOURCE CODE
indicates a 200-character string that identifies the location where the generated source code resides.

HOST
indicates the host where the scheduling server is running. This property is a WHHOST object.

JOBS
specifies an SCL list of all of the jobs that are active on this scheduling server.

LOCAL WORK DIRECTORY
identifies the location of a directory on the local platform that is used for working storage by a local scheduling server.

LOG FILENAME
indicates a 200-character string that identifies the location where the job log resides. This follows the —LOG option in the command string.

OPTIONS
indicates a 200-character string that contains additional SAS options that are appended to the command string.

PRINT FILENAME
indicates a 200-character string that identifies the location where the job listing resides. This follows the —PRINT option in the command string.

REMOTE WORK DIRECTORY
identifies the location of a directory on a remote platform that is used for working storage by a remote scheduling server.

SYSIN FILENAME
indicates a 200-character string that identifies the location where the job sysin resides. This follows the —SYSIN option in the command string.

TRACKING
specifies a numeric indicator that states whether the scheduling server has a job track enabled.
0—disabled
1—a job track enabled.

Using WHSRVNUL

Add	Update	Delete
No	No	No

WHSRVNUL is an independent type. To understand how scheduling servers relate to other types, see the metadata models on the foldout at the back of this document.

WHSUBJCT

Metadata type for subjects in a warehouse

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHSUBJCT models the metadata for a subject in a warehouse that is managed by SAS/Warehouse Administrator. A *subject* is a grouping element for data that is related to one topic within a data warehouse. Typical subjects might include Products, Sales, and Customers. In SAS/Warehouse Administrator, each subject might be

composed of a number of different data collections: SAS data sets, SAS Multidimensional Databases (MDDBs), database tables, charts, reports, or graphs.

To add a subject with the SAS/Warehouse Administrator Explorer:

1. Select a warehouse with the right mouse button.
2. Select

   Add Item ▶ Subject

3. Select the subject with the right mouse button.
4. Select **`Properties`**.
5. Enter the subject information.

Properties

The following table lists all of the properties for WHSUBJCT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	* Req	No	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	Yes	Yes
NValue	N	Yes	Yes	No
Owner	L	Yes	Yes	No

Using WHSUBJCT

Add	Update	Delete
Yes	Yes	Yes

WHSUBJCT is an independent type, like its parent, WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

WHSUBSET

Metadata type for subsetting processes associated with data mappings

Category: Process Types

Parent

"WHPRCSPR" on page 217

Overview

WHSUBSET models the metadata for processes that subset information into data mappings for use in SAS/Warehouse Administrator tables. WHSUBSET corresponds to a WHERE clause that is entered on the Where tab of the Mapping Process Properties window. Here is one way to add a subsetting process (WHERE clause) to a new data mapping in SAS/Warehouse Administrator:

1. In the Explorer, select a detail table or a data table with the right mouse button.
2. Select **Process** from the pop-up menu.
3. In the Process Editor, select the table with the right mouse button.
4. Select

   Add ▶ Inputs

5. Enter the mapping information, until you come to the Where tab.
6. Enter a WHERE clause on the Where tab.

Properties

The following table lists all of the properties for WHSUBSET and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method.

Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Host	L	No	No	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Is Active	N	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	No	No	No
Process	L	* Auto supplied	No	No
Responsibility	C	No	No	No
Source Text	L	* Req	Yes	No
Step Source Code	L	No	No	Yes

WHSUBSET has the following new properties:

INPUT OBJECTS
specifies an SCL list of general identifying information about the columns that are input to this subprocess.

INPUT SOURCES
specifies an SCL list of general identifying information about the columns that are input to this subprocess. This list must be of type WHCOLUMN or a subtype of WHCOLUMN, and it must have an appropriate relation to the main process. For WHSUBSET, this property returns the same list as INPUT OBJECTS.

OUTPUT OBJECTS
specifies a currently unused property.

OUTPUT TARGETS
specifies a currently unused property.

PROCESS
specifies an SCL list of general identifying information about the mapping process to which this subsetting process belongs. This list must be of type WHPRCMAP or a subtype of WHPRCMAP.

SOURCE TEXT
indicates an SCL list of character items that specify a WHERE clause or other subsetting code. Each item can contain a maximum of 200 characters of source code.

Using WHSUBSET

Add	Update	Delete
No	Yes	Yes

WHSUBSET is a dependent type. To understand how all subtypes of WHPROCES relate to other types, see the process models in "Relationships Among Metadata Types" on page 53.

For details about reading process information, see "Reading Process Flow Metadata" on page 62.

WHSUMDDB

Metadata type for SAS Summary MDDBs

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHSUMDDB models the metadata for a SAS Multidimensional Database summary in SAS/Warehouse Administrator. A *SAS MDDB* is a SAS table that stores data in presummarized format and in more than two dimensions; that is, it stores more than the usual down and across columns in a standard table. For example, where a

summary table might show sales in dollars for a given company by month, an MDDB could show sales in dollars for a given company by month, region, and product.

To add an MDDB with the SAS/Warehouse Administrator Explorer:

1. Select a summary group with the right mouse button.
2. Select

   Add Item ▶ Summary MDDB

3. Select the MDDB with the right mouse button.
4. Select **`Properties`**.
5. Enter the MDDB information.

Properties

The following table lists all of the properties for WHSUMDDB and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

** Req* indicates that the property is required; you must provide a value for this property when you use a given method. ** Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Administrator	L	No	No	No
Columns	L	No	No	Yes
Cvalue	C	No	No	No
Creating Job	L	No	No	Yes
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Fiscal Day of Month	N	No	No	No
Fiscal Day of Week	C	No	No	No
Fiscal Month of Year	C	No	No	No
Fiscal Time of Day	N	No	No	No
Group	L	No	No	No
Host	L	No	No	No
Icon	C	No	No	No

Id	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	No	No	No
Process	L	No	No	Yes
Table Name	C	No	No	No
Using Processes	L	No	No	No

New properties for WHSUMDDB are as follows:

COLUMNS
specifies an SCL list of general identifying information about the columns in this MDDB.

CREATING JOB
specifies a list of general identifying information about the job that creates this summary MDDB.

FISCAL DAY OF MONTH
specifies the number that indicates the start day of the month of the fiscal year. Valid numbers are between 1 and 31.

FISCAL DAY OF WEEK
indicates the character string that contains the start day of the week of the fiscal year. Valid strings range from SUNDAY through SATURDAY.

FISCAL MONTH OF YEAR
specifies the number that indicates the start month of the year in which the fiscal year begins. Valid numbers range between 1 and 12.

FISCAL TIME OF DAY
indicates the character string for the start time of day of the fiscal year.

HOST
specifies an SCL list of general identifying information about the host on which this MDDB is accessed.

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this MDDB. For more information about input objects, see “INPUT and OUTPUT Properties” on page 64.

INPUT SOURCES
specifies an SCL list of general identifying information about the sources that are input to this MDDB. For more information about input sources, see "INPUT and OUTPUT Properties" on page 64.

LIBRARY
specifies an SCL list of general identifying information about the SAS Library (WHLIBRY type) that contains this MDDB.

OUTPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are output from this MDDB. For more information about output objects, see "INPUT and OUTPUT Properties" on page 64.

OUTPUT TARGETS
specifies an SCL list of general identifying information about the targets that are output to this MDDB. For more information about output targets, see "INPUT and OUTPUT Properties" on page 64.

PROCESS
specifies an SCL list of general identifying information about the process that created this MDDB.

TABLE NAME
indicates the character string that contains the table name that is associated with this MDDB.

Using WHSUMDDB

Add	Update	Delete
No	No	No

WHSUMDDB is an independent type, like its parent, WHOBJECT. To understand how all subtypes of WHOBJECT relate to other types, see the general information model in "Relationships Among Metadata Types" on page 53.

WHSUMTBL

Metadata type for summary tables

Category: Object Types—Explorer

Parent

"WHTABLE" on page 254

Overview

WHSUMTBL models the metadata for summary tables in SAS/Warehouse Administrator. A *summary table* can be a SAS table or view or a DBMS table or view.

Each summary table corresponds to one of these time dimensions: day, week, half-month, month, quarter, or year. To add a summary table with the SAS/Warehouse Administrator Explorer:

1. Select a summary group with the right mouse button.
2. Select

   Add Item ▶ summary table

3. Select the table with the right mouse button.
4. Select **`Properties`**.
5. Enter the summary table information.

Properties

The following table lists all of the properties for WHSUMTBL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	No	No	No
Administrator	L	No	No	No
Aggregation Level	C	No	No	No
Columns	L	No	No	Yes
Creating Job	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Fiscal Day of Month	N	No	No	No
Fiscal Day of Week	C	No	No	No
Fiscal Month of Year	C	No	No	No
Fiscal Time of Day	N	No	No	No

Group	L	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	No	No	No
Physical Storage	L	No	No	Yes
Process	L	No	No	Yes
Table Name	C	No	No	No
Using Jobs	L	No	No	No

New properties for WHSUMTBL are as follows:

AGGREGATION LEVEL
specifies the character string that indicates the aggregation level of this table: DAY, WEEK, MONTH, and so on.

FISCAL DAY OF MONTH
specifies the number that indicates the start day of the month of the fiscal year. Valid numbers are between 1 and 31.

FISCAL DAY OF WEEK
specifies the character string that contains the start day of the week of the fiscal year. Valid strings range from SUNDAY through SATURDAY.

FISCAL MONTH OF YEAR
specifies the number that indicates the start month of the year in which the fiscal year starts. Valid numbers range between 1 and 12.

FISCAL TIME OF DAY
specifies the SAS datetime character string for the start time of day of the fiscal year. (A character value that is formatted with a SAS DATETIME. format.)

Property Dependencies You must define a CREATING JOB property in order to add any INPUT SOURCES to a table. If a table does not have a CREATING JOB property, then you must specify one when you add an input source to the table.

Using WHSUMTBL

Add	Update	Delete
No	No	No

WHSUMTBL is an independent type, like its parent, WHTABLE. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

WHTABLE

Base metadata type for tables

Category: Object Types—Explorer

Parent

"WHOBJECT" on page 182

Overview

WHTABLE is the base metadata type for SAS/Warehouse Administrator tables.

Properties

The following table lists all of the properties for WHTABLE and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	No	No	No
Administrator	L	No	No	No
Columns	L	No	No	Yes

Creating Job	L	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Group	L	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	No	No	No
Physical Storage	L	No	No	Yes
Process	L	No	No	Yes
Table Name	C	No	No	No
Using Jobs	L	No	No	No

New properties for WHTABLE are as follows:

ACCESS SAME AS PHYSICAL
specifies a numeric value that indicates whether the access location view information for this table is the same as the physical storage information.

0—No, the access location view information for this table is not the same as the physical storage information.

1—Yes, the access location view information for this table is the same as the physical storage information. (For SAS data sets only).

See SAS/Warehouse Administrator documentation for more details about the ACCESS SAME AS PHYSICAL option for tables that are stored as SAS data sets.

COLUMNS
specifies an SCL list of general identifying information about the columns in this table.

CREATING JOB
specifies a list of general identifying information about the job that creates this table. This property returns a WHJOB object. A valid CREATING JOB property is

required before you can add any INPUT SOURCES. If the CREATING JOB property is removed, then any work tables in the chain of INPUT SOURCES will be deleted as well.

You cannot change the CREATING JOB property for a single table that is output from a mapping with more than one output table. You can use the OUTPUT TABLES property of the WHJOB object to which the tables are to be moved to change the value of the CREATING JOB property for all output tables of a mapping simultaneously.

HOST
specifies an SCL list of general identifying information about the host on which this data is accessed.

INPUT OBJECTS
specifies an SCL list of general identifying information about the objects that are input to this table. For more information about input objects, see “INPUT and OUTPUT Properties” on page 64.

INPUT SOURCES
specifies an SCL list of general identifying information about the sources that are input to this table. For more information about input sources, see “INPUT and OUTPUT Properties” on page 64.

LIBRARY
specifies an SCL list of general identifying information about the SAS Library that contains this table.

OUTPUT OBJECTS
an SCL list of general identifying information about the objects that are output from this table. For more information about output objects, see “INPUT and OUTPUT Properties” on page 64.

OUTPUT TARGETS
an SCL list of general identifying information about the targets that are output from this table. For more information about output targets, see “INPUT and OUTPUT Properties” on page 64.

PHYSICAL STORAGE
specifies an SCL list of general identifying information about where this table is physically stored.

PROCESS
specifies an SCL list of general identifying information about the process that created this table. This list must be of type WHPRCMAN. Adding a process beneath itself is prevented.

TABLE NAME
indicates the character string—the name of the table.

USING JOBS
specifies a list of general identifying information about all the jobs that use this table as input to output tables of the job.

Using WHTABLE

Add	Update	Delete
No	No	No

WHTABLE and its children are independent types. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

WHTBLMAP

Metadata type for intermediate output tables produced by column mapping processes

Category: Object Types—Intermediate Output Tables

Parent

"WHTBLPRC" on page 259

Overview

WHTBLMAP models the metadata for intermediate output tables that are produced by column mapping processes in SAS/Warehouse Administrator. In the Process Editor, these tables are represented as text boxes that are labeled Mapping. For details about how intermediate output tables are displayed in the Process Editor, see "Reading Process Flow Metadata" on page 62.

Properties

The following table lists all of the properties for WHTBLMAP and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Access Same As Physical	N	* Auto supplied	* Auto supplied	No
Administrator	L	No	No	No
Columns	L	No	No	Yes

Creating Job	L	No	No	No
Creates Data	N	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	No	No	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	* Auto Supplied	No	No
Owner	L	No	No	No
Physical Storage	L	Yes	Yes	Yes
Process	L	* Default	No	Yes
Row Selector	L	No	Yes	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

WHTBLMAP has the following new property:

ROW SELECTOR
: specifies an SCL list of general identifying information about any row selector subprocess of the mapping that might be defined for this table. This list must be of type WHROWSEL or a subtype of WHROWSEL.

Using WHTBLMAP

Add	Update	Delete
No	Yes	Yes

For a discussion of how you can use intermediate output tables, see "Reading Process Flow Metadata" on page 62 .

WHTBLMAP is a dependent type. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

WHTBLPRC

Base metadata type for intermediate output tables produced by processes

Category: Object Types—Intermediate Output Tables

Parent

"WHTABLE" on page 254

Overview

WHTBLPRC is the base metadata type for intermediate output tables that are produced by processes in SAS/Warehouse Administrator. In the Process Editor, these tables are represented as text boxes with appropriate labels such as Mapping, User Exit, and so on. For details about how intermediate output tables are displayed in the Process Editor, see "Reading Process Flow Metadata" on page 62.

Properties

The following table lists all of the properties for WHTBLPRC and indicates how each you can use property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Same As Physical	N	No	No	No
Administrator	L	No	No	No
Columns	L	No	No	Yes
Creating Job	L	No	No	No
Creates Data	N	No	No	No

Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Group	L	No	No	No
Host	L	No	No	No
Icon	C	No	No	No
Id	C	No	No	No
Input Objects	L	No	No	No
Input Sources	L	No	No	No
Library	L	No	No	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Output Objects	L	No	No	No
Output Targets	L	No	No	No
Owner	L	No	No	No
Physical Storage	L	No	No	Yes
Process	L	No	No	Yes
Table Name	C	No	No	No
Using Jobs	L	No	No	No

WHTBLPRC has the following new property:

CREATES DATA
specifies the numeric value that indicates whether the table has output data or is just a placeholder only.

0—No, this table has no output data. It is a placeholder only. (The **This process has no output data** selection has been made on the process properties Output Data tab.) An analogy would be a DATA step that performs processing but is coded with DATA _NULL_.

1—Yes, this table has output data.

Using WHTBLPRC

Add	Update	Delete
No	No	No

For a discussion of how you can use intermediate output tables, see "Reading Process Flow Metadata" on page 62.

WHTBLPRC is a dependent type. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

WHTBLREC

Metadata type for intermediate output tables produced by record selector processes

Category: Object Types—Intermediate Output Tables

Parent

"WHTBLPRC" on page 259

Overview

WHTBLREC models the metadata for intermediate output tables that are produced by record selector processes in SAS/Warehouse Administrator. In the Process Editor, these tables are represented as text boxes that are labeled Record Selection. For details about how intermediate output tables are displayed in the Process Editor, see "Reading Process Flow Metadata" on page 62.

Properties

The following table lists all of the properties for WHTBLREC and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Access Same As Physical	N	* Auto supplied	* Auto supplied	No
Administrator	L	No	No	No
Columns	L	No	No	Yes
Creating Job	L	No	No	No
Creates Data	N	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	No	No	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	* Auto supplied	No	No
Owner	L	No	No	No
Physical Storage	L	Yes	Yes	Yes
Process	L	* Default	No	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Using WHTBLREC

Add	Update	Delete
No	Yes	Yes

For a discussion of how you can use intermediate output tables, see "Reading Process Flow Metadata" on page 62.

WHTBLREC is a dependent type. To understand how all subtypes of WHTABLE relate to other types, see the models in "Relationships Among Metadata Types" on page 53.

WHTBLUSR

Metadata type for intermediate output tables produced by user exit processes

Category: Object Types—Intermediate Output Tables

Parent

"WHTBLPRC" on page 259

Overview

WHTBLUSR models the metadata for an intermediate output table that is produced by a user exit process in SAS/Warehouse Administrator. In the Process Editor, these tables are represented as text boxes that are labeled User Exit. For details about how intermediate output tables are displayed in the Process Editor, see "Reading Process Flow Metadata" on page 62.

Properties

The following table lists all of the properties for WHTBLUSR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Access Same As Physical	N	* Auto supplied	* Auto supplied	No
Administrator	L	No	No	No

Columns	L	No	No	Yes
Creating Job	L	No	No	No
Creates Data	N	Yes	Yes	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No
Extended Attributes	L	Yes	Yes	Yes
Group	L	No	No	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	Yes	No	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	* Auto supplied	No	No
Owner	L	No	No	No
Physical Storage	L	Yes	Yes	Yes
Process	L	* Default	No	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Using WHTBLUSR

Add	Update	Delete
No	Yes	Yes

For a discussion of how you can use intermediate output tables, see “Reading Process Flow Metadata” on page 62.

WHTBLUSR is a dependent type. To understand how all subtypes of WHTABLE relate to other types, see the models in “Relationships Among Metadata Types” on page 53.

WHTBLXFR

Metadata type for intermediate output tables produced by data transfer processes

Category: Object Types—Intermediate Output Tables

Parent

"WHTBLPRC" on page 259

Overview

WHTBLXFR models the metadata for intermediate output tables that are produced by data transfer processes in SAS/Warehouse Administrator. In the Process Editor, these tables are represented as text boxes that are labeled Data Transfer. For details about how intermediate output tables are displayed in the Process Editor, see "Reading Process Flow Metadata" on page 62.

Properties

The following table lists all of the properties for WHTBLXFR and indicates how you can use each property with metadata API methods.

In the table, you can specify properties with a *Yes* in the Indirect Add column when you indirectly add one object through another, as described in the documentation for the _UPDATE_METADATA_ method. Use this approach to add a new dependent object. For details, see "Using _UPDATE_METADATA_" on page 46.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method, as described in the documentation for this method. Use this method to update properties of an existing object. For details, see "Using _UPDATE_METADATA_" on page 46.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one. * *Auto supplied* means that the property is automatically supplied; any value that you specify for such a property is ignored.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Indirect Add	Update Method	Read Method Expand Parm.
Access Same As Physical	N	* Auto supplied	* Auto supplied	No
Administrator	L	No	No	No
Columns	L	No	No	Yes
Creating Job	L	No	No	No
Creates Data	N	No	No	No
Cvalue	C	Yes	Yes	No
Desc	C	Yes	Yes	No

Extended Attributes	L	Yes	Yes	Yes
Group	L	No	No	No
Host	L	Yes	Yes	No
Icon	C	Yes	Yes	No
Id	C	* Req	* Req	No
Input Objects	L	No	No	No
Input Sources	L	Yes	Yes	No
Library	L	Yes	Yes	No
Members	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	* Default	Yes	No
Note	L	No	No	Yes
NValue	N	Yes	Yes	No
Output Objects	L	No	No	No
Output Targets	L	* Auto supplied	No	No
Owner	L	No	No	No
Physical Storage	L	Yes	Yes	Yes
Process	L	* Default	No	Yes
Table Name	C	Yes	Yes	No
Using Jobs	L	No	No	No

Using WHTBLXFR

Add	Update	Delete
No	Yes	Yes

For a discussion of how you can use intermediate output tables, see “Reading Process Flow Metadata” on page 62.

WHTBLXFR is a dependent type. To understand how all subtypes of WHTABLE relate to other types, see the models in “Relationships Among Metadata Types” on page 53.

WHTFILE

Base metadata type for text files

Category: Text File Types

Parent

"WHROOT" on page 226

Overview

WHTFILE is the base metadata type for text files in SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHTFILE and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No

Using WHTFILE

Add	Update	Delete
No	No	No

WHTFILE is a dependent type. To understand how all subtypes of WHTFILE relate to other types, see the process model in “Relationships Among Metadata Types” on page 53.

WHTXTCAT

Base metadata type for SAS catalog entry text files

Category: Text File Types

Parent

“WHTFILE” on page 267

Overview

WHTXTCAT is the base metadata type for SAS catalog entry text files in SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHTXTCAT and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Cvalue	C	No	No	No
Desc	C	No	No	No
Entry	C	No	No	No

Extended Attributes	L	No	No	Yes
Full Entry	C	No	No	No
Id	C	No	No	No
Library	L	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No

New properties for WHTXTCAT are as follows:

ENTRY
indicates the three-level name of the catalog entry that contains the text. An example would be *source.loadfile.source*.

FULL ENTRY
indicates the four-level name of the catalog entry that contains the text. An example would be *libref.source.loadfile.source*.

LIBRARY
specifies an SCL list of general identifying information about the SAS Library (instance of type WHLIBRY) that contains this catalog.

Using WHTXTCAT

Add	Update	Delete
No	No	No

WHTXTCAT is a dependent type, like all of the subtypes of WHTFILE. To understand how all subtypes of WHTFILE relate to other types, see the process model in "Relationships Among Metadata Types" on page 53.

WHTXTFIL

Base metadata type for external text files

Category: Text File Types

Parent

"WHTFILE" on page 267

Overview

WHTXTFIL is the base metadata type for external text files in SAS/Warehouse Administrator.

Properties

The following table lists all of the properties for WHTXTFIL and indicates how you can use each property with metadata API methods.

In the table, you can pass properties with a *Yes* in the Add column to the _ADD_METADATA_ method. Use this method to add a new object.

You can pass properties with a *Yes* in the Update column to the _UPDATE_METADATA_ method. Use this method to update properties of an existing object.

* *Req* indicates that the property is required; you must provide a value for this property when you use a given method. * *Default* indicates that the system will provide a default value for that property if you do not provide one.

Properties with a *Yes* in the Read Method Expand Parm column are valid with the *expand* parameter of the _GET_METADATA_ method. This method enables you to get detailed metadata about a property and its associated objects through a single method call.

Properties	Type	Add Method	Update Method	Read Method Expand Parm.
Access Method	C	No	No	No
Cvalue	C	No	No	No
Desc	C	No	No	No
Extended Attributes	L	No	No	Yes
Fileref	C	No	No	No
Id	C	No	No	No
Metadata Created	C	No	No	No
Metadata Updated	C	No	No	No
Name	C	No	No	No
Note	L	No	No	Yes
NValue	N	No	No	No
Options	L	No	No	No
Path	L	No	No	No

Using WHTXTFIL

Add	Update	Delete
No	No	No

WHTXTFIL is a dependent type, like all subtypes of WHTFILE. To understand how all subtypes of WHTFILE relate to other types, see the process model in “Relationships Among Metadata Types” on page 53.

APPENDIX

1

Sample Metadata API Code

Appendix Overview

This appendix contains SCL code that uses the metadata API to access SAS/Warehouse Administrator metadata. For longer examples, see the sample metadata API applications in the SAMPSIO.DWADDIN catalog. That catalog includes the BLDPAGE program, which reads metadata and outputs it in HTML format.

Read Metadata Code Sample

```
    * Sample Read Metadata Program
    * for SAS/Warehouse Administrator
    */

length primary_repos_id $ 8;
length sec_repos_id $ 8;
init:
   /*
    * Create an instance of the metadata API.
    */
i_api=instance(loadclass
('SASHELP.METAAPI.METAAPI.CLASS'));
l_rc=0;
   /*
    * Access the SAS/Warehouse Administrator
    * Sample repository.
    */
path="!SASROOT\whouse\dwdemo\_master";
repos_type='WHDWENV';
   /*
    * Insert the Location information into the
    * metadata list with
    * a name of LIBRARY.
```

```
    */
l_inmeta=makelist();
l_lib=makelist();
l_inmeta=insertl(l_inmeta,l_lib,-1,'LIBRARY');
   /*
    * Use the default Libname Engine to
    * access a Local Path.
    */
l_lib=insertc(l_lib,' ',-1,'ENGINE');
l_path=makelist();
l_lib=insertl(l_lib,l_path,-1,'PATH');
l_opts=makelist();
l_lib=insertl(l_lib,l_opts,-1,'OPTIONS');
l_path=insertc(l_path,path,-1);
   /*
    * Set the primary repository. If a bad
    * return code is returned,
    * then we can't continue.
    */
call send(i_api,'_SET_PRIMARY_REPOSITORY_',
l_rc,l_inmeta, repos_type,primary_repos_id,l_meta);
l_inmeta=dellist(l_inmeta,'Y');
if l_rc = 0 then do;
   /*
    * We were able to access the primary repository
    * correctly.
    */
   /*
    * Get the list of available secondary repositories
    * under this primary repository.
    *
    */
   l_reps=makelist();
   l_meta=setniteml(l_meta,l_reps,'REPOSITORIES');
   call send(i_api,'_GET_METADATA_',l_rc,l_meta);
   if l_rc = 0 then do;
      num_reps=listlen(l_reps);
      if num_reps > 0 then do;
         /*
          * If any secondary repositories, select one to set as
          * the active one.
          */
       l_sec_rep=getiteml(l_reps,1);
       call send(i_api,'_SET_SECONDARY_REPOSITORY_',l_rc,
       l_sec_rep,sec_repos_id);
          /*
           * If l_rc = 0 then sec_repos_id contains the 8
           * character repository id of this repository. This
           * id is used as the first part of any identifiers that are used
           * to access metadata in this secondary repository.
           */
       if l_rc = 0 then do;
          /*
           * Get the List of Detail Tables in the secondary
```

```
* repository.
*/
of_type=sec_repos_id||'.'||'WHDETAIL';
l_tables=makelist();
call send(i_api,'_GET_METADATA_OBJECTS_',l_rc,
of_type,l_tables);
num_tables=listlen(l_tables);
if num_tables > 0 and l_rc = 0 then do;
   table_id=nameitem(l_tables,1);
   table_name=getitemc(l_tables,1);
   put 'Processing Table: ' table_name;

     /*
      * Get the metadata about the Load Process that created
      * this table. Note that this is an example of a
      * selective query, for example preformat the input list with
      * only the properties desired.
      */

   get_all=0;
   expand=0;
   l_table_meta=makelist();
   l_table_meta=insertc(l_table_meta,table_id,-1,'ID');
   l_table_meta=insertl(l_table_meta,0,-1,'PROCESS');
   call send(i_api,'_GET_METADATA_',l_rc,l_table_meta);
   if l_rc = 0 then do;
      l_process_meta=getniteml(l_table_meta,'PROCESS');
           /*
            * It is possible that the process has not yet been
            * defined for this table. If this is the case, an
            * empty list will be returned.
            */

      if listlen(l_process_meta) > 0 then do;

           /*
            * Get all metadata known about this process by
            * issuing a _GET_METADATA_ with the get_all
            * parameter as 1.
            */

      get_all=1;
      expand=0;
      call send(i_api,'_GET_METADATA_',l_rc,
      l_process_meta,get_all,expand);
      if l_rc = 0 then do;
         /*
          * Perform some processing on the returned
          * metadata list.
          */

          end; /* if */
          else do;
             msg=getnitemc(l_rc,'MSG');
```

```
                                rc=getnitemn(l_rc,'RC');
                                put msg;
                                put 'RC=' rc;

                                end; /* else */

                             end; /* if */
                          end; /* if */
                          else do;
                             msg=getnitemc(l_rc,'MSG');
                             rc=getnitemn(l_rc,'RC');
                             put msg;
                             put 'RC=' rc;
                             end; /* else */

                             /*
                              * Delete the table metadata list and all of its
                              * sublists.
                              *
                              * NOTE: Be extremely careful when using the DELLIST
                              * with the 'Y' option.
                              */
                          l_table_meta=dellist(l_table_meta,'Y');
                          end; /* if */
                       else do;
                          if l_rc = 0 then do;
                             put 'No detail tables found.';
                             end; /* if */
                          else do;
                             msg=getnitemc(l_rc,'MSG');
                             rc=getnitemn(l_rc,'RC');
                             put msg;
                             put 'RC=' rc;
                             end; /* else */
                          end; /* else */

                       l_tables=dellist(l_tables);
                       end; /* if */
                    else do;
                       msg=getnitemc(l_rc,'MSG');
                       rc=getnitemn(l_rc,'RC');
                       put msg;
                       put 'RC=' rc;
                       end; /* else */
                    end; /* if */
                 end; /* if */
              else do;
                 msg=getnitemc(l_rc,'MSG');
                 rc=getnitemn(l_rc,'RC');
                 put msg;
                 put 'RC=' rc;
                 end; /* else */
              end; /* if */
           else do;
```

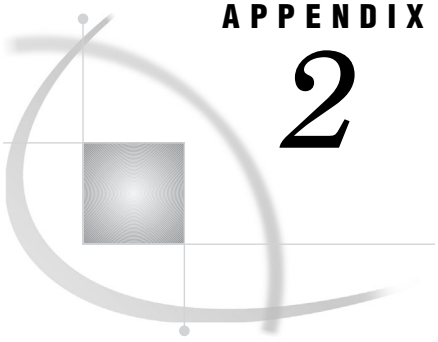
```
   end; /* else */

return;

term:
   /*
    * Make sure to _TERM_ the api object
    * so that an orderly clean up
    * is performed.
    */
call send(i_api,'_TERM_');
return;
```

Write Metadata Code Sample

```
   /* Sample Write Metadata Program
    * for SAS/Warehouse Administrator
    */
   /* Insert code to instantiate the metadata API
    * and attach to the primary and secondary
    * metadata repositories.
    */
   /*
    *   Add a new Detail Table.
    */

l_meta=makelist();

   /*
    *  Set which group to add this new table to.
    */

l_groups=makelist();
l_group=makelist();

l_groups=insertl(l_groups,l_group,-1);

l_group=insertc(l_group,group_id,-1,'ID');

l_meta=insertl(l_meta,l_groups,-1,'GROUP');

   /*
    *  Use the same repository id as the group.
    */

repos_id=scan(group_id,1,'.');

new_type=repos_id||'.WHDETAIL';

l_meta=insertc(l_meta,new_type,-1,'ID');
```

```
   /*
    *  Set the name for the display.
    */

l_meta=insertc(l_meta,'NEW TABLE',-1,'NAME');

   /*
    *  Set the desc for the display.
    */

l_meta=insertc(l_meta,'New table added
 through API',-1,'DESC');

   /*
    *  Set an icon for the display.
    */

l_meta=insertc(l_meta,'SASHELP.I0808.ADD.IMAGE',
-1,'ICON');

   /*
    *  Define a column.  The COLUMNS property
    * contains a sublist
    *  per column.
    */

l_cols=makelist();
l_col=makelist();

l_cols=insertl(l_cols,l_col,-1);

l_meta=insertl(l_meta,l_cols,-1,'COLUMNS');

col_id=repos_id||'.'||'WHCOLUMN';

l_col=insertc(l_col,col_id,-1,'ID');
l_col=insertc(l_col,'CUSTOMER',-1,'NAME');
l_col=insertc(l_col,'Name of Customer',-1,
 'DESC');
l_col=insertc(l_col,'C',-1,'TYPE');
l_col=insertn(l_col,75,-1,'LENGTH');


   /*
    *  Add any additional properties.
    *  :
    *  :
    */

   /*
    *  Add the table.
    */
```

```
call send(i_api,'_ADD_METADATA_',
    l_rc,l_meta);

if l_rc = 0 then do;

   put 'Table Added successfully';

   end; /* if */
else do;

   msg=getnitemc(l_rc,'MSG',1,1,
   'ERROR: _ADD_METADATA_ FAILED');
   put msg;

   list_rc=dellist(l_rc);

   end; /* else */


l_meta=dellist(l_meta,'Y');
```



APPENDIX

2

Metadata Type Inheritance Tree

SAS/Warehouse Administrator Metadata Type Inheritance Tree

The following figures illustrate the inheritance tree for SAS/Warehouse Administrator metadata types.

Figure A2.1 Metadata Type Inheritance Tree: Part 1

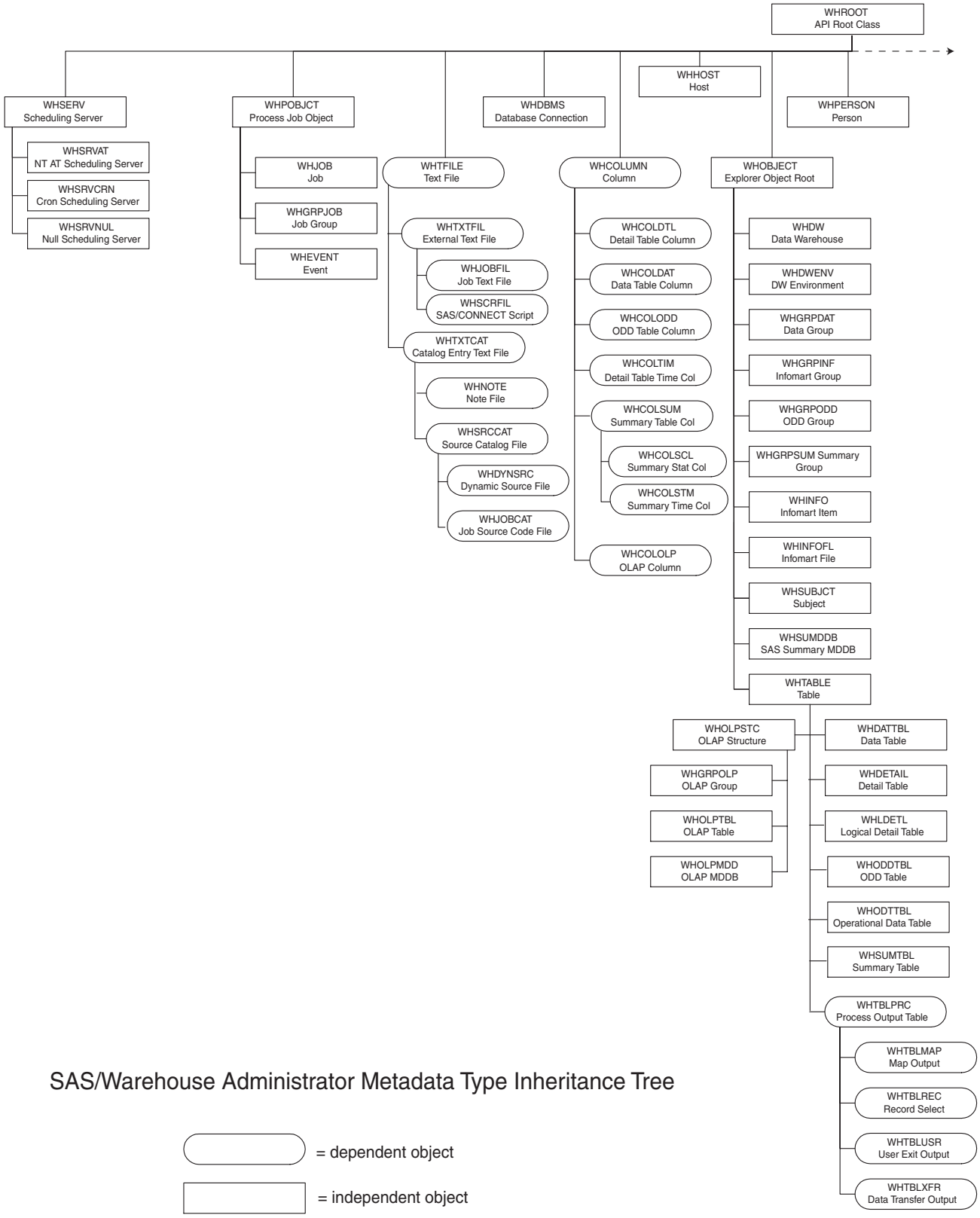
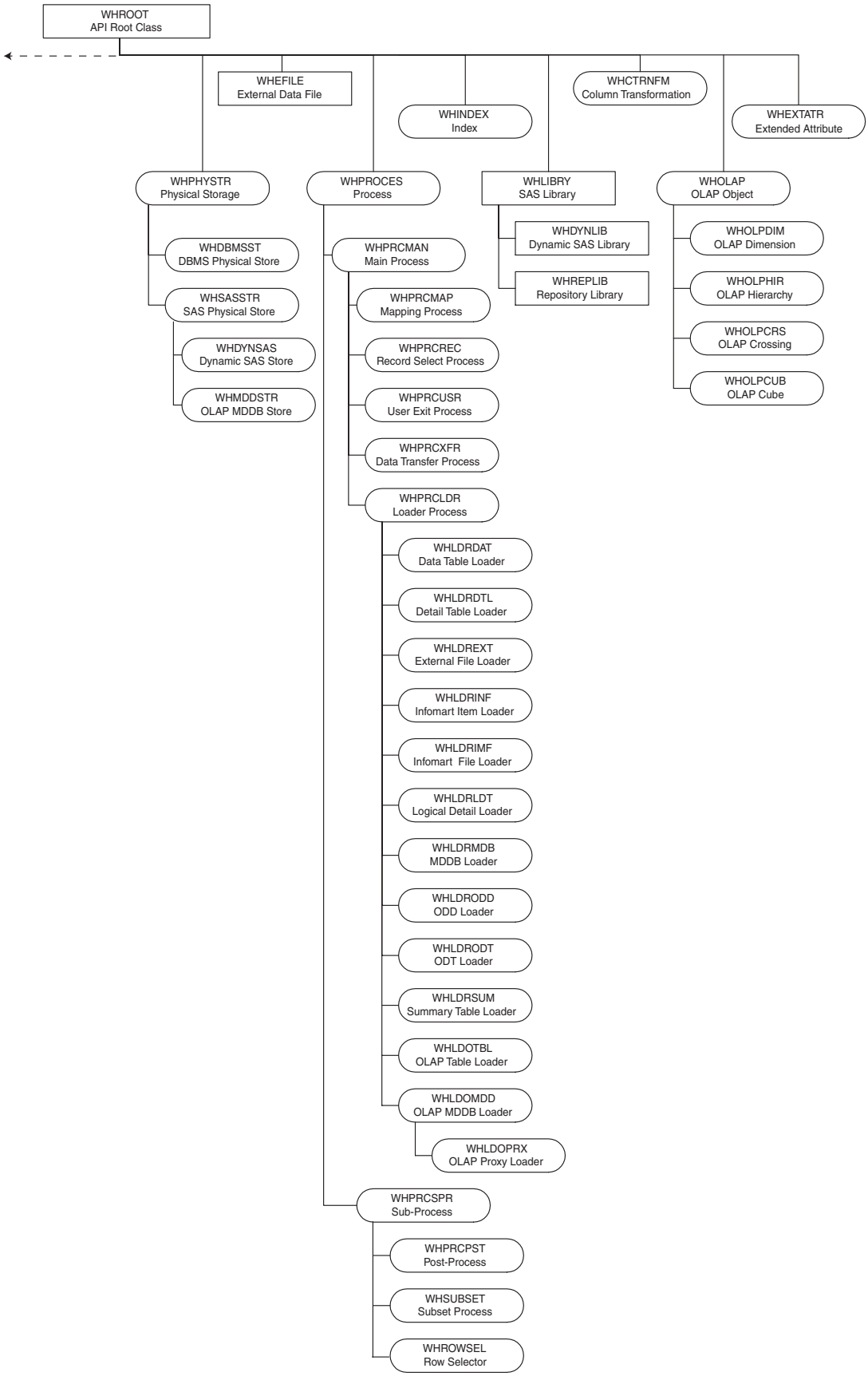
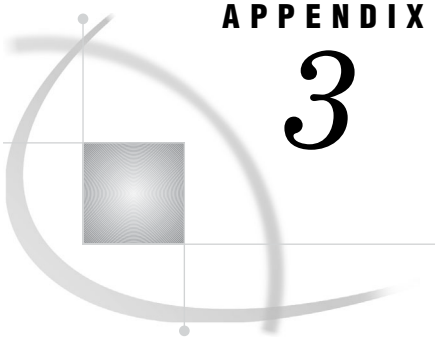


Figure A2.2 Metadata Type Inheritance Tree: Part 2





APPENDIX
3

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *Base SAS Procedures Guide*
- *Cody's Data Cleaning Techniques Using SAS Software*
- *Getting Started with the SAS System in the MVS Environment*
- *SAS/CONNECT User's Guide*
- *SAS/GRAPH Reference, Volumes 1 and 2*
- *SAS/STAT User's Guide*
- *SAS/Warehouse Administrator User's Guide*

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/pubs**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

application program interface (API) interpreter
specifies a program that translates the metadata type that is requested by a client to the corresponding metadata object in a repository.

business metadata
represents 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 who is responsible for the accuracy of the information in the table.

component
indicates a set of related metadata types. Each component has an ID, such as WHOUSE, and a name, such as SAS/Warehouse Administrator, that often matches the name of the application whose metadata is modeled by the component. WHOUSE is the component that is used for SAS/Warehouse Administrator.

contact records
indicates a metadata record that specifies the owner or administrator who is responsible for a given warehouse element. An owner is a person who formulates policy and makes decisions about an object. An administrator is a person who implements decisions that are formulated by the owner in accordance with established policy.

You can include contact records in the metadata for groups, data stores, processes, jobs, and other objects in the current environment. They are part of the global metadata for a warehouse environment.

Although contact records are not required for code generation, you might find them essential for project management. They enable you to identify—and perhaps programmatically contact—the people who are responsible for a given warehouse element.

data file
represents a metadata record that specifies a SAS file that is an input to an ODD.

If you are defining an ODD whose Load Step is a DATA step view or an SQL view (but not a Pass-Through view), you *must* define its inputs in the Process Editor. Even if your ODD does not meet the conditions above, you might still want to specify a process flow for this job for documentation purposes.

You can define a data file that simply registers the location of a SAS table or view or are that registers the location of a DBMS table with the help of a DBMS

LIBNAME definition. You can also define a data file that extracts information from a data source, saves the results to a SAS table or view, and then specifies the location of the extraction table or view.

Data files are added in the Process View of the Process Editor. In the process flow for an ODD, you can add a data file by clicking the ODD (or the background) with the right mouse button, selecting **Add**, and then **New data file**.

data group
specifies a simple grouping element for data tables, InfoMarts, and other data groups.

In the SAS/Warehouse Administrator Explorer, you can add a data group only to a subject, a data warehouse, or another data group.

You could use a data group to implement a data mart.

data table
represents a metadata record that specifies a SAS table or view or a DBMS table or view that can serve multiple purposes. Data tables are frequently used to define intermediate data stores, such as look-up tables that are included as part of a join. You can use them to define detail data stores, summary data stores (if you write your own summary code and register it as the Load Step for the data table), or tables that hold information that does not fit anywhere else.

In the SAS/Warehouse Administrator Explorer, you can add a data table to a data group.

data transfer process
specifies a metadata record that is used to generate or retrieve a routine that moves data from one host to another. Data transfers are required when an input source and the target data reside on different hosts.

If SAS/Warehouse Administrator generates the code for a data transfer, it uses SAS/CONNECT software and PROC UPLOAD or PROC DOWNLOAD to move the data. This method is most easily applied to transfers between a local host (host where SAS/Warehouse Administrator is installed) and a remote host.

If you need a remote-to-remote transfer, one solution is to specify a user-written transfer routine in the metadata for the data transfer process. The *SAS/Warehouse Administrator User's Guide* might offer other solutions for remote-to-remote data transfers.

Note: Data transfers always execute on the remote host (a host other than the host where SAS/Warehouse Administrator is installed). △

A data transfer, like a mapping, a user exit, or record selector, is inserted in the process flow for a data store.

data warehouse
represents a metadata record that specifies the SAS library _DWMD. The _DWMD library is the metadata repository for most groups and data stores in a data warehouse or a data mart at your site.

In the SAS/Warehouse Administrator Explorer, you can add a data warehouse object only to a data warehouse environment.

DBMS connection profile
represents a metadata record that specifies a user name, a password, DBMS options, and other information that SAS can use to access source data or warehouse data stores in a database management system (DBMS) other than SAS. DBMS connection profiles are included in the metadata records for DBMS data stores or DBMS LIBNAME definitions in the current environment. DBMS connection profiles are part of the global metadata for a warehouse environment.

If you want SAS/Warehouse Administrator to generate code that will access source data in a DBMS or load warehouse data in a DBMS, you will probably need at least one DBMS connection profile for each target DBMS.

If you want to connect to the same DBMS but with different levels of privilege or with different options, you need to create different DBMS connection profiles with the appropriate user names, passwords, and options.

DBMS LIBNAME definition
specifies a special SAS library definition that you can use to extract source data in DBMS format or to create warehouse data stores in a DBMS.

SAS/Warehouse Administrator uses a DBMS LIBNAME definition to generate a SAS/ACCESS LIBNAME statement. Some of the metadata that you specify in the definition corresponds to the options in the LIBNAME statement. For example, a DBMS LIBNAME definition specifies a SAS/ACCESS engine—such as Oracle or SYBASE—that enables you to access the corresponding DBMS as if it were a SAS library.

A DBMS LIBNAME definition also specifies a DBMS connection profile, which includes the DBMS user ID, password, server name, and other connection information that is used to access the DBMS. These options are passed to DBMS client software, which makes the connection to the DBMS.

DBMS LIBNAME definitions are part of the global metadata for a warehouse environment. You can use DBMS LIBNAME definitions in ODDs to access source data in DBMS format. By default, for new DBMS data stores, SAS/Warehouse Administrator generates Load Steps that use SAS/ACCESS LIBNAME statements.

For details about the SAS/ACCESS LIBNAME statement, see *SAS Language Reference: Dictionary*.

detail logical table
represents a metadata record that specifies a SAS table or view that can serve multiple purposes. A detail logical table is often used to implement a view on multiple, related detail tables. You can use detail logical tables as grouping elements for detail tables or as detail data stores.

In the SAS/Warehouse Administrator Explorer, you can add a detail logical table only to a subject. A subject can have only one detail logical table. A detail logical table can contain any number of detail tables. Detail logical tables in different subjects can share (link to) the same detail table.

detail table
indicates a metadata record that specifies a SAS table or view or a DBMS table or view that serves as a detail data store.

In the SAS/Warehouse Administrator Explorer, you can add a detail table only to a detail logical table. A detail logical table is often used to implement a view on multiple related detail tables.

event
is a metadata record that specifies a condition for controlling a job, such as checking for certain return codes or verifying the existence of a file. To use events, you must create them, include them in a job flow, then write a metadata API program that reads the job flow and generates code for it.

You can add a new event only in the Job Hierarchy view in the Process Editor. In the Job Hierarchy view, an event can only be added to a data warehouse environment, data warehouse, or a job group.

external file
specifies an input to an ODD that extracts information from one or more sources that are not in SAS format. That is, an external file is an input to an ODD whose Load Step is a DATA step view.

External files are added in the Process View of the Process Editor. In the process flow for an ODD, you can add an external file by clicking the ODD (or the background) with the right mouse button, selecting **Add**, and then **New External File**.

If you are defining an ODD whose Load Step is a DATA step view or an SQL view (but not a Pass-Through view), you *must* define its inputs in the Process Editor. Even if your ODD does not meet the conditions above, you might still want to specify a process flow for this job for documentation purposes.

host definition
indicates a metadata record that specifies a computer where data stores reside, where processes and jobs execute, or where process output is sent. Host definitions are included in the metadata records for data stores, processes, and scheduling server definitions in the current environment. Host definitions are part of the global metadata for a warehouse environment.

Host definitions are required in order to access source data and to load warehouse data stores.

InfoMart
(also called an information mart) specifies a simple grouping element for InfoMart items and InfoMart files.

Unlike most objects in SAS/Warehouse Administrator, InfoMart items and InfoMart files are used to display information rather than store it. For example, you might use an InfoMart item to display a chart that summarizes sales information from a warehouse data store. Also you might use an InfoMart file to open a spreadsheet that contains information that is useful to the person who is managing a given warehouse environment.

In the SAS/Warehouse Administrator Explorer, you can add an InfoMart only to a subject, a data group, or an ODD group.

InfoMart file
(also called an information mart file) indicates a metadata record that specifies a file other than a SAS file that you want to register in a warehouse environment. The file can be a spreadsheet, an HTML report, or any file that you can open using an external application. InfoMart file metadata describes the location of an external file and the technique for opening that file.

Unlike most objects in SAS/Warehouse Administrator, InfoMart files are used to display information rather than store it. For example, you might use an InfoMart file to open a spreadsheet that contains information that is useful to the person who is managing a given warehouse environment.

In the SAS/Warehouse Administrator Explorer, you can add an InfoMart file to an InfoMart.

InfoMart item
(also called an information mart item) indicates a metadata record that specifies a routine that generates output from detail data stores or summary data stores in a data warehouse. The output is usually a SAS chart, report, graph, or query result.

Unlike most objects in SAS/Warehouse Administrator, InfoMart items are used to display information rather than store it. For example, you might use an InfoMart item to display a chart that summarizes sales information from a warehouse data store.

In the SAS/Warehouse Administrator Explorer, you can add an InfoMart item only to an InfoMart.

job
indicates a metadata record that specifies the processes that create one or more data stores (output tables).

A job must include a process flow if SAS/Warehouse Administrator will generate the source code for the job. If you will supply the source code for a job, no process flow is required, but you might want to create one for documentation purposes.

A job might include scheduling metadata that enables the process flow or user-supplied program to be executed in batch mode at a specified date and time. A job might also include a job flow.

load process
specifies a metadata record that is used to generate or retrieve a routine that puts data into a specified target object. After you define the metadata for a given data store, you must define a load process, which creates and loads the data store.

To define a load process for a given data store, display that data store in the Process View of the Process Editor, click its icon with the right mouse button, and select **`Edit Load Step`**.

metadata
specifies information that is internal to an application that describes elements in the application, such as tables and columns. There are two main kinds of metadata: physical metadata and business metadata. See also *business metadata* and *physical metadata*.

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.

metadata application program interface (API)
specifies a set of software tools that enable programmers to write applications that access metadata. The SAS/Warehouse Administrator metadata API enables you to access metadata in SAS/Warehouse Administrator.

metadata client
indicates 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 object
represents an instance of a metadata type—the metadata for an element in an application, such as a table or column.

metadata property list
specifies a list of properties for a given metadata type that you pass to a metadata API method or a list of properties that a metadata API method returns to you.

metadata repository
indicates a data store that contains an application's 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.

ODD
(operational data definition) specifies a metadata record that provides access to data sources. The ODDs, in turn, are used as inputs to detail data stores in a warehouse environment.

At a minimum, in order for a data source to be visible in a warehouse environment, you must specify the location of that data source in an ODD. You can define an ODD that simply registers the location of a SAS table or view or one that registers the location of a DBMS table with the help of a DBMS LIBNAME definition. You can also define an ODD that extracts information from a data source, saves the results to a SAS table or view, and then specifies the location of the extraction table or view.

In the SAS/Warehouse Administrator Explorer, you can add an ODD only to an ODD Group.

ODD group
specifies a simple grouping element for ODDs. It might also contain InfoMarts.

In the SAS/Warehouse Administrator Explorer, you can add an ODD group only to a warehouse environment.

OLAP group
(online analytical processing group) organizes related summary data, which is stored in OLAP tables or OLAP MDDBs. The OLAP group properties specify the logical structure of the summarized data and how they relate to the detail data in a data warehouse. OLAP groups have a type attribute, which you specify as: ROLAP, MOLAP, HOLAP, or MIXED.

In the SAS/Warehouse Administrator Explorer, you can add an OLAP group only to a subject.

OLAP MDDB
indicates a metadata record that specifies a SAS MDDB. A SAS MDDB is not a SAS table. It is a specialized storage format that stores derived summary data in a multidimensional form, which is a highly indexed and compressed format. To load an OLAP MDDB, SAS/Warehouse Administrator generates code for the MDDB procedure, which summarizes data similar to the SUMMARY procedure.

OLAP MDDBs are the only kind of data stores in an OLAP group of type MOLAP. You can include OLAP MDDBs in an OLAP group of type HOLAP or MIXED.

Each MDDB in an OLAP group of type MOLAP must have an NWAY crossing that represents all of the data summarized to the lowest level, and it must be named *NWAY*. The MDDB can also contain additional crossings.

In the SAS/Warehouse Administrator Explorer, you can add an OLAP MDDB only to an OLAP group.

OLAP table
indicates a metadata record that specifies a file to store derived summary data. This file can be a SAS table or view or a DBMS table or view. An OLAP table can have multiple crossings.

To load an OLAP table, SAS/Warehouse Administrator generates code for the SUMMARY procedure, which summarizes data by computing descriptive statistics for columns across rows or within groups of rows.

OLAP tables are the only kind of tables in an OLAP group of type ROLAP. You can include OLAP tables in an OLAP group of type HOLAP or MIXED.

In the SAS/Warehouse Administrator Explorer, you can add an OLAP table only to an OLAP group.

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

process
specifies a metadata record that is used to generate or retrieve a routine that creates warehouse data stores extracts data, transforms data, or loads data into data stores. Mappings, user exits, data transfers, record selectors, and load steps are all processes.

Each process that you define in the Process View of the Process Editor generates or retrieves code. SAS/Warehouse Administrator can generate source code for any process except a user exit or an ODD load step. However, you can specify a user-written routine for any process.

record selector process
specifies a metadata record that is used to generate or retrieve a routine that subsets data prior to loading it to a specified table.

Note: In the current release, you can use a record selector only to subset the source data that is specified in an ODD or in a data file (which is an input to an ODD). △

A record selector process, like a mapping process, a user exit process, or a data transfer process, is inserted in the process flow for a data store.

SAS library definition
specifies a metadata record for a SAS library that contains data, views, source code, or other information that is used in the current warehouse environment. SAS library definitions are included in the metadata records for data stores, processes, and jobs in the current environment. Library definitions are part of the global metadata for a warehouse environment.

Library definitions are required in order to access source data and to load warehouse data stores.

Note: A SAS library definition does not include a host definition. In a separate task, you must create a host definition for the host where the library will reside. In the metadata for data stores and other objects, you must specify both the library definition and the host definition for the computer where the library resides. △

See also: *DBMS LIBNAME definition*.

scheduling server definitions
indicates a metadata record that specifies a scheduling server application (such as CRON under UNIX System V), a host definition for the computer where the scheduling server runs, directories where the scheduling server can send temporary files, the commands that are used to start SAS on the scheduling server host, and the default job-tracking option for jobs that use this scheduling server definition.

Scheduling server definitions are part of the global metadata for a warehouse environment. They are required if you want SAS/Warehouse Administrator to generate the code to schedule a job.

subject
specifies a grouping element for data that is related to one topic within a data warehouse. For example, a data warehouse for a company might have a subject that is called *Products* (information related to company products) or *Sales* (information related to company sales). Each subject can be composed of a number of different data collections: detail data, summary data, charts, reports, or graphs.

In the SAS/Warehouse Administrator Explorer, you can add a subject only to a data warehouse.

user exit process
specifies a metadata record that is used to retrieve a user-written routine. You must store the routine in a SAS catalog with an entry type of SOURCE or SCL. A user exit routine often extracts or transforms information for a warehouse data store, but it could do many other tasks.

A user exit, like a mapping, a data transfer, or a record selector, is inserted into the process flow for a data store.

warehouse environment
indicates a metadata record that specifies the SAS library _MASTER. The _MASTER library is the metadata repository for host definitions and other global metadata that is shared among one or more data warehouses and ODD groups.

On the SAS/Warehouse Administrator desktop, environments are displayed as icons. The default icons are green cylinders.

To open an environment in the SAS/Warehouse Administrator Explorer, on the desktop, put the cursor on the environment icon, click your right mouse button and select **`Edit`** from the pop-up menu.

In the Explorer, the environment that you opened from the desktop is the top-most object.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Your Turn

If you have comments or suggestions about *SAS/Warehouse Administrator 2.3 Metadata API Reference*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: **yourturn@sas.com**

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: **suggest@sas.com**